Also available as eBook
www.technicalpublications.org

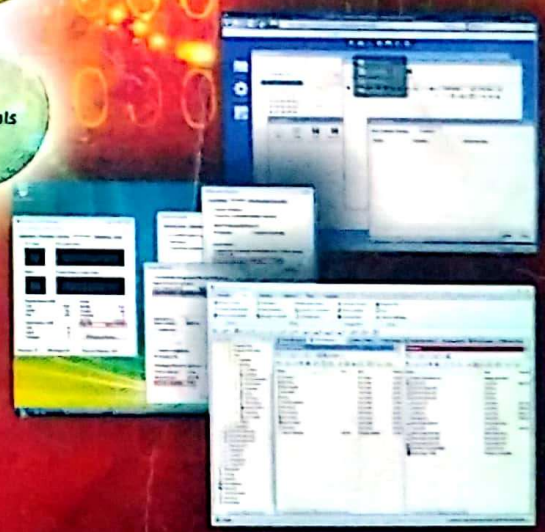
Operating System

As per New Syllabus of
Gujarat Technological University
Semester - IV (CE / CSE / IT / I&CT)









UNIX

Includes
- Solved University Questions Summer-2013 to Summer-2014

As per New Syllabus of

GUJARAT TECHNOLOGICAL UNIVERSITY

OPERATING SYSTEM

Iresh A. Dhotre

M.E. (Information Technology)
Ex-Faculty, Sinhgad College of Engineering
Pune

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

Price : ₹ 350/-

ISBN 978-93-332-0056-1

OPERATING SYSTEM

Semester - IV (CE / CSE / IT / I & CT)

First Edition : January 2015 (As per New Syllabus of Gujarat Technological University)

© Copyright with Author

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :

TECHNICAL PUBLICATIONS

Amit Residency, Office No.1, 412, Shaniwar Peth, Pune - 411030, M.S. INDIA.
Ph : +91-020-24495496/97, Telefax : +91-020-24495497
Email : technical@vtubooks.com Website : www.technicalpublications.org

Printer :

Rachana Offset
S No.15, Sukhsagar Nagar,
Waghjai Chowk, Arihant Marg,
Katraj, Pune - 411 046

Price : ₹ 350/-

ISBN 978-93-332-0056-1

ISBN 9789333200561 (Printed book)

ISBN 9789333200578 (E-Book)

eBook available on www.technicalpublications.org

PREFACE

The importance of **Operating System** is well known in vario engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Operating System.**

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
D. A. Dhotre

Dedicated to God

Syllabus (Operating System)

1. Introduction (Chapter - 1)

Basics of operating systems : Definition - Generations of operating systems - Types of operating systems. OS service. System calls. OS structure : Layered. Monolithic. Microkernel operating systems - Concept of virtual machine.

2. Process Management (Chapter - 2)

Processes : Definition , Process relationship , Process states , Process state transitions , Process control block. Context switching - Threads - Concept of multithreads. Benefits of threads - Types of threads.

Process scheduling : Definition. Scheduling objectives. Types of schedulers. Scheduling criteria : CPU utilization. Throughput. Turnaround time. Waiting time. Response time (Definition only). Scheduling algorithms : Preemptive and Non preemptive. FCFS - SJF - RR , Multiprocessor scheduling : Types. Performance evaluation of the scheduling.

3. Interprocess Communication (Chapter - 3)

Race conditions. Critical section. Mutual exclusion. Hardware solution. Strict alternation. Peterson's solution. The producer consumer problem. Semaphores. Event counters. Monitors. Message passing. Classical IPC problems : Reader's and writer problem. Dinning philosopher problem etc., Scheduling , Scheduling algorithms.

4. Deadlocks (Chapter - 4)

Definition. Deadlock characteristics. Deadlock prevention. Deadlock avoidance : Banker's algorithm. Deadlock detection and recovery.

5. Memory Management (Chapter - 5)

Basic memory management : Definition. Logical and physical address map. Memory allocation : Contiguous memory allocation - Fixed and variable partition - Internal and external fragmentation and compaction. Paging : Principle of operation - Page allocation - Hardware support for paging - Protection and sharing - Disadvantages of paging.

Virtual memory : Basics of virtual memory - Hardware and control structures - Locality of reference, Page fault , Working set, Dirty page/Dirty bit - Demand paging (Concepts only) - Page replacement policies : Optimal (OPT), First in First Out (FIFO), Second Chance (SC), Not Recently Used (NRU) and Least Recently Used (LRU).

6. I/O Management (Chapter - 6)

Principles of I/O hardware : I/O devices, Device controllers, Direct memory access, Principles of I/O software : Goals of interrupt handlers, Device drivers , Device independent I/O software, Secondary - Storage structure : Disk structure, Disk scheduling algorithm.

7. File Management (Chapter - 7)

File concept, Access methods, File types, File operation, Directory structure, File system structure, Allocation methods (Contiguous, Linked, Indexed), Free-space management (Bit vector, Linked list, Grouping), Directory implementation (Linear list, Hash table), Efficiency and performance.

8. Security and Protection (Chapter - 8)

Security environment, Design principles of security, User authentication, Protection mechanism : Protection domain, Access control list.

9. Unix/Linux Operating System (Chapters - 9, 10, 11)

Development of Unix/Linux, Role and function of Kernel, System calls, Elementary Linux command and shell programming, Directory structure, System administration case study : Linux, Windows operating system.

Table of Contents

Chapter - 1 Introduction ..... (1 - 1) to (1 - 38)

1.1 Basic of Operating System ..... 1 - 2

1.1.1 Operating System as a User Interface ..... 1 - 3

1.1.2 Operating System as Resource Manager ..... 1 - 4

1.2 Generations of Operating Systems ..... 1 - 5

1.3 Types of Operating Systems ..... 1 - 6

1.3.1 Batch Systems ..... 1 - 7

1.3.2 Multiprogramming Operating System ..... 1 - 9

1.3.3 Time Sharing System ..... 1 - 11

1.3.4 Multiprocessor System ..... 1 - 12

1.3.5 Clustered System ..... 1 - 14

1.3.6 Distributed Systems ..... 1 - 15

1.3.7 Network Operating System ..... 1 - 16

1.3.8 Real Time Operating Systems ..... 1 - 17

1.3.8.1 Difference between Hard and Soft Real Time System. ..... 1 - 19

1.3.9 Interactive System. ..... 1 - 19

1.4 Operating System Services ..... 1 - 19

1.5 System Calls ..... 1 - 21

1.5.1 Classification of System Calls ..... 1 - 23

1.6 Operating System Structure ..... 1 - 24

1.6.1 Simple Structure ..... 1 - 24

1.6.2 Layered Approach ..... 1 - 25

1.7 Kernel ..... 1 - 26

1.7.1 Monolithic Kernel ..... 1 - 28

1.7.2 Microkernel ..... 1 - 28

1.7.3 Comparison between Monolithic Kernel and Microkernel . . . . . 1 - 30
1.7.4 Reentrant Kernel . . . . . 1 - 30
1.8 Design and Implementation of OS . . . . . 1 - 31
1.9 Operations of Operating System . . . . . 1 - 32
1.9.1 Dual Mode Operation . . . . . 1 - 33
1.10 System Programs . . . . . 1 - 33
1.11 Concept of Virtual Machine . . . . . 1 - 34

Chapter - 2 Process Management **(2 - 1) to (2 - 58)**

2.1 Process . . . . . 2 - 2
2.1.1 Difference between Process and Program . . . . . 2 - 3
2.1.2 Process Control Block . . . . . 2 - 3
2.2 Process States . . . . . 2 - 5
2.3 Process Description . . . . . 2 - 7
2.4 Process Control . . . . . 2 - 9
2.4.1 Process Creation . . . . . 2 - 9
2.4.2 Process Termination . . . . . 2 - 10
2.5 Execution of the Operating System . . . . . 2 - 11
2.6 Threads . . . . . 2 - 12
2.6.1 Thread Advantages . . . . . 2 - 13
2.6.2 Difference between Thread and Process . . . . . 2 - 13
2.6.3 Thread Lifecycle . . . . . 2 - 14
2.7 Types of Threads . . . . . 2 - 15
2.7.1 User Level Thread . . . . . 2 - 15
2.7.2 Kernel Level Thread . . . . . 2 - 16
2.7.3 Difference between User Level and Kernel Level Thread . . . . . 2 - 17
2.8 Concept of Multithreads . . . . . 2 - 17
2.9 Types of Scheduling . . . . . 2 - 20
2.9.1 Schedulers . . . . . 2 - 20
2.9.2 Difference between Long Term, Short Term and Medium Term Scheduler . . 2 - 22
2.9.3 Context Switch . . . . . 2 - 22

2.10.2 Difference between Preemptive and Non-preemptive Scheduling ..... 2 - 25
2.10.3 CPU Scheduling Criteria ..... 2 - 25
2.11 Scheduling Algorithms ..... 2 - 26
2.11.1 First Come First Serve Scheduling ..... 2 - 26
2.11.2 Shortest Job First Scheduling ..... 2 - 28
2.11.3 Priority Scheduling ..... 2 - 29
2.11.4 Round Robin Scheduling ..... 2 - 30
2.11.5 Comparison between FCFS and RR ..... 2 - 32
2.11.6 Comparison of CPU Scheduling Algorithm ..... 2 - 32
2.12 Multiprocessor Scheduling ..... 2 - 48
2.12.1 Issue Relating to the Scheduling ..... 2 - 48
2.12.2 Various Design Issues ..... 2 - 49
2.12.3 Thread Scheduling ..... 2 - 50
2.13 Performance Evaluation of the Scheduling ..... 2 - 52
2.13.1 Deterministic Modeling ..... 2 - 53
2.13.2 Queueing Models ..... 2 - 54
2.13.3 Simulations ..... 2 - 55

Chapter - 3 Interprocess Communication (3 - 1) to (3 - 40)

3.1 Introduction of Concurrent Processes ..... 3 - 2
3.1.1 Principle of Concurrency ..... 3 - 2
3.1.2 Race Condition ..... 3 - 3
3.1.3 Critical Section Problem ..... 3 - 4
3.1.4 Critical Region ..... 3 - 6
3.1.5 Mutual Exclusion ..... 3 - 6
3.2 Hardware Solution ..... 3 - 7
3.2.1 Test and Set Operations ..... 3 - 7
3.3 Strict Alternation ..... 3 - 8
3.4 Lock ..... 3 - 10
3.5 Mutex ..... 3 - 10

3.6 Peterson's Solution ..... 3 - 10
3.7 Producer - Consumer Problem ..... 3 - 11
3.8 Semaphore ..... 3 - 15
3.8.1 Binary Semaphore ..... 3 - 16
3.8.2 Busy Waiting ..... 3 - 16
3.8.3 Drawbacks of Semaphore ..... 3 - 17
3.8.4 Semaphore Programming ..... 3 - 17
3.9 Dekker's Solution ..... 3 - 20
3.10 Classical IPC Problem ..... 3 - 22
3.10.1 Dinning Philosopher Problem ..... 3 - 22
3.10.2 Reader-Writer Problem ..... 3 - 24
3.11 Monitors ..... 3 - 24
3.11.1 Drawbacks of Monitors ..... 3 - 29
3.11.2 Difference between Monitors and Semaphore ..... 3 - 29
3.12 Inter Process Communication Models ..... 3 - 30
3.12.1 Pipes ..... 3 - 32
3.12.2 Features of Message Passing ..... 3 - 36
3.12.3 IPC Message Format ..... 3 - 36
3.12.4 IPC Synchronization ..... 3 - 38
3.12.5 Shared Memory ..... 3 - 39

Chapter - 4 Deadlock ..... (4 - 1) to (4 - 24)

4.1 Introduction to Deadlock ..... 4 - 2
4.1.1 Resource Allocation Graphs ..... 4 - 3
4.2 Deadlock Characteristics ..... 4 - 6
4.2.1 Necessary Condition for Deadlock ..... 4 - 6
4.3 Deadlock Solution ..... 4 - 6
4.4 Deadlock Prevention ..... 4 - 7
4.5 Deadlock Avoidance ..... 4 - 7
4.5.1 Banker's Algorithm ..... 4 - 8
4.6 Deadlock Detection ..... 4 - 17

4.6.1 Wait for Graph ..... 4 - 17
4.7 Deadlock Recovery ..... 4 - 18
4.8 Comparison between Detection, Prevention and Avoidance Methods of Handling Deadlock ..... 4 - 19

Chapter - 5 Memory Management ..... **(5 - 1) to (5 - 62)**

5.1 Basic Memory Management ..... 5 - 2
5.1.1 Memory Management Function ..... 5 - 2
5.1.2 Basic Hardware of Memory ..... 5 - 2
5.1.3 Address Space Mapping ..... 5 - 3
5.1.4 Concept of Memory Address ..... 5 - 4
5.1.5 Dynamic Loading ..... 5 - 5
5.2 Swapping ..... 5 - 6
5.3 Contiguous Memory Allocation with Fixed Partitions ..... 5 - 7
5.3.1 Dynamic Memory Partitions ..... 5 - 8
5.3.2 Memory Protection and Mapping ..... 5 - 9
5.4 Contiguous Memory Allocation with Variable Partitions ..... 5 - 11
5.4.1 Difference between Contiguous and Noncontiguous Memory Allocation ..... 5 - 12
5.4.2 Fragmentation ..... 5 - 13
5.4.3 Difference between Internal and External Fragmentation ..... 5 - 13
5.4.4 Compaction ..... 5 - 14
5.4.5 Placement Algorithms ..... 5 - 15
5.5 Paging ..... 5 - 18
5.5.1 Protection and Sharing ..... 5 - 22
5.5.2 Paging with TLB ..... 5 - 23
5.5.3 Page Table Structure ..... 5 - 24
5.5.3.1 Multilevel or Hierarchical Page Table ..... 5 - 24
5.5.3.2 Hashed Page Tables ..... 5 - 26
5.5.3.3 Inverted Page Table ..... 5 - 27
5.5.4 Advantages of Paging ..... 5 - 27
5.5.5 Disadvantages of Paging ..... 5 - 27
5.6 Segmentation ..... 5 - 27

5.6.1 Protection and Sharing . . . . . 5 - 29
5.6.2 Advantages . . . . . 5 - 30
5.6.3 Disadvantages . . . . . 5 - 30
5.6.4 Difference between Segmentation and Paging . . . . . 5 - 30
5.7 Paged Segmentation . . . . . 5 - 31
5.8 Virtual Memory Concepts . . . . . 5 - 34
5.9 Demand Paging . . . . . 5 - 36
5.9.1 Steps in Handling a Page Fault . . . . . 5 - 38
5.9.2 Demand Paging Performance . . . . . 5 - 39
5.9.3 Advantages of Demand Paging . . . . . 5 - 39
5.9.4 Disadvantages of Demand Paging . . . . . 5 - 39
5.9.5 Comparison of Demand Paging with Segmentation . . . . . 5 - 39
5.10 Page Replacement Algorithms . . . . . 5 - 40
5.10.1 First-In-First-Out . . . . . 5 - 40
5.10.1.1 Belady's Anomaly . . . . . 5 - 41
5.10.2 LRU Page Replacement Algorithm . . . . . 5 - 42
5.10.3 LRU Approximation Algorithms . . . . . 5 - 44
5.10.4 Optimal Page Replacement . . . . . 5 - 45
5.10.5 Difference between FIFO, LRU and Optimal . . . . . 5 - 46
5.10.6 Second Chance Algorithm . . . . . 5 - 57
5.10.7 Not Recently Used . . . . . 5 - 59
5.11 Thrashing . . . . . 5 - 60
5.11.1 Working Set Model . . . . . 5 - 60

Chapter - 6 I/O Management (6 - 1) to (6 - 44)

6.1 Principles of Input/Output Hardware . . . . . 6 - 2
6.1.1 I/O Devices . . . . . 6 - 2
6.1.2 Device Controllers . . . . . 6 - 3
6.1.3 Memory Mapped I/O . . . . . 6 - 4
6.1.4 Direct Memory Access . . . . . 6 - 6
6.2 Principles of I/O Software . . . . . 6 - 9
6.2.1 Goals of I/O Software . . . . . 6 - 9

6.2.2 Programmed I/O ..... 6 - 11
6.3 I/O Software Layers ..... 6 - 12
6.3.1 Interrupt Handlers ..... 6 - 13
6.3.2 Device Drivers ..... 6 - 14
6.3.2.1 Functions of Device Drivers ..... 6 - 16
6.3.2.2 Character Device Driver ..... 6 - 16
6.3.3 Device Independent I/O Software ..... 6 - 18
6.3.3.1 Buffering ..... 6 - 19
6.4 Disks ..... 6 - 21
6.4.1 Characteristics of Storage Device ..... 6 - 22
6.4.2 Hard Disk ..... 6 - 22
6.5 RAID ..... 6 - 27
6.5.1 Software RAID ..... 6 - 28
6.5.2 Hardware RAID ..... 6 - 28
6.5.3 RAID Array Components ..... 6 - 28
6.5.4 RAID Levels ..... 6 - 29
6.5.5 Comparison of RAID Levels ..... 6 - 33
6.6 Disk Performance Parameters ..... 6 - 34
6.7 Disk ARM Scheduling Algorithm ..... 6 - 35
6.7.1 First In First Out ..... 6 - 36
6.7.2 Shortest Seek Time First ..... 6 - 37
6.7.3 SCAN ..... 6 - 39
6.7.4 Circular SCAN ..... 6 - 40
6.7.5 LOOK Scheduling ..... 6 - 42
6.7.6 C-LOOK ..... 6 - 43

Chapter - 7 File Management **(7 - 1) to (7 - 42)**

7.1 Basic File Concept ..... 7 - 2
7.1.1 File Naming ..... 7 - 2
7.1.2 Types of File ..... 7 - 3
7.1.2.1 File Types ..... 7 - 3
7.1.3 File Attributes ..... 7 - 3
7.1.4 Operation on File ..... 7 - 5
..... 7 - 6

6 - 11
6 - 12
6 - 13
6 - 14
6 - 16
6 - 16
- 18
- 19
- 21
- 22
- 22
- 27
- 28
- 28
- 28
- 29
- 33
34
35
36
7
9
0
2
3

7.1.5 File Structure ..... 7 - 7
7.2 File Access Methods ..... 7 - 8
7.2.1 Sequential Access Method ..... 7 - 9
7.2.2 Direct Access Method ..... 7 - 9
7.2.3 Indexed Sequential Access ..... 7 - 10
7.3 File Directory and Disk Structure ..... 7 - 11
7.3.1 Single Level Directory ..... 7 - 12
7.3.2 Hierarchical Directory System ..... 7 - 13
7.3.2.1 Two-level Directory Structure ..... 7 - 13
7.3.2.2 Tree Structured Directory ..... 7 - 14
7.3.2.3 Acyclic Graph Directory ..... 7 - 15
7.4 File System Mounting ..... 7 - 17
7.5 File Sharing ..... 7 - 19
7.6 Record Blocking ..... 7 - 23
7.7 File System Structure ..... 7 - 28
7.8 File System Implementation ..... 7 - 29
7.9 Log Structured File Systems ..... 7 - 30
7.10 Virtual File Systems ..... 7 - 31
7.11 Directory Implementation ..... 7 - 32
7.12 Allocation Methods ..... 7 - 32
7.12.1 Contiguous Allocation ..... 7 - 33
7.12.2 Linked Allocation ..... 7 - 35
7.12.3 Indexed Allocation ..... 7 - 37
7.13 Free Space Management ..... 7 - 38
7.14 Efficiency and Performance ..... 7 - 41
7.14.1 Efficiency ..... 7 - 41
7.14.2 Performance ..... 7 - 41

Chapter - 8 Security and Protection (8 - 1) to (8 - 14)
8.1 Security ..... 8 - 2

8.1.1 Security Goals ..... 8 - 3
8.1.2 Policy and Mechanism ..... 8 - 5
8.1.3 Security Terminology ..... 8 - 5
8.1.4 Intruders ..... 8 - 5
8.1.5 Security Design Principles ..... 8 - 6
8.2 User Authentication ..... 8 - 7
8.2.1 Password Based Authentication ..... 8 - 8
8.3 Protection Mechanisms ..... 8 - 9
8.3.1 Access Control Matrix ..... 8 - 10
8.3.2 Access Control Lists ..... 8 - 11
8.3.3 Implementation of Access Matrix ..... 8 - 11

Chapter - 9 UNIX Operating System ..... (9 - 1) to (9 - 58)

9.1 History of UNIX ..... 9 - 2
9.1.1 UNIX Features ..... 9 - 3
9.1.2 General UNIX Architecture ..... 9 - 4
9.1.3 Architecture of UNIX OS ..... 9 - 5
9.1.4 Modern UNIX Kernel ..... 9 - 6
9.2 Shell ..... 9 - 7
9.2.1 Shell's Responsibilities ..... 9 - 8
9.3 UNIX Shell ..... 9 - 9
9.3.1 Bourne Shell ..... 9 - 10
9.3.2 C Shell ..... 9 - 10
9.3.3 Bash ..... 9 - 11
9.3.4 Shell as a Programming Language ..... 9 - 11
9.4 Interfaces to UNIX ..... 9 - 12
9.5 UNIX Utility Programs ..... 9 - 13
9.5.1 Internal and External Commands ..... 9 - 13
9.5.2 File Handling Utilities ..... 9 - 14
9.5.3 Security File Permissions ..... 9 - 17
9.5.4 Process Utilities ..... 9 - 19
9.5.5 Disk Utilities ..... 9 - 22

9.6 UNIX File System ..... 9 - 25
9.6.1 File Types ..... 9 - 25
9.6.2 File System Structure ..... 9 - 26
9.6.3 Directories ..... 9 - 28
9.6.3.1 Special Directories ..... 9 - 30
9.6.4 Files and Devices ..... 9 - 31
9.6.5 File Descriptor ..... 9 - 32
9.6.6 Inode ..... 9 - 33
9.6.7 The U Area ..... 9 - 35
9.6.8 File Allocation ..... 9 - 38
9.7 vi Editor ..... 9 - 39
9.8 Shell Meta Characters ..... 9 - 40
9.9 Environment Variable ..... 9 - 42
9.9.1 Shell Variables ..... 9 - 44
9.9.2 Difference between Shell Variables and Environment Variables ..... 9 - 45
9.10 Control Structure ..... 9 - 46
9.10.1 If Statements ..... 9 - 46
9.10.2 Case Statements ..... 9 - 46
9.10.3 For Loops ..... 9 - 47
9.10.4 While Loops ..... 9 - 49
9.10.5 Until Loops ..... 9 - 50
9.11 Command Substitution ..... 9 - 51
9.12 Debugging Shell Scripts ..... 9 - 51
9.12.1 Running Programs on UNIX ..... 9 - 52
9.13 Shell Script ..... 9 - 52

Chapter - 10 LINUX Operating System (10 - 1) to (10 - 12)

10.1 Introduction to Linux ..... 10 - 2
10.1.1 Components of Linux System ..... 10 - 2
10.2 System Administration ..... 10 - 3
10.2.1 Requirements for a Linux System Administrator ..... 10 - 4
10.2.2 Adding Users : Basic Steps ..... 10 - 4

10.2.3 Adding Users with useradd ..... 10 - 7
10.2.4 Removing User ..... 10 - 12

Chapter - 11 Windows Operating System **(11 - 1) to (11 - 12)**

11.1 History of Windows OS ..... 11 - 2
1.11.1 Microsoft Windows CE ..... 11 - 3
11.2 Design Principles ..... 11 - 4
11.3 Windows Architecture ..... 11 - 5
11.4 File System ..... 11 - 10
11.4.1 NTFS ..... 11 - 10

1 Introduction

Syllabus

Basics of operating systems : Definition - Generations of operating systems - Types of operating systems, OS service, System calls, OS structure : Layered, Monolithic, Microkernel operating systems - Concept of virtual machine

Contents

1.1 Basic of Operating System . . . . . **Winter-13, Summer-14,** · · Marks 7

1.2 Generations of Operating Systems

1.3 Types of Operating Systems . . . . . **Winter-13, Summer-13,** · · · Marks 6

1.4 Operating System Services

1.5 System Calls

1.6 Operating System Structure

1.7 Kernel

1.8 Design and Implementation of OS

1.9 Operations of Operating System

1.10 System Programs

1.11 Concept of Virtual Machine



1.1 Basic of Operating System

GTU : Winter-13, Summer-14

- **OS definition :** *Operating System is a program that controls the execution of application Programs. It is an interface between applications and hardware.*
- OS provides different types of view. For user, it is abstract view because it provides features which are important for users. OS is intermediary between user and the computer system.
- The major design goals/ functions of an operating system are :
  1. Efficient use of a computer system 2. User convenience 3. Ability to evolve
- An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.
- Efficiency is the one of the parameter for use of computer system. Operating system consumes some resources during its own operation. For example it uses CPU and memory. CPU is busy with scheduling and memory is occupied by instruction and required data.
- This is one type of overhead and because of this lesser resources are available to the user.
- An OS makes a computer more convenient to use. If the operating system can not allocates the free available resources to program or it over allocates the resources then efficiency is poor.

Efficient use

- For efficient use of resources, it must be monitored by operating system. Proper scheduling of resources is also required.
- Computer contains different type's resources like CPU, memory and I/O device etc. Proper monitoring is required on these resources to avoid the overhead. As per the resource, scheduling is required.
- Special attention to be given for CPU and memory. If memory is not free then user can not load new program into the memory. Then CPU will be busy with memory management.

User convenience

- User convenience is affected by computing environment of the computer system. The computing environment is comprised of computer system, its interfaces with other systems and nature of computations performed by its users.
- Computer architecture and use change the computing environment of the system. Following factors are considered while considering user convenience :

1. Good service 2. Ease of use 3. New programming model
4. Evolution 5. User friendly OS

Ability to evolve

- An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service.
- Task performed by operating systems :
  1. Maintain the list of resources in the system
  2. Maintain the list of authorized users
  3. Initiate the execution of programs and process
  4. Maintain resource usage list
  5. Maintain the resource allocated list
  6. Scheduling of resources (CPU, Secondary storage etc.)
  7. Also maintain the protection information.

1.1.1 Operating System as a User Interface

- Computer system consists of software and hardware to solve specific problems. User, application program, operating system and the hardware are the components of the computer systems.
- Application program used to solve specific program. Student attendance monitoring is the example of application program.
- Operating system is a subset of the system software. OS interacts directly with the hardware to provide an interface to other system software.
- System software and hardware exist to support the creation and effective use application software.
- Fig. 1.1.1 shows conceptual view of a computer system. (See Fig. 1.1.1 on next page).
- Resource sharing and resource abstraction are two key aspects of the operating system.
- Purposes of the operating system :

1. OS provides an interface between the computer hardware and computer user (programmer). It simplifies the programmer job like editing, coding, creation etc.
2. Allocation and use of computer resources among the programmer is controlled by OS.

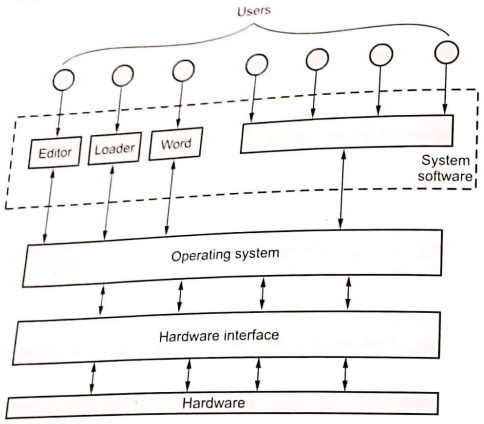


Fig. 1.1.1 Computer system conceptual view

1.2 Operating System as Resource Manager

A computer is a set of resources. These resource provides various functions to the user. Functions like data movement, storing of data and program, operation on data are control by an operating system.

Fig. 1.1.2 shows OS as a resource manager.

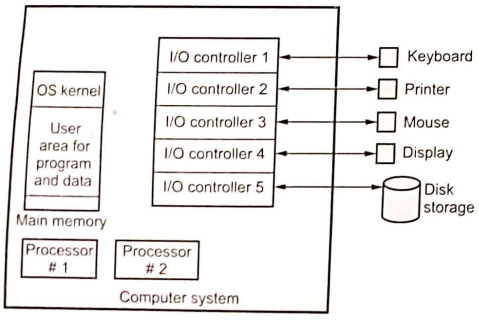


Fig. 1.1.2 OS as a resource manager

- The operating system is responsible for managing the all resources. A portion of the OS is in main memory. This portion of the OS is called kernel.
- User program and data is also stored in remaining parts of the memory. Allocation of main memory is controlled by operating system with the help of memory management hardware.
- I/O device is controlled by OS and it decides when an I/O device can be used by program in execution. Processor is one type of resource and OS control the execution of user program on the processor.
- Modern OS allows multiple programs to run at the same time. If multiple users are using computer then there is need of managing and protecting the memory, I/O devices and other resources.
- Resource management includes sharing resources in different ways. Time and space are the two concept for resource sharing.
  1. **Time :** Time slot is allocated to each program first one gets to use the resource then another and so on.
  2. **Space :** Consider the example of main memory. Main memory is normally divided up among several running programs, so each one can be resident at the same time.

University Questions

1. *What is operating system ? Give functions of operating system.* GTU : Winter-13, Marks 4
2. *Explain the objectives and functions of operating systems.* GTU : Summer-14, Marks 7

1.2 Generations of Operating Systems

- Generation of operating system according to the requirement is possible. Generation of OS for one machine includes design, coding and implementation. Now a day, operating system runs on any type of computer architecture with various configuration.
- System generation is the process of generation of OS source code. Sysgen stands for YStem GENeration. It is a utility that enables an operating system to configure hardware and software setups properly.
- SYSGEN program obtains information concerning the specific configuration of the hardware system.
- CD-ROM, DVD and disk are normally used to distribute the operating system installable code. Sysgen program reads from a given file or it ask to the user to provide information before installation. It determines following information :

1. Type of processor used in the system. It also checks for register set, instruction set, floating point arithmetic, interrupts etc.
2. It checks for disk partitions and type of format.
3. Main memory size
4. Which and how may I/O devices connected to the system? It also read details specification of the I/O devices.
5. While installing operating system, it supports many options. It includes buffer size, CPU scheduling method, size of cache memory etc.

- All these information's are complied before installation. System admin can change or modify a copy of the source code of the operating system. After this operating system is completely compiled. It produces an output of the operating system. This generated operating system considers declarations, initializations of data and constant.

1.3 Types of Operating Systems

GTU : Winter-13, Summer-13

- Types of an operating system are a grouping that differentiates or identifies the operating system based on how it works the type of hardware it controls and the applications it supports.

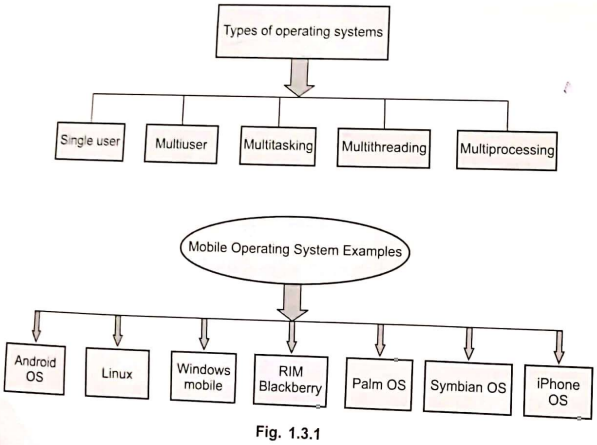


Fig. 1.3.1

1.3.1 Batch Systems

- Batch system process a collection of jobs, called a batch. Batch is a sequence of user jobs.
- Job is a predefined sequence of commands, programs and data that are combined into a single unit.
- Each job in the batch is independent of other jobs in the batch. A user can define a job control specification by constructing a file with a sequence of commands.
- Jobs with similar needs were batched together to speed up processing. Card readers and tape drives are the input device in batch systems. Output devices are tape drives, card punches and line printers.
- Primary function of the batch system is to service the jobs in a batch one after another without requiring the operator's intervention. There is no need for human/user interaction with the job when it runs, since all the information required to complete job is kept in files.
- Some computer systems only did one thing at a time. They had a list of instructions to carry out and these would be carried out one after the other. This is called a **serial system.** The mechanics of development and preparation of programs in such environments are quite slow and numerous manual operations involved in the process.
- **Batch monitor** is used to implement batch processing system. Batch monitor is also called **kernel**. Kernel resides in one part of the computer main memory.
- The memory allocator managed the main memory space. Batch monitor controls the sequence of events. Main memory store the batch monitor and users program and data (jobs). Fig. 1.3.2 shows memory layout for a batch system.

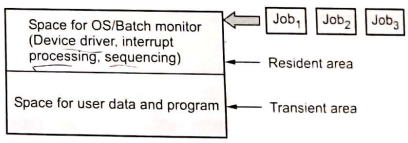


Fig. 1.3.2 Memory layout for batch system

- Computer operator gives a command to start the processing of a batch, the kernel sets up the processing of the first job. Job was selected from the job queue and loaded into main memory. When a job completed execution, its memory was released and the output for the job was copied.
- When a job is completed, it returns control to the monitor, which immediately reads in the next job.

- Fig. 1.3.3 shows concept of batch system.

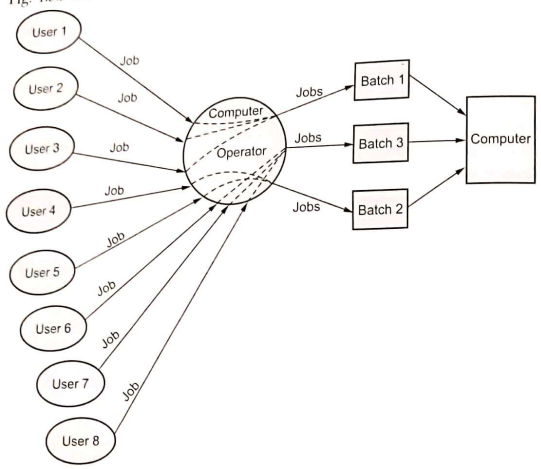


Fig. 1.3.3 Concept of batch system

- Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e. first come first served fashion.
- When a job completes execution, its memory is released and the output for the job gets copied into an output **spool** for later printing.
- Spooling an acronym for **simultaneous peripheral operation on line.** Spooling uses the disk as a large buffer for outputting data to printers and other devices. It can also be used for input, but is generally used for output. Its main use is to prevent two users from alternating printing lines to the line printer on the same page, getting their output completely mixed together. It also helps in reducing idle time and overlapped I/O and CPU.
- Batch system often provides simple forms of file management. Access to file is serial. Batch systems do not require any time critical device management.
- Batch systems are inconvenient for users because users cannot interact with their jobs to fix problems. There may also be long turnaround times. Example of this system is generating monthly bank statement.

Spooling

- Spooling refers to putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready.
- Spooling is useful because device access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up.
- Computer can perform I/O in parallel with computation, it becomes possible to have the computer read a deck of cards to a tape, drum or disk and to write out to a tape printer while it was computing. This process is called **spooling.**
- The most common spooling application is print spooling.
- Spooling batch system were the first and are the simplest of the multiprogramming systems.

Advantages of spooling

1. The spooling operation uses a disk as a very large buffer.
2. Spooling is however capable of overlapping I/O operation for one job with processor operations for another job.

Advantages of batch system

1. Move much of the work of the operator to the computer.
2. Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of batch system

1. Turn around time can be large from user standpoint.
2. Program debugging is difficult.
3. There was possibility of entering jobs in infinite loop.
4. A job could corrupt the monitor, thus affecting pending jobs.
5. Due to lack of protection scheme, one batch job can affect pending jobs.

1.3.2 Multiprogramming Operating System

- CPU remains idle in batch system. At any time either CPU or I/O device was idle in batch system. To keep CPU busy, more than one program/job must be loaded for execution. It increases the CPU utilizations. So multiprogramming increases the CPU utilization.
- Resource management is the main aim of multiprogramming operating system. File system, command processor, I/O control system and transient area are the essential components of a single user operating system. Multiprogramming

operating system divides the transient area to store the multiple programs and provides resource management to the operating system.

- The concurrent execution of programs improves the utilization of system resources. A program in execution is called a "Process", "Job" or a "Task".
- When two or more programs are in the memory at the same time, sharing the processor is referred to the multiprogramming operating system. Fig. 1.3.4 shows the memory layout for a multiprogramming operating system.

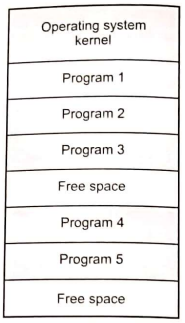


Fig. 1.3.4 Multiprogramming OS memory view

- Operating system keeps number of programs into the memory. It selects one program from the memory and executes it. All the programs that enter the system are kept in the job pool. Job pool consists of all processes residing on disk and waiting to allocate primary memory. Job scheduling concept is used if there is no space for process in the primary memory.
- Memory management is also required to manage the memory for process. CPU scheduling is applied for selecting process from memory. When computer loads more than one program in to the memory, CPU executes one program and I/O system is busy with other programs. Multiprogramming operating system do not provide user interaction with the program.
- In multiprogramming operating system, programs are competing for resources. A function of the multiprogramming operating system is CPU scheduling, memory management and I/O management.

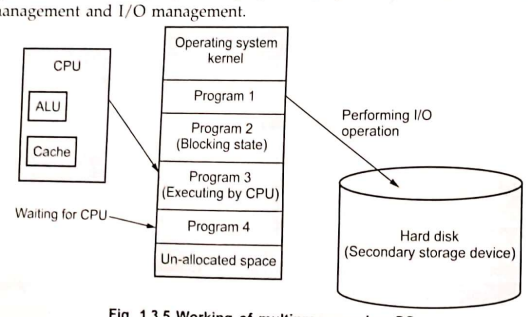


Fig. 1.3.5 Working of multiprogramming OS

- Suppose there are four programs for execution. All four programs are loaded into the memory. CPU select first program for execution. Normally programs contain instruction for CPU and I/O operation.
- CPU bound instructions : c = a + b
- I/O bound instructions : printf, scanf etc.
- When any I/O instruction is encounter in the program, CPU select next program for processing. CPU select second program for execution and I/O system select first program for performing I/O operation. Multiprogramming operating system monitors the state of all active programs and system resources.

Advantages :

1. CPU utilization is high.
2. It increases the degree of multiprogramming.

Disadvantages :

1. CPU scheduling is required.
2. Memory management is also required.

1.3.3 Time Sharing System

- Time sharing is also called multitasking operating system. It is logical extension of the multiprogramming operating systems. User interaction with program is possible in time sharing operating system. During execution of the program, user interacts directly with the program, supplying information to the program.
- Multi-tasking means that the computer can work with more than one program at a time. For example, user could be working with information from one database on the screen analyzing data, while the computer is sorting information from another database, while a excel sheet is performing calculations on a separate worksheet.
- Many users share the computer system simultaneously in time sharing operating system. Time sharing system uses multiprogramming and CPU scheduling. Each user has at least one separate program in memory.
- In time sharing system, each user is given a time slice for

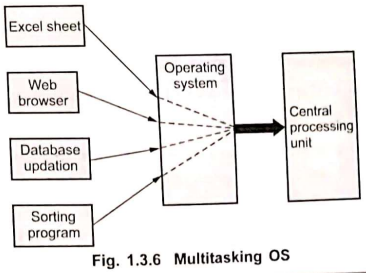


Fig. 1.3.6 Multitasking OS

executing his/her job in round robin fashion. Job continues until the time slice ends.

- Fig. 1.3.6 shows multitasking OS.
- Concept of virtual machine is used in time sharing system. It creates virtual machine one per user. User interaction with system by using virtual machine. User enters the command for virtual machine and result will received back to user.
- Time sharing system is more complex than multiprogramming operating system. It also takes help of file system. File system is stored on the disk so disk management is also required.
- Major problem with time sharing system is protection and security of data.
- Time sharing system uses medium term scheduling such as round robin for the foreground. Background process uses can use a different scheduling method.
- Difference between multiprogramming and multitasking operating system is context switching. In multiprogramming system a context switching occurs only when the currently executing process stalls for some reasons. Time sharing system gives each user the impression that the entire system is dedicated to his use. Context switching simply allows several applications to be open, but only one is working at a time.
- Truly speaking, even in true multi-tasking, only one application program is ever running at anyone instant. Because the computer automatically switches from one program to the next program so quickly, all the programs seem to run simultaneously.

1.3.4 Multiprocessor System

- Multiprocessor system means more than one processor in close communication. All the processor share common bus, clock, memory and peripheral devices.
- Multiprocessor system is also called parallel system or tightly coupled systems.

Features of Multiprocessor Systems :

1. The processor should support efficient context switching operation.
2. It supports large physical address space and larger virtual address space.
3. If one processor fails, then other processors should retrieve the interrupted process state so that execution of the process can continue.
4. The inter-process communication mechanism should be provided and implemented in hardware as it becomes efficient and easy.

- Multiprocessor systems are of two types : **symmetric multiprocessing and asymmetric multiprocessing.**

Symmetric multiprocessing

- In symmetric multiprocessing, number of homogeneous processor are running independently without affecting other programs. Each processor uses different data and program but sharing some common resources like memory, I/O device etc.
- Fig. 1.3.7 shows symmetric multiprogramming system.

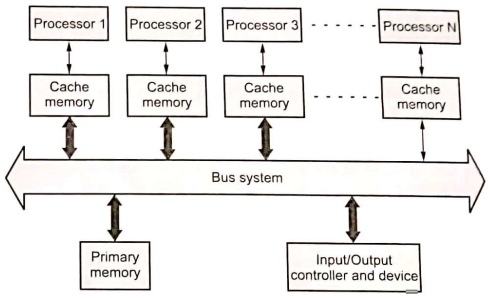


Fig. 1.3.7 Symmetric multiprogramming system

- In SMP, all processor are at the same level. They work in peers. There is no master-slave relationship in between the processor.
- Each processor shares a single copy of operating system.
- Symmetric multiprocessing is easier to implement in operating systems. Examples of symmetric multiprocessing operating systems are Windows NT 4.0, Novell Netware 2.1 etc.

Asymmetric multiprocessor

- All CPUs are not equal. There is one master processor and remaining is the slave processor. Each processor has its own memory address space.
- A master processor controls the whole operations. It gives the instruction to the slave processor or slave processor uses some predefined set of tasks.
- Each processor has own task which assign by master processor. Asymmetric multiprocessing is difficult to implement.

Advantages of multiprocessor system

1. **Throughput :** Throughput increases because number of processor is increases.

2. **Cost :** Multiprocessor system is cheaper than the multiple single processor system.
3. **Reliability :** If one processor fails, it has no effect on whole system operation.
4. **Response time :** Response time is less because of increased number of processor.

Difference between symmetric and asymmetric multiprocessing

- Symmetric multiprocessing treats all processors are equals, an I/O can be processed on any CPU. Asymmetric multiprocessing has one master CPU and the remainder CPU's are slaves. The Master distributes takes among the slaves and I/O is usually done by the master only.

1.3.5 Clustered System

- Clustered system is a group of computer system connected with a high speed communication link. Each computer system has its own memory and peripheral devices. Clustering provides high availability.
- Clustering systems are integrated with hardware cluster and software cluster. Hardware cluster support sharing of high performance disks. Software cluster is in the form of unified control of the computer system in a cluster.
- Cluster node contains layer of cluster software and it run on the node. Each node monitors the network whether the cluster is working properly or not. If the monitored computer fails, then the monitoring computer takes the control and ownership of its storage and other resources. It restarts the application of failed computer. The users and other clients will get the resources. The failed computer can remain down until it repair.
- Clustered systems are divided into two groups :
  1. Asymmetric clustering.
  2. Symmetric clustering.
- **Asymmetric clustering :** In this type of clustering, one machine is always in the hot standby mode and other machines are running its applications. The hot standby machine only monitors the active server. When active server fails, then hot standby machine takes the control and becomes the active server.
- **Symmetric clustering :** More than one host runs the applications. It uses all of the available hardware for operation. Host monitors working of other host for smooth operation of the cluster.
- Fig. 1.3.8 shows structure of cluster system. (See Fig. 1.3.8 on next page).
- Clustered system and symmetric multiprocessor provide a support for high demand applications.
- Clustered system is different than parallel system.

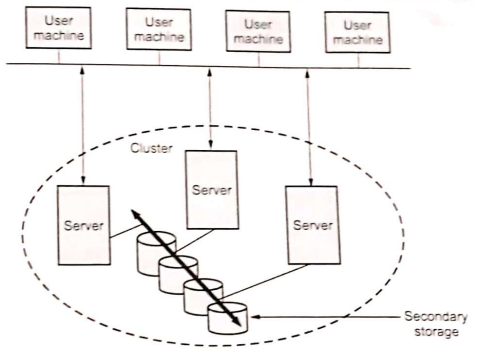


Fig. 1.3.8 Cluster system

- Other type of clustered system is parallel clusters. It uses shared storage of data. Parallel cluster requires some special purpose software and special type of applications. Example of parallel cluster is Oracle Parallel Server.
- Cluster technology changes rapidly. New developments are added to this cluster technology. Now a day, global cluster concept is used for large organizations.

1.3.6 Distributed Systems

- Distributed operating systems depend on networking for their operation. Distributed OS runs on and controls the resources of multiple machines. It provides resource sharing across the boundaries of a single computer system. It looks to users like a single machine OS. Distributing OS owns the whole network and makes it look like a virtual uniprocessor or may be a virtual multiprocessor.
- **Definition :** A distributed operating system is one that looks to its users like an ordinary operating system but runs on multiple, independent CPU.

Advantages of distributed OS :

1. **Resource sharing :** Sharing of software resources such as software libraries, database and hardware resources such as hard disks, printers and CDROM can also be done in a very effective way among all the computers and the users.
2. **Higher reliability :** Reliability refers to the degree of tolerance against errors and component failures. Availability is one of the important aspect of reliability. Availability refers to the fraction of time for which a system is available for use. Availability of a hard disk can be increased by having multiple hard disks located

at different sites. If one hard disk fails or is unavailable, the program can use some other hard disk.

3. **Better price performance ratio :** Reduction in the price of microprocessor and increasing computing power gives good price-performance ratio.

4. Shorter **responses times** and higher **throughput**.

5. **Incremental growth** : To extend power and functionality of a system by simply adding additional resources to the system.

- Fig. 1.3.9 shows the distributed system.

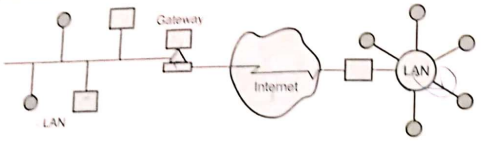


Fig. 1.3.9 Distributed system

- Examples of distributed operating system are amoeba, chrous, mach and v-system.

Difficulties in distributed OS are

1. There are no current commercially successful examples.
2. Protocol overhead can dominate computation costs.
3. Hard to build well.
4. Probably impossible to build at the scale of the internet.

1.3.7 Network Operating System

- Network operating system is designed by using existing operating system.
- In network operating system, user can login to the remote resources and then access that resource. There are multiple machines are connected by communication link. User can transfer data from their own machine to remote machine and vice versa.
- Telnet and remote file transfer are the example of network operating system.
- NOS supports multiple user accounts at the same time and enables concurrent access to shared resources by multiple clients.
- A function of network operating system is distributed over the network computers. It adds new functions whenever required. Server machine support multitasking. Operating system must be capable of executing multiple tasks at the same time.

- User knows the location of the resources before using that resource. Each machine has its own network operating system.

1.3.8 Real Time Operating Systems

- Time constraints is the key parameter is real time systems. It controls autonomous system such as robots, satellites, air traffic control and hydroelectric dams.
- When user gives an input to the system, it must process within the time limit and result is sent back. Real time system fails if it does not give result within the time limits.
- Real time systems are of two types : **Hard real time and soft real time**.
- Critical task is completed within the time limit in hard real time system. All the delay in the system is fixed and time bounded. Existing general purpose operating system does not support the hard real time system functions. Real time task cannot keep waiting for longer time without allocating kernel.
- Soft real time system is less restrictive type. Soft real time cannot guarantee that it will be able to meet deadline under all condition. Example of soft real time system is digital telephone and digital audio.
- Real time operating system uses priority scheduling algorithm to meet the response requirement of a real time application. General real time applications with their example are listed below :
  1. Transportation : Air traffic control and traffic light system
  2. Communication : Digital telephone
  3. Process control : Petroleum and paper mill
  4. Detection : Burglar system and radar system
  5. Flight simulation : Auto pilot shuttle mission simulator.

Memory management in real time system is less demanding than in other type of multiprogramming operating systems. Time critical device management is one of the main characteristics of the real time systems.

- A Real-Time Operating System (RTOS) is an operating system intended to serve real-time application process data as it comes in, typically without buffering delays.
- In general, an operating system is responsible for managing the hardware resources of a computer and hosting applications that run on the computer. An RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability. This can be especially

important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard.

- Fig. 1.3.10 shows hardware and software hierarchy in RTOS.
- An RTOS can guarantee that a program will run with very consistent timing. Real-time operating systems do this by providing programmers with a high degree of control over how tasks are prioritized, and typically also allow checking to make sure that important deadlines are met.

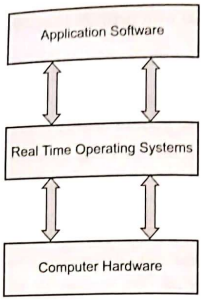


Fig. 1.3.10 RTOS

- A real-time kernel is software that manages the time and resources of a microprocessor, microcontroller or Digital Signal Processor (DSP), and provides indispensable services to your applications.

RTOS kernel services

- Along with objects, most kernels provide services that help developers create applications for real-time embedded systems.
  1. Timers
  2. Device I/O supervisor
  3. Dynamic memory allocation
  4. Intertask communication and synchronization
  5. Task management
- RTOS generally contains a real-time kernel and other higher-level services such as file management, protocol stacks, a Graphical User Interface and other components. Most additional services revolve around I/O devices.
- The core of an RTOS is known as the kernel. An API is provided to allow access to the kernel for the creation of threads, among other things. A thread is like a function that has its own stack, and a Thread Control Block (TCB). In addition to the stack, which is private to a thread, each thread control block holds information about the state of that thread.

1.3.3.1 Difference between Hard and Soft Real Time System

Hard real time	Soft real time
Hard response time is required.	Soft response time is required.
Data integrity is short term.	Data integrity is long term.
Size of data file is small or medium.	Size of data file is large.
Peak load performance is predictable.	Peak load performance is degraded.
Hard real-time systems have little laxity and generally provide full deadline compliance.	Soft real-time systems are more flexible. They have greater laxity and can tolerate certain amounts of deadline misses.
Safety critical systems are typically hard real time systems.	Linux and many OS provide soft real time system.

1.3.9 Interactive System

- OS for Interactive processing allowed programs to carry on dialogue with user via remote machines. It support real-time processing and users demand timely response.
- User issue an instruction to operating systems. System gives response back to the user immediately. Keyboard is used as input device and monitor as output device in interactive system.
- After finishing of the one command then system takes next command from keyboard instead of card reader.
- The whole sequence is that: user gives the command and wait for response. Based on the result of command, user decides about next command.
- Example of interactive system is **ATM machine.**

University Questions

1. *Give the features of Real Time Operating System and Time Sharing Operating System.* **GTU : Winter-13, Marks 3**
2. *List the types of operating systems and explain any one in detail.* **GTU : Summer-13, Marks 6**

1.4 Operating System Services

- Operating system provides different types of services to different users. It provides services to program like load of data into memory, allocating disk for storage, files or directory for open or read etc.

- Services will change according to the operating system. May be the two different operating system provides same type of services with different names. OS makes programmer job easy by providing different services.
- Following are the list of services provided by operating systems :
  1. Program execution
  2. Input-output operation
  3. Error detection
  4. File and directory operation
  5. Communication
  6. Graphical User Interface

1. **Program execution** : Before executing the program, it is loaded into the memory by operating system. Once program loads into memory, its start execution. Program finishes its execution with error or without error. It is up to the user for next operation.
2. **Input - output operation** : Program is combination of input and output statement. While executing the program, it requires I/O device. OS provides the I/O devices to the program.
3. **Error detection** : Error is related to the memory, CPU, I/O device and in the user program. Memory is full, stack overflow, file not found, directory not exist, printer is not ready, attempt to access illegal memory are the example of error detection.
4. **File and directory operation** : User wants to read and writes the file and directory. User wants to search the file/directory, rename file, and modify the file etc. user also create the file or directory. All these operation is performed by user by using help of operating system.
5. **Communication** : Communication may be inter-process communication and any other type of communication. Data transfer is one type of communication. Communication is in between two process of same machine or two process of different machine. Pipe, shared memory, socket and message passing are the different methods of communication.
6. **Graphical user interface** : User interacts with operating system by using user interface. All operating system provides user interface. Command line interface and batch interface are two types of user interface used in the operating system. In command line interface, user enters the text command for performing operation. Batch interface uses files. A file contains the command for various operations. When file executes, command output is displayed on the screen.

- All above services provided by operating system is only for single user. If suppose there are multiple user in the system, then operating system provides some additional services. These services are resource allocation, accounting, protection of data and security.
- Different types of resources are managed by the operating system. Multiple users require different resource for their execution. One type of resource may be required by two different users. So resource allocation is necessary.
- Accounting means keeping information of resources and users. Which types of resources are allocated to which user and whether particular user has permission for using this type of resources.

1.5 System Calls

- System calls provide the interface between a running program and the operating system. Any single CPU computer can execute only one instruction at a time. If a process is running a user program in user mode and needs a system service, such as reading a data from a file, it has to execute a trap instruction to transfer control to the operating system.
- Operating system provides services and system call provides interface to these services. System call is written in language C and C++ as routines. System calls are performed in a series of steps.
- System call is a technique by which a program executing in user mode can request the kernel's service.
- An **application programmer interface** is a **function definition** that specifies how to obtain a given service.
- A system call is an explicit request to the kernel mode via a software interrupt. When user mode process invokes a system call, the CPU switches to kernel mode and starts the execution of the kernel function. Making a system call is like making a special kind of procedure call. Only system call enters the kernel and procedure call does not enter into the kernel.
- Kernel implements many different types of system calls. The user mode process must pass a parameter called the system call number to identify the required system call. All system calls return an integer value. In the kernel, positive or 0 values denote a successful termination of the system call and negative values denote an error condition.
- An API does not necessarily correspond to a specific system calls.
  1. API could offer its services directly in user mode.
  2. Single API function could make several system calls.

3. Several API functions could make the same system call.

- API supports a set of functions for application programmer. It includes passing parameter to each function and return values. API used by application programmer are as follows :
  1. Windows system : win32 API
  2. POSIX system : POSIX API
  3. Virtual machine : Java API
- User function uses trap instruction for using kernel services. Application programmer uses ordinary procedure call. Operating system provides a library of user functions with name corresponding to each actual system call. Each of these stub function contains a trap to the operating system function.

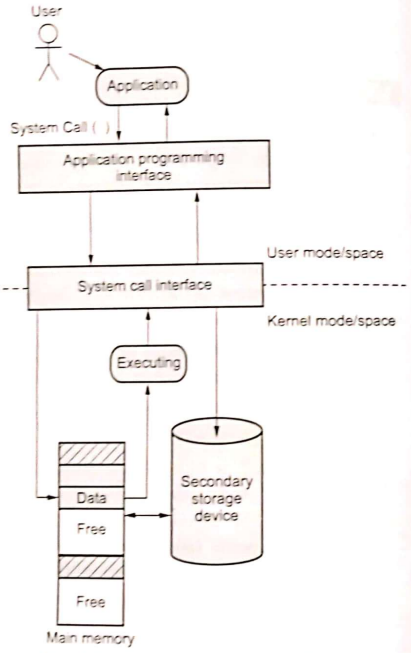


Fig. 1.5.1 Working of system call

- Trap is also called system call interface. Fig. 1.5.1 shows the working of system call.
- When application program calls the stub, trap instruction is executed and CPU switches to supervisor mode. Each system call contains its identification number. OS maintains the table of system call number. Operating system executes the system call using that number.
- When the function completes, it switches the processor to user mode and then returns control to the user process.

1.5.1 Classification of System Calls

- System call is divided into following types :
  1. File management
  2. Process management
  3. Interprocess communication
  4. I/O device management
  5. Information processing and maintenance

1. File management

- File management system calls are *create file, delete file, open file, close file, read file, write file, get and set file attribute.*
- User can create a file using create( ) system call. File name with attributes are required for creating and deleting a file through system call. After creating a file, user can performs various operations on the file.
- Read, write, reposition are the operation performed on the file. File is closed after finished using. Same type of operation is performed on directory.
- Every file has file attributes. File attributes includes name of file, type of file, accounting information etc. To perform any operation on the file, set file attribute and get file attribute executed to check the attributes.

2. Process management

- System calls for process management are create, terminate, load, execute, abort, set and get process attributes. Other system call for process management is wait for time, wait event, allocate and free memory.
- In some situation user want to terminate the currently running process abnormally then system call used. Other reasons for abnormal process termination are error message generated, memory dump, error trap.
- Operating system provides debugging facility to determine the problem of dump. Dump is written to secondary storage disk.
- Debugger is a one type of system program. It provides facility for finding and correcting bug.

3. Interprocess communication

- Pipe, socket, message passing and shared memory are used for interprocess communication. Send message, receive message, create and delete connection, attach remote device are the system calls used in interprocess communication.
- Shared memory : A process uses memory for communication. One process will create a memory portion which other processes can access. A shared segment can

be attached multiple times by the same process. Shared memory is the fastest form of IPC because data does not need to be copied between processes.

- A *socket* is a bidirectional communication device that can be used to communicate with another process on the same machine or with a process running on other machine.
- Message passing : Two processes communicate with each other by passing messages. Message passing is direct and indirect communication. Indirect communication uses mailbox for sending receiving message from other process.

4. I/O device management

- System calls for device management are request ( ) device, release ( ) device, get and set device attributes, read, write etc.
- Process needs several resources in its life time. When process request for resources, request is granted if it is free otherwise request is rejected. Once request is granted, control is transfer to the process.

5. Information processing and maintenance

- System calls for this category is set time and date, get time and date, get system data, get system data, get and set process, file and device.
- Most of the operating system provides system call for set and get the time and date.
- This type of system call is used for transferring information from user to operating system and vice versa.

1.6 Operating System Structure

- Modern operating system is complex in nature. Here we study different structure of the operating systems.

1.6.1 Simple Structure

- Microsoft-Disk Operating System (MS-DOS) is example of simple structure operating system. Most of the commercial systems do not have well defined structure. Initially DOS is small and simple in size. When new versions are introduced, size goes on increasing.
- There is no CPU Execution Mode (user and kernel), and so errors in applications can cause the whole system to crash.
- MS-DOS is layered structure operating system. It consists of following layers :
  1. Application program layer
  2. System program layer for resident program

  3. Device driver layer
  4. ROM BIOS device driver layer
- In DOS, application program directly interact with BIOS device driver. If user makes any changes in the BIOS device driver, it creates the problem and affects the whole system. Memory size is also limited so after use memory must be made free for other users.
- Another example of this type is UNIX operating system. Initially it provides limited hardware functionality. UNIX is divided into two parts : **Kernel and system programs**.
- Kernel provides system calls for CPU scheduling, I/O management, file management and memory management. System call uses application program interface in UNIX. API defines user interface by using set of system programs. Kernel supports an API and user interface.
- To support more advanced version of hardware, new UNIX operating system was developed. It is consists of different modules for controlling the hardware. Module will communicate with each other for proper operation. Designers are free to design operating systems which support all new versions of hardware and user requirements.

1.6.2 Layered Approach

- Second type of operating system design is layered approach. Operating system is divided into number of layers. Each layer boundary is properly defined. Bottom layer is called layer 0 and top most layer is called layer N. Layer N provides user interface.
- A function of layer is also fixed. Fig. 1.6.1 shows the operating system layer. Each layer consists of data structure and a set of routines. Layer provides services to upper and lower layers.
- Modularity is the advantage of the layered system. Number of

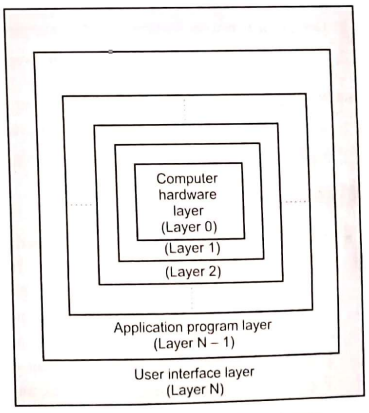


Fig. 1.6.1 Operating system layers

layers are selected properly and each layer uses the services and functions of lower level layer (Below layer of the current layer). This type of design simplifies the debugging and system verification.

- First layer contains only basic hardware to implement function. So only first layer is debugged for checking the whole system. If error is not found then the system will works properly. If error is encounter while debugging second layer, then error is related to the second layer only. In this operating system is designed and implemented. It simplifies the design task of the operating system.
- Care should be taken while designing the layers. Which functions are added to which layer must be designing properly? Layer boundaries must be defined properly. Some of the function includes only in the lower layer. Device driver, memory management and input/output operation function must be included in the lower layer.
- Secondary storage is required for all operation. When CPU changes the process as per scheduling method, currently executing process is stored on the secondary storage. Disk space is required for this purpose. So CPU scheduling is includes in the above layer of the secondary storage layer.
- Each layer uses system calls for performing their operation. Layer adds overhead to the system call.

Advantages of Layered Approach :

1. Modularization makes debugging much easier.
2. Design and implementation makes simple.
3. It provides transparency between the layers.

Disadvantages of Layered Approach :

1. Less efficient is system call takes longer times.
2. Interaction between the layers and parameters passing is difficult.

1.7 Kernel

- The Kernel is a software code that resides in the central core of a operating system. It has complete control over the system. A Kernel is different than the shell. The shell is the outermost part of an operating system and a program that interacts with user commands. Fig. 1.7.1 shows the position of the Kernel.

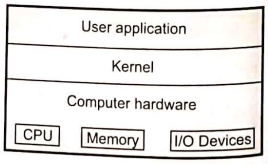


Fig. 1.7.1 Kernel

- The kernel does not interact directly with the user. But it interacts with the shell, other programs and hardware.
- When operating system boots, kernel is the first part of the operating system to load into memory. Kernel remains in memory for the entire duration of the computer session. The kernel code is usually loaded into a protected area of memory.
- The kernel performs its tasks in kernel space. Executing processes and handling interrupts are performed by kernel in kernel space. User performs its task in user area of memory. Memory is divided into system area and user area. This memory separation is made in order to prevent user data and kernel data from interfering with each other.
- When a computer crashes, it actually means the kernel has crashed. Single program crash is not a kernel crash. The kernel provides services for process management, memory management, file management and I/O management. System calls are used for providing this type of services.
- Kernel content will change according to the operating system but typically it includes :
  1. Scheduler
  2. Supervisor
  3. Interrupt handler
  4. Memory manager
- Scheduler allocates the kernel's processing time to various processes. Supervisor grants the permission to use computer system resources to each process. Interrupt handler handles all requests from the various hardware devices which compete for the kernel's services. Memory manager allocates space in memory for all users of the kernel services.

Types of kernel :

1. Monolithic kernel
2. Microkernel
3. Hybrid kernel
4. Exo-kernel

1.7.1 Monolithic Kernel

- Traditional UNIX operating system uses monolithic kernel architecture. The entire operating system runs as a single program in kernel mode. Program contains operating system core function and device drivers.
- Most of the operation performed by kernel is via system call. Required system calls are made within programs and a checked copy of the request is passed through a system call. Fig. 1.7.2 shows monolithic kernel.
- LINUX operating system and FreeBSD uses modern monolithic kernel architecture. It loads the modules at run time. There is easy access of kernel function as required and minimize the code running in kernel space.

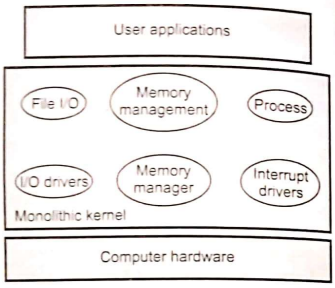


Fig. 1.7.2 Monolithic kernel

- Monolithic kernel used in Windows 95, Windows 98, Linux and FreeBSD etc.

dvantages :

1. Simple to design and implement.
2. Simplicity provides speed on simple hardware.
3. It can be expanded using module system.
4. Time tested and design well known.

sadvantages :

1. Runtime loading and unloading is not possible because of module system.
2. If code base size increases, maintain is difficult.
3. Fault tolerance is low.

7.2 Microkernel

- Microkernel provides minimal services like defining memory address space, IPC and process management. It is small operating core. Hardware resource management is implemented whenever process is executing. The function of microkernel is to provide a communication facility between the client programs. It

also provides facility to various services which are running in user space. Message passing method is for communication two processes.

- Microkernel runs in kernel mode and rest run in normal user processes. Microkernel also provides more security and reliability. Most of the services are running as user rather than kernel processes. By running device driver and file system as a separate user process, a error in one can crash only single component. Fig. 1.7.3 shows microkernel.

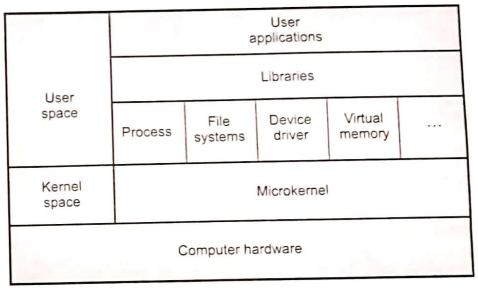


Fig. 1.7.3 Microkernel

- Mach operating system uses microkernel architecture. Microkernel in Windows NT operating system provides portability and modularity. The kernel is surrounded by a number of compact subsystems so that the task of implementing Windows NT on a variety of platform is easy.
- Microkernel architecture assigns only a few essential functions to the kernel, including address space, IPC and basic scheduling. QNX is a real time operating system that is also based upon the microkernel design.
- Main disadvantage is poor performance due to increased system overhead from message passing.

Advantages of microkernel :

1. Microkernel allows the addition of new services.
2. Microkernel architecture design provides a uniform interface on requests made by a process.
3. Microkernel architecture supports object oriented operating system.
4. Modular design helps to enhance reliability.
5. Microkernel lends itself to distributed system support.

6. Microkernel architecture support flexibility. User can add or subtract services according to the requirement.

1.7.3 Comparison between Monolithic Kernel and Microkernel

Sr. No.	Monolithic Kernel	Microkernel
1.	Kernel size is large.	Kernel size is small.
2.	OS is complex to design.	OS is easy to design, implement and install.
3.	Request may be serviced faster.	Request may be serviced slower than monolithic Kernel.
4.	All the operating system services are included in the Kernel.	Kernel provides only IPC and low level device management services.
5.	No message passing and no context switching are required while the Kernel is performing the job.	Microkernel requires message passing and context switches.

1.7.4 Reentrant Kernel

- Several processes may be in kernel mode at the same time. A reentrant kernel is able to suspend the current running process even if process is in the kernel mode.
- An interrupts from I/O devices may not be handled immediately by the operating system.
- Normally operating system is composed of reentrant and non-reentrant functions.
- UNIX kernels are reentrant kernel. This means that several processes may be executing in kernel mode at the same time.
- A Kernel that implements "reentrant routines" are referred to as "Reentrant kernels". When two tasks can execute the function at the same time without intertering with each other, then the function is reentrant.
- In Non reentrant kernel mode, every single process acts on its own set of memory locations and thus cannot interfere with the others.
- On uni-processor systems, only one process can progress, but many may be blocked in kernel mode waiting for some events.
- Reentrant functions only modify local variables but do not alter global data structures. To provide reentrant kernel, the kernel is implemented as a collection of reentrant functions.

- With a reentrant kernel, the scheduler will assign CPU to another process until an interrupt from disk controller indicates that the data is available and the earlier thread can be resumed. This process can still access I/O, like user input.
- If a hardware interrupt occurs, a reentrant kernel is able to suspend the current running process even if that process is in Kernel Mode. This capability is very important, because it improves the throughput of the device controllers that issue interrupts. Once a device has issued an interrupt, it waits until the CPU acknowledges it. If the kernel is able to answer quickly, the device controller will be able to perform other tasks while the CPU handles the interrupt.

1.8 Design and Implementation of OS

Design :

- Before designing the system, first define goals and specification of the system. Goals and specification will give the idea about the system and it also understands the requirement of the user. At the higher level, system design is dominated by the choice of hardware and system type.
- System type may be batch system, multiprogramming, multitasking, real time and distributed system. Hardware must support all these type of operating system.
- Once the goal and specification is clearly understood, then a requirement is related to user and system. These requirements are called user goals and system goals.
- User goals are as follows :
  1. Simple to use and understand.
  2. Provides reliability.
  3. Provides security.
  4. Fast speed.
- System goals are as follows :
  1. Easy to design.
  2. Implementation and maintenance is also simple and easy.
  3. Free from error.
  4. Efficient to use.
- Operating system requirements is not fixed. It will change according to the user requirements. Single user operating system is required and multiple user operating systems is also required.

Implementation

- At first, operating systems were written in assembly, but now a days C/C++ is the language commonly used.

- Small blocks of assembly code are still needed, especially related to some low level I/O functions in device drivers, turning interrupt on and off and the Test and Set instruction for synchronization facilities.
- Using higher level language allow code to be written faster, is easier to read, and can be debugged easier. It also makes the OS much easier to port to different hardware platforms.

1.9 Operations of Operating System

- For single-user programmer operating systems, programmer has the complete control over the system. They operate the system from the console. When new operating systems developed with some additional features, the system control transfers from programmer to the operating system.
- Old operating systems were called resident monitors. The operating system began to perform many of the functions, like input-output operation with the resident monitor.
- Before the operating system, programmer is responsible for the controls of input-output device operations. As the requirements of programmers from computer systems go on increasing and development in the field of communication helps to the operating system to fulfill the needs of programmers.
- Sharing of resource among different programmers is possible without increasing cost. It improves the system utilization but problems increase. If single system was used without share, an error occurs, that could cause problems for only the one program which was running on that machine.
- In sharing, other programs also affected by single program. For example, batch operating system faces the problem of infinite loop. This loop could prevent the correct operation of many jobs. In multiprogramming system, one erroreous program affects the other program or data of that program.
- For proper operation and error free result, protection of error is required. Without protection, only single process will execute one at a time otherwise the output of each program is separated. While designing the operating system, this type of care must be taken into consideration.
- Computer hardware detects the errors. Operating system handled this type of errors. Execution of illegal instruction or access of memory that is not in the user's address space, this type of operation found by the hardware and will trap to the operating system.

- Trap uses an interrupt vector and it transfer the control to the operating system. Whenever error occurs, OS terminates the program abnormally. Hardware protection is used to handle this type of situation.

1.9.1 Dual Mode Operation

- For proper operation and correct output, operating system must be protected. The users program and data must be protected from any malfunctioning program. Shared resource also needs some kind of protection.
- Dual mode uses user mode and monitor mode for working of OS. The monitor mode also called system mode, supervisor mode or privileged mode.
- For indicating current mode of the system, mode bit is used in the computer hardware. The mode bit is 0 for monitor and 1 for user. With the mode bit, user are able to distinguish between a task that is executed in user mode or monitor mode.
- At the booting time, the hardware starts in the monitor mode, then operating system is loaded. The hardware switches from user mode to monitor mode when interrupts occur. In monitor mode, OS gains control of the system.
- The dual mode operation provides the protection to the operating system from unauthorised users. The privileged instructions are executed only in the monitor mode. The computer hardware is not allowed for executing the privillege instructions in other mode, i.e. user mode. If anybody tries to execute the instructions in user mode, it is considered as illegal instruction and also traps it to the operating system.
- Software may trigger an interrupt by executing a special operation called a system call. System call is one type of request which is invoked by the user or system. Using privileged instructions, user will interact with the operating system. This type of request is invoked by user to execute the privileged instructions. As said earlier, this request is called system call or monitor call. When a system call is executed, it is treated by the hardware as a software interrupt.

1.10 System Programs

- Modern operating system consists of a collection of system programs. System program that provides an application programming environment on top of the hardware. Some of them are simply user interfaces to system calls. They can be divided into these categories :
  1. File management

2. Status information
3. File modification
4. Programming language support
5. Program loading and execution
6. Communications.

- File management programs are used to create, delete, copy, rename, list, dump on files and directory. Status information system programs covers the date, time, disk space, available memory and users. All this information is formatted and displayed on output device or printed.
- Text editors are used for file modification. In this, new file is created and content of file is modified. Programming language support includes the compilers, assemblers, interpreters for common programming language like C, C++, Java, Visual Basic.
- For program loading and execution, it is loaded into memory then program is executed by processor. Operating system provides different types loaders and linkers to complete execution operation.
- Debugging facility is provided by the operating system with the help of application program. It is used for checking errors.
- Communication between two devices are performed by creating temporary connection. Communication is in between process, users and other I/O devices. File transfer, remote login, electronic mail, browsing web, downloading multimedia data are the example of communication.

1.11 Concept of Virtual Machine

- In a pure virtual machine architecture the operating system gives each process the illusion that it is the only process on the machine. The user writes an application as if only its code were running on the system.
- Each user interacts with the computer by typing commands to the virtual machine on a virtual system console and receiving results back from the machine as soon as they are computed.
- Each user directs the virtual machine to perform different commands. These commands are then executed on the physical machine in a multiprogramming environments.
- Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility.

- It allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine. Fig. 1.11.1 shows virtual machine.

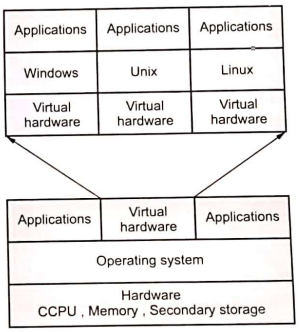


Fig. 1.11.1 Virtual machine

- Each virtual machine has its own set of virtual hardware (e.g., RAM, CPU, NIC, etc.) upon which an operating system and applications are loaded. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- The main components of virtual machine are the control program, conversational monitor system, remote spooling communication and interactive problem control system. The control program creates the environments in which virtual machines can executes. It also manages the real machines underlying the virtual machine environment.
- The java virtual machine is one of the most widely used virtual machines.
- Virtual machines tend to be less efficient than real machines because they access the hardware indirectly.

Benefits :

1. There is no overlap amongst memory as each Virtual Memory has its own memory space.
2. Virtual machines are completely isolated from the host machine and other virtual machines.
3. Data does not leak across virtual machines.

Virtualization

- Virtualization means running multiple machines on a single hardware. The "Real" hardware invisible to operating system. OS only sees an abstracted out picture. Only Virtual Machine Monitor (VMM) talks to hardware.
- It is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. This includes making a single physical resource appear to function as multiple logical resources; or it can include making multiple physical resources appear as a single logical resource."
- It is divided into two main categories :
  1. Platform virtualization involves the simulation of virtual machines.
  2. Resource virtualization involves the simulation of combined, fragmented, or simplified resources.

Hypervisor

- In computing, a hypervisor is a virtualization platform that allows multiple operating systems to run on a host computer at the same time. The term usually refers to an implementation using full virtualization.
- Hypervisors are currently classified in two types :
  1. Type 1 hypervisor is software that runs directly on a given hardware platform A "guest" operating system thus runs at the second level above the hardware.
  2. Type 2 hypervisor is software that runs within an operating system environment. A "guest" operating system thus runs at the third level above the hardware.
- Bochs and QEMU are PC emulators that allow operating systems such as Windows or Linux to be run in the user-space of a Linux operating system.
- VMware is a popular commercial full-virtualization solution that can virtualizes unmodified operating systems.
- Xen is an open source para-virtualization solution that requires modifications to the guest operating systems but achieves near native performance by collaborating with the hypervisor.
- Microsoft Virtual PC is a para-virtualization virtual machine approach.
- User-mode Linux (UML) is another para-virtualization solution that is open source. Each guest operating system executes as a process of the host operating system.
- Cooperative Linux, is a virtualization solution that allows two operating systems to cooperatively share the underlying hardware.

- Linux-Vserver is an operating system-level virtualization solution for GNU/Linux systems with secure isolation of independent guest servers.
- The Linux KVM is virtualization technology that has been integrated into the mainline Linux kernel . Runs as a single kernel loadable module, a Linux kernel running on virtualization-capable hardware is able to act as a hypervisor and support unmodified Linux and Windows guest operating systems.
- Fig. 1.11.2 shows para-virtualization architecture. In Para-virtualization, the virtual machine does not necessarily simulate hardware, but instead offers a special API that can only be used by modifying the "guest" OS. This system call to the hypervisor is called a "hypercall" in Xen.

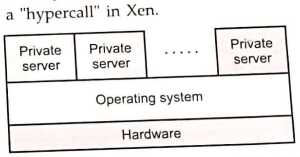


Fig. 1.11.2 Para-virtualization architecture

- Para-virtualization shares the process with the guest operating system

Problems with para-virtualization

1. Para-virtualized systems won't run on native hardware
2. There are many different para-virtualization systems that use different commands, etc.

Platform virtualization

- The creation of a virtual machine using a combination of hardware and software is referred to as platform virtualization
- Platform virtualization is performed on a given hardware platform by "host" software , which creates a simulated computer environment for its "guest" software.
- The "guest" software, which is often itself a complete operating system, runs just as if it were installed on a stand-alone hardware platform.
- Typically, many such virtual machines are simulated on a given physical machine.
- For the "guest" system to function, the simulation must be robust enough to support all the guest system's external interfaces, which may include hardware drivers.

Resource virtualization

- The basic concept of platform virtualization, was later extended to the virtualization of specific system resources, such as storage volumes, name spaces, and network resources.
- Resource aggregation, spanning, or concatenation combines individual components into larger resources or resource pools. For example : RAID and volume managers combine many disks into one large logical disk.
- Virtual Private Network (VPN), Network Address Translation (NAT), and similar networking technologies create a virtualized network namespace within or across network subnets. Multiprocessor and multi-core computer systems often present what appears as a single, fast processor.

❑❑❑

2 Process Management

Syllabus

Processes : *Definition , Process relationship , Process states , Process state transitions , Process control block,Context switching - Threads - Concept of multithreads, Benefits of threads - Types of threads*

Process scheduling : *Definition, Scheduling objectives, Types of schedulers, Scheduling criteria CPU utilization, Throughput, Turnaround time, Waiting time, Response time (definition only), Scheduling algorithms : Preemptive and Non preemptive, FCFS - SJF - RR , Multiprocessor scheduling : Types, Performance evaluation of the scheduling*

Contents

2.1	Process	Winter-13, Summer-13	Marks 7
2.2	Process States	Winter-13, Summer-13,14	Marks 7
2.3	Process Description		
2.4	Process Control		
2.5	Execution of the Operating System		
2.6	Threads	Summer-13,14	Marks 7
2.7	Types of Threads		
2.8	Concept of Multithreads		
2.9	Types of Scheduling		
2.10	Process Scheduling		
2.11	Scheduling Algorithms	Winter-13, Summer-13,14	Marks 7
2.12	Multiprocessor Scheduling		
2.13	Performance Evaluation of the Scheduling	Summer-14	Marks 7

2.1 Process

GTU : Winter-13, Summer-11

- Process term is used in MULTICS system in the 1960. Process is an asynchronous activity.
- Process is an active entity that requires a set of resources, including a processor, program counter, registers to perform its function. Multiple processes may be associated with one program.
- Process means a program in execution. Process execution must progress in sequential order. It also called task. Task is a single instance of an executable program.
- Each process has its own address space. Address space is divided into two regions :
  1. Text region 2. Data region 3. Stack region
- **Text region :** It stores the code that that the processor executes.
- **Data region :** It stores variables and dynamically allocated memory that the process uses during execution.
- **Stack region :** It stores instructions and local variables for active procedure calls.
- Process is a dynamic entity that executes a program on a particular set of data using resources allocated by the operating system. A process can run to completion only when all requested hardware and software resources have been allocated to the process.
- Program is a passive entity. Program becomes process when executable file loaded into memory. A process may be independent of other processes in the system.
- After booting operating system, it creates foreground processes and background processes. Foreground processes interact with user and background processes is used by system. Background processes are related e-mail, web pages, news and printing.
- A process is used as a fundamental unit for resource allocation in operating system.

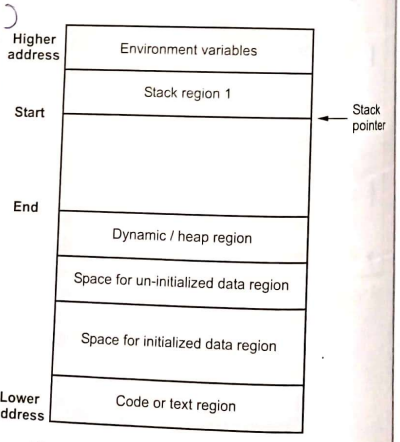


Fig. 2.1.1 Memory layout for a process

Fig. 2.1.1 shows memory layout for a process. A process normally has its own private memory area in which it runs. (See Fig. 2.1.1 on previous page)

- Text region contains executable instructions. It is be placed below the heap or stack.
- An initialized data region contains the static and global variables that are initialized by the user.
- Un-initialized data region contains all static and global variables that are initialized by kernel to zero.
- Heap is used for dynamic memory allocation and stack contains program counter.

2.1.1 Difference between Process and Program

Sr. No.	Process	Program
1.	Process is active entity.	Program is passive entity.
2.	Process is a sequence of instruction executions.	Program contains the instructions.
3.	Process exists in a limited span of time.	A program exists at single place and continues to exist.
4.	Process is a dynamic entity.	Program is a static entity.

2.1.2 Process Control Block

- Operating system keeps an internal data structure to describe each process it manages. When OS creates process, it creates this process descriptor. In some operating system, it calls Process Control Block (PCB). Fig. 2.1.2 shows process control block.
- Process control block will change according to the operating system. PCB is also called **task control block.**
- The PCB is identified by an integer Process ID (PID). When a process is running, its hardware state is inside the CPU. When the OS stops running a process, it saves the register's values in the PCB.

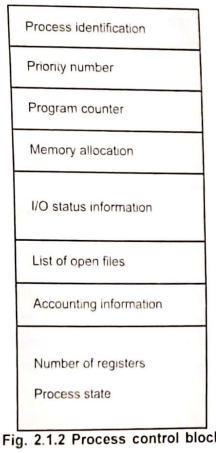


Fig. 2.1.2 Process control block

- When a process is created by operating system, it allocates a PCB for it. OS initializes PCB and puts PCB on the correct queue. Following information is stored in process control block.

1. **Process identification :** Each process is uniquely identified by the user's identification and a pointer connecting it to its descriptor.
2. **Priority number :** Operating system allocates the priority number to each process. According to the priority number it allocates the resources.
3. **Program counter :** The PC indicates the address of the next instruction to be executed for this current process.
4. **Memory allocation :** It contains the value of the base registers, limit registers and the page tables depending on the memory system used by the operating system.
5. **I/O status information :** It maintains information about the open files, list of I/O devices allocated to the process etc.
6. **List of open files :** Process uses number of files for operation. Operating system keeps track of all opened file by this process.
7. **Process state :** Process may be in any one of the state : new, ready, running, and waiting, terminate.

- When process changes the state, the operating system must update information in the process's process control block. Process control block maintain other information which is not included in PCB block diagram. This information includes CPU scheduling, file management, input-output management information.
- When process is created, hardware registers and flags values are set by loader or linker.
- Operating system maintains pointers to each process's PCB in a per user process table or system wide process table. This information is used to access PCB quickly. Fig. 2.1.3 shows process table with PCB.

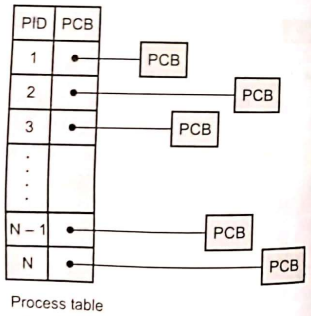


Fig. 2.1.3 Process table with PCB

University Questions

1. What is proces control block ? Explain various entries of it. **GTU : Winter-13, Marks 7**
2. Define process. List the major events for creation of a process and explain them. **GTU : Summer-13, Marks 7**
3. What is PCB ? Discuss its major fields. **GTU : Summer-13, Marks 7**

2.2 Process States

GTU : Winter-13, Summer-13,14

- Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process. Fig. 2.2.1 shows a process state diagram. A state diagram is composed of a set of states and transitions between states.

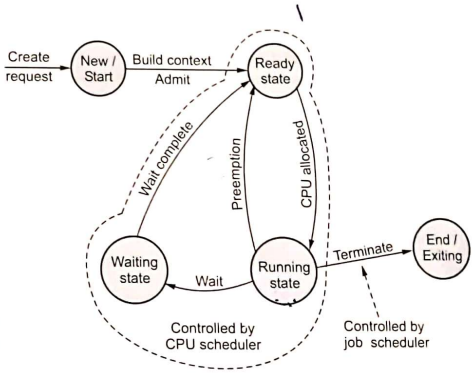


Fig. 2.2.1 Process state diagram

- State diagram is used by process manager to determine the type of service to provide to the process. The process states are as follows :

1. New 2. Ready 3. Running 4. Blocked/Waiting 5. Exit/End

- **New :** Operating system creates new process by using fork( ) system call. These process are newly created process and resources are not allocated.
- **Ready :** The process is competing for the CPU. Process reaches to the head of the list (queue).
- **Running :** The process that is currently being executed. Operating system allocates all the hardware and software resources to the process for execution.

- **Blocked/Waiting :** A process is waiting until some event occurs such as completion of an input-output operation.
- **Exit/End :** A process completes its operations and releases all resources.
- Operating system maintains a ready list of ready process and a blocked list of blocked processes. The ready list is maintained in priority order and blocked list is typically unordered.
- The act of assigning a processor to the first process on the ready list is called **dispatching** and is performed by a system entity called the **dispatcher.**
- When process is in new state, the program remains in the secondary storage. Here process itself is not in the main memory and space is not allocated.
- The process exists a system because of two reasons :
  1. Process is terminated when it completes its operation.
  2. Process aborts due to an unrecoverable error.
- To prevent any one process from continuously using the system, the operating system sets a hardware interrupting clock to allow a process to run for a specific time interval or time quantum.
- The process may request a resource when it is in the running state. In most of the operating system, if a running process requests an immediately available resources the process is allowed to continue in the running state.
- From the blocked state, the process can move to the ready state only by being allocated the requested resource.
- User only initiate process state transition is **blocked** and remaining all other state transitions is initiated by the operating system.

Sr. No.	State transition	Remarks
1.	Ready to running	Process is dispatched.
2.	Running to ready	Process time slice expires.
3.	Running to blocked	When a process blocks.
4.	Blocked to ready	When the event for which it has been waiting occurs.
5.	Ready to exit	This is possible when parent process may terminate child process at any time.
6.	Running to exit	When currently running process completes its operation then OS terminates the process.

Suspended processes

- Each process to be executed must be loaded into main memory. In uniprocessor system, processor remains idle because of execution speed mismatch of processor and I/O devices. An I/O device is slower than processor.
- Main memory accommodates multiple processes and the processor select next process when one process is blocked. So even with multiprogramming operating system, a processor could be idle most of the time. Main memory size is increased for accommodating more processes but cost will increase.
- **Suspended state :** when all of the processes in main memory are in the blocked state, the operating system can suspend one process by putting it in the suspend state and transferring it to disk. The space that is freed in main memory can then be used to bring in another process.
- Characteristics of a suspended process :
  1. The process cannot execute immediately.
  2. The process may or may not be waiting for an event.
  3. The process was placed in a suspended state by a parent process, or itself or by operating system.
  4. The process may not be removed from suspended state until its turn comes.

University Questions

1. *Explain "5 state" process state transition diagram with illustration.* **GTU : Winter-13, Marks 4**
2. *Draw process state diagram for THREE states and explain all states.* **GTU : Summer-13, Marks 7**
3. *Define a process. Explain the process state transition with a neat diagram.* **GTU : Summer-14, Marks 7**

2.3 Process Description

- Operating systems are considered as a manager of the various hardware resources. Goal of the OS is the management of hardware resources and control of processes accessing the resources. Fig. 2.3.1 shows the processes with resources.
(See Fig 2.3.1 on next page)
- OS as a service provider, naturally also imposes control on processes that consume the services. That is both processes and resources are under the operating system's control. Control table is used by OS.

2. **Memory tables :** Operating system uses memory table for keeping track of both main memory and secondary memory. Main memory is used by operating system and processes. For operating system, memory is reserved and it cannot allocate to the user process. Following information is stored in the memory table.
   a. Allocation of main memory to user processes.
   b. Allocation of secondary memory to user processes.
   c. Protection attributes.
3. **Input-Output tables :** This helps operating system for managing I/O devices. It maintains information like :
   a. Device availability
   b. Status of I/O operation
   c. Location of device in main memory
4. **File tables :** It maintains information about the existence of files and their location on secondary memory with attributes.

2.4 Process Control

- Following operations are performed on the process :
  1. Process creation 2. Process termination
- A mechanism for process creation and termination via system calls. Programmers usually access these system calls via application interface (API) / library.

2.4.1 Process Creation

- Operating system creates the process in following situations :
  1. Starting of new batch job.
  2. User request for creating new process.
  3. To provide new services by OS.
  4. System call from currently running process.
- Operating system creates a new process with the specified or default attributes and identifier. A process may create several new sub-process.
- Parent process is creating process and the new processes are called the children of the process. When operating system creates process, it builds the data structure for managing process and allocates address space in primary main memory.
- Operating system creates foreground and background process. Process is identified by unique process identifier (PID) in UNIX and windows operating system. PID

- All processes in UNIX are created using the fork( ) system call. The forking process is called the parent process. The new process is called the child process. Both the parent and child process have their own and private memory. Open files are shared between parent and child.
- If the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork( ) calls will not be affected even though they have identical variable names.

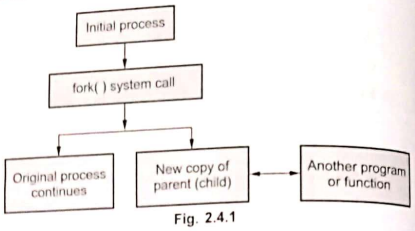


Fig. 2.4.1

- When a process is created, OS assigns some attributes. These are priority, privilege level, requirement of memory, access right, memory protection, PID etc. To perform operation, process needs software and hardware resources. It includes CPU time, files, memory, I/O device.
- Relation between parent process and child process is as follows :
  1. Parent process continues to execute concurrently with its child process.
  2. Parent process waits until some or all of its children have terminated.

```
void main( )
    {
        printf("Operating System\n");
        fork( );
          printf("Technical Publications\n");
    return 0;
}
```

- In above program Operating System is printed only once and Technical Publications is printed two times.

2.4.2 Process Termination

- When process finishes its normal execution then that process is terminate. Operating system delete that process using exit ( ) system call. After deleting process, memory space becomes free.
- OS passes the child's exit status to the parent process and then discards the process. At the same time, it de-allocate all the resources hold by this process.

- Following are the various reasons for process termination :
  1. Normal completion of operation
  2. Memory is not available
  3. Time slice expired.
  4. Parent termination
  5. Failure of I/O
  6. Request from parent process
  7. Misuse of access rights

2.5 Execution of the Operating System

- We know that, user programs run in a computer system as processes. Have you ever thought about operating system ? Whether is consider itself as a process or not ? There is no fix answer for this but it usually depends upon the design of the operating system.
- Different design options are as follows :
  1. Non-process kernel
  2. Execution within user processes
  3. Process-based operation system
- Fig. 2.5.1 shows the relationship between OS and user processes.

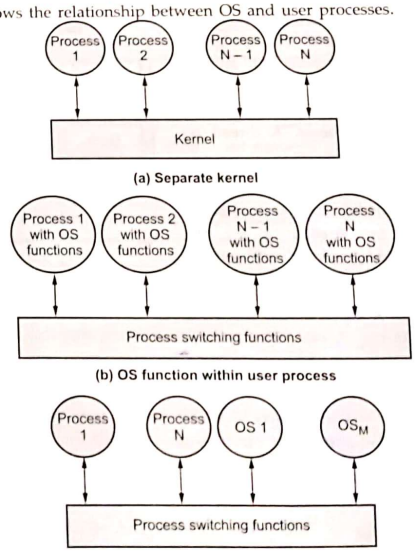


Fig. 2.5.1 OS and user process relationship

- **Non-process kernel (Separate kernel) :** Kernel and user process are executing separately. Here process means only user programs. Kernel code is executing in privileged mode.
- **Execution within user processes :** Operating system is primarily a collection of routines that the user calls to perform various functions, executed within the environment of the user's process.
- **Process-based operation system :** It is to implement the operating system as a collection of system processes. Here kernel still executes in a kernel mode, major kernel functions are organized as separate processes. Advantages :
  1. Support modular programming.
  2. Improve the performance.
  3. It consider as a separate process for noncritical OS function.

2.6 Threads

GTU : Summer-13,14

- Thread is a dispatchable unit of work. It consists of thread ID, program counter, stack and register set. Thread is also called a Light Weight Process (LWP). Because they take fewer resources then a process. A thread is easy to create and destroy.

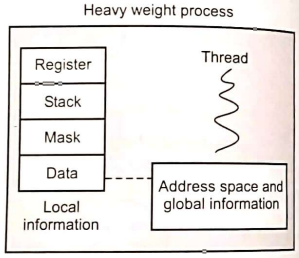


Fig. 2.6.1 Thread

- It shares many attributes of a process. Threads are scheduled on a processor. Each thread can execute a set of instructions independent of other threads and processes. Fig. 2.6.1 shows a thread.
- Every program has at least one thread. Programs without multithreading execute sequentially. That is, after executing one instruction the next instruction in sequence is executed.
- Thread is a basic processing unit to which an operating system allocates processor time and more than one thread can be executing code inside a process.
- When user double click on an icon by using mouse, the operating system creates a process and that process has one thread that runs the icon's code.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.

- Normally, an operating system will have a separate thread for each different activity. The OS will have a separate thread for each process and that thread will perform OS activities on behalf of the process. In this condition, each user process is backed by a kernel thread. When process issues a system call to read a file, the process's thread will take over, find out which disk accesses to generate, and issue the low level instructions required to start the transfer. It then suspends until the disk finishes reading in the data.
- When process starts up a remote TCP connection, its thread handles the low-level details of sending out network packets.
- In Remote Procedure Call (RPC), servers are multithreaded. Server receives the messages using a separate thread.
- Different operating system uses threads in different ways.
  1. Free memory is managed by kernel thread in LINUX OS.
  2. Solaries uses a kernel thread for interrupt handling.

2.6.1 Thread Advantages

1. Context switching time is minimized.
2. Thread support for efficient communication.
3. Resource sharing is possible using threads.
4. A thread provides concurrency within a process.
5. It is more economical to create and context switch threads.

2.6.2 Difference between Thread and Process

Sr. No.	Thread	Process
1.	Thread is also called lightweight process.	Process is also called heavyweight process.
2.	Operating system is not required for thread switching.	Operating system interface is required for process switching.
3.	One thread can read, write or even completely clean another threads stack.	Each process operates independently of the other process.
4.	All threads can share same set of open files and child processes.	In multiple processing, each process executes the same code but has its own memory and file resources.

If one thread is blocked and waiting then second thread in the same task can run.	If one server process is blocked then other server process cannot execute until the first process is unblocked.
Uses fewer resources.	Uses more resources.

Thread Lifecycle

Fig. 2.6.2 shows a thread lifecycle. A thread has one of the following states.

New : Thread is just created.

Ready : Thread's start( ) method invoked and now it is executing. OS put thread into ready queue.

Running : Highest priority ready thread enters the running state. Thread is assigned a processor and now is running.

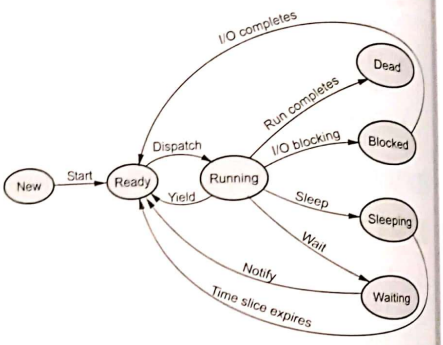


Fig. 2.6.2 Thread lifecycle

Blocked : This is the state when thread is waiting for a lock to access an object.

Waiting : Here thread is waiting indefinitely for another thread to perform an action.

Sleeping : Thread sleep for a specified time of period. When sleeping time expires, it enters to ready state. CPU is not used by sleeping thread.

Dead : When thread completes task or operation.

University Questions

Explain the classical thread model with its implementation strategies.

GTU : Summer-13, Marks 7

What is thread and what are the differences between user-level threads and kernel supported threads ? Under what circumstances is one type "better" than the other ?

GTU : Summer-14, Marks 7

2.7 Types of Threads

- Threads are of two types :
  1. User level thread 2. Kernel level thread
- All modern operating system support threading model. Implementation of thread will change according to the operating system.

2.7.1 User Level Thread

- User level thread uses user space for thread scheduling. These threads are **transparent** to the operating system. User level threads are created by runtime libraries that cannot execute privileged instructions.
- User-level threads have low overhead but it can achieve high performance in computation. User-level threads are managed entirely by the run-time system.

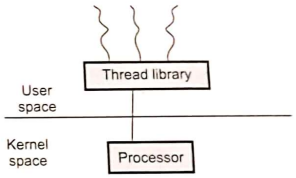


Fig. 2.7.1 User level thread

- User-level threads are small and faster. A thread is simply represented by a PC, registers, stack and small thread control block. Fig. 2.7.1 shows user level thread.
- The code for creation and destroying thread, message passing and data transfer, thread scheduling is included into thread library. Kernel is unaware of user level thread.
- User level threads do not invoke the Kernel for scheduling decision.
- User level thread are also called **many to one mapping** thread because the operating system maps all threads in a multithreaded process to a single execution context. The operating system considers as each multithreaded processes as a single execution unit.
- Example : POSIX Pthreads and Mach C-threads.

Advantages :

1. Kernel mode privilege does not require for thread switching.
2. These threads are fast to create and manage.
3. User level thread works even if the OS does not support threads.
4. User level threads are more portable.
5. Threading library controls flow of thread.

Disadvantages :

1. If thread blocks, the Kernel may block the all threads.
2. Not suitable for multiprocessor system.
3. User level threads also do not support system wide scheduling priority.

2.7.2 Kernel Level Thread

- In Kernel level thread, thread management is done by Kernel. Operating systems support the Kernel level thread.
- Since Kernel managing threads, Kernel can schedule another thread if a given thread blocks rather than blocking the entire processes. Fig. 2.7.2 shows Kernel level thread.

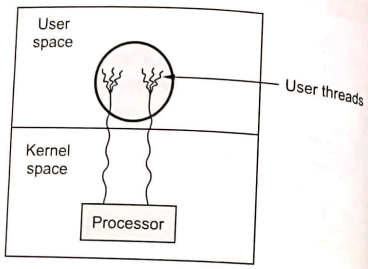


Fig. 2.7.2 Kernel level thread

- Kernel level thread support one to one thread mapping. This mapping requires each user thread with kernel thread. Operating system performs this mapping.
- Threads are constructed and controlled by system calls. The system knows the state of each thread.
- Thread management code is not included in the application code. It is only API to the Kernel thread. Windows operating system uses this facility.

Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

Kernel performs scheduling on a thread basis. The Kernel support for scheduling and management, thread creation only in Kernel space.

Kernel level threads are slower than user level threads.

Example : Windows 95/98/NT, Sun Solaris and Digital UNIX.

tages :

ach thread can be treated separately.

thread blocking in the Kernel does not block all other threads in the same rocess.

rnel routines itself as multithreaded.

Disadvantages :

1. Slower than the user level thread.
2. There will be overhead and increased in Kernel complexity.

2.7.3 Difference between User Level and Kernel Level Thread

Sr. No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to Kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.
4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called Kernel level threads.
5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.
6.	Example : POSIX Pthreads and Mach C-threads	Example : Windows 95/98/NT, Sun Solaris and Digital UNIX
7.	User level threads are also called many to one mapping thread.	Kernel level thread support one to one thread mapping.

2.8 Concept of Multithreads

- Operating system uses user level thread and kernel level thread. User level threads are managed without kernel support. Operating system support and manage the kernel level threads.
- Modern operating system support kernel level threads. Kernel performs multiple simultaneous tasks in operating system. In most of the application, user threads are mapped with kernel level threads.
- Different methods of mapping is used in operating system are as follows :
  1. One to one 2. Many to one 3. Many to many

1. One to one

- In this model, one user level thread maps with one kernel level thread. If there are three user threads then system creates three separate kernel thread. This model provides more concurrency because of separate thread. Fig. 2.8.1 shows one to one mapping.
- In this method, the operating system allocates data structure that represents kernel threads. Here multiple threads are run in parallel on multiprocessor system.

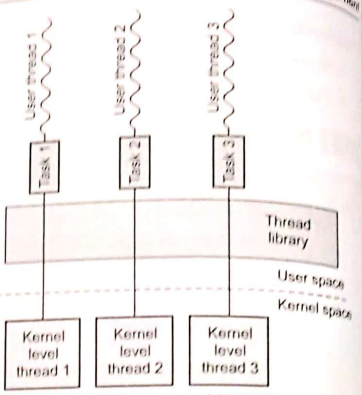


Fig. 2.8.1 One to one multithreading model

- Number of threads in the system increases, the amount of memory required is also increases.
- Windows 95/XP and Linux operating system uses this one to one thread mapping.
- Only overhead in this method is creation of kernel level thread for user level thread. Because of this overhead, system performance is slowdown.

2. Many to one

- Many to one mapping means many user thread maps with one kernel thread. Fig. 2.8.2 shows many to one mapping.
- Operating system blocks the entire multi-threaded process when a single thread blocks because the entire multi-threaded process is a single thread of control. So when operating system receive any a blocking I/O request, it blocks the entire process.

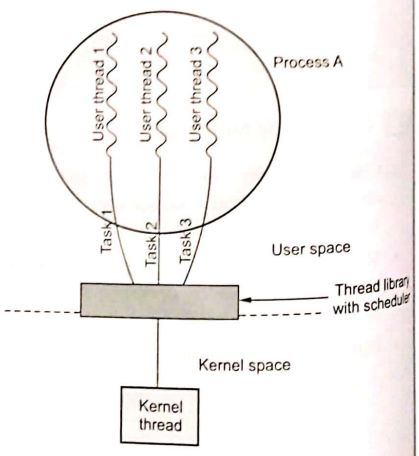


Fig. 2.8.2 Many to one multithreading model

- Thread library handle thread management in user space. The many-to-one model does not allow individual processes to be split across multiple CPUs.
- This type of relationship provides an effective context-switching environment, easily implementable even on simple kernels with no thread support.
- Green threads for Solaris and GNU portable threads implement the many-to-one model in the past, but few systems continue to do so today.
- **Advantage :** System performance is improved by customizing the thread library scheduling algorithm.

3. Many to many

- In the many to many mapping, many user-level threads maps with equal number of kernel threads. Fig. 2.8.3 shows many to many thread mapping.

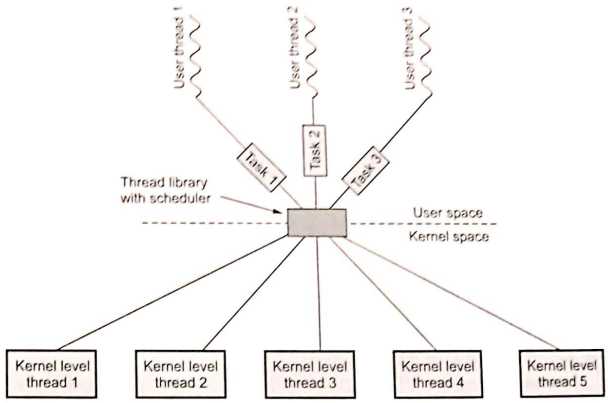


Fig. 2.8.3 Many to many multithreading model

- Many to many mapping is also called M-to-N thread mapping. Thread pooling is used to implement this method.
- Users can create required number of thread and there is no limitation for user. Again here blocking Kernel system calls do not block the entire process.
- This model support for splitting processes across multiple processors.
- **Limitations :** Operating system design becomes complicated.

2.9 Types of Scheduling

- CPU utilization is maximizing by using multiprogramming concept. Processor is not idle, it is executing a process. Processor scheduler selects one process for execution from the ready queue.

Scheduling Queue

- Scheduling queue is queue of processes or input-output devices. When the user process enters into the system, they put into the job queue. Job queue consists of all processes of the system.
- Operating system maintains different types of queues for different purposes. The queue are ready queue, device queue etc. Fig. 2.9.1 shows ready queue and device queue.

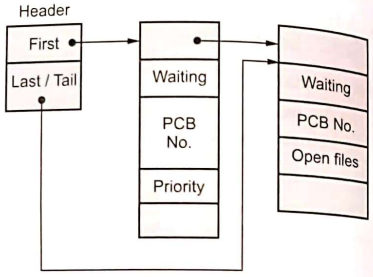


Fig. 2.9.1 Ready and device queue

- **Ready queue :** The processes which are ready and waiting to execute are kept in the ready queue. Ready queue is stored in main memory. Linked list is used for representing ready queue. Pointer field of PCB is used for this.
- **Device queue :** Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has maintained its own device queue.
- Process scheduling is represented by queueing diagram. Newly created process is kept into the ready queue and waits for processor. When CPU select the process for executing, following things happens :
  1. Process may require I/O device to perform operation. So it is put into the I/O queue.
  2. Process may create child process and wait for child process termination.
  3. Because of interrupt, process preempted from CPU and put into the ready queue.

2.9.1 Schedulers

- Schedulers are used to handles process scheduling. It is one type of system software and selects the jobs from the system and decide which process to run. Schedulers are of three types -

1. Long term scheduler 2. Short term scheduler 3. Medium term scheduler

- Fig. 2.9.2 shows queueing diagram for process scheduling.

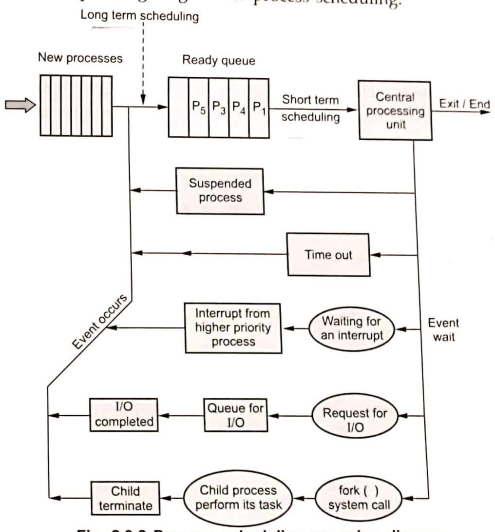


Fig. 2.9.2 Process scheduling queueing diagram

Long term scheduler

- Long term scheduler is also called job scheduler. It determines which process are admitted to the system for processing. Processes are selected from the queue and loads into the main memory for execution.
- Long term scheduling controls the degree of multiprogramming in multitasking systems. It provides a balanced mix of jobs, such as I/O bound and CPU bound.
- Long term scheduling is used in real time operating system. Time sharing operating system has no long term scheduler.

Medium term scheduler

- Medium term scheduler is part of swapping function. Sometimes it removes the process from memory. It also reduces the degree of multiprogramming.
- If process makes an I/O request and it is in memory then operating system takes this process into suspended states. Once the process becomes suspended, it cannot make any progress towards completion. In this situation, the process is removed from memory and makes free space for other process. The suspended process is stored in the secondary storage device i.e. hard disk. This process is called swapping.

Short term scheduler

- Short term scheduler is also called CPU scheduler. It selects the process fr queue which are ready to execute and allocate the CPU for execution.
- Short term scheduler is faster that long term scheduler. This scheduler m scheduling decisions much more frequently than the long-term or mid-te schedulers. A scheduling decision will at a minimum have to be made after e time slice, and these are very short.
- It is also known as dispatcher.

2.9.2 Difference between Long Term, Short Term and Medium Term Scheduler

Sr. No.	Long term	Short term	Medium term
1.	It is job scheduler.	It is CPU scheduler.	It is swapping.
2.	Speed is less than short term scheduler.	Speed is very fast	Speed is in between both.
3.	It controls the degree of multiprogramming.	Less control over degree of multiprogramming	Reduce the degree o multiprogramming.
4.	Absent or minimal in time sharing system.	Minimal in time sharing system.	Time sharing system us medium term scheduler.
5.	It select processes from pool and load them into memory for execution.	It select from among the processes that are ready to execute.	Process can b reintroduced into memor and its execution can b continued.
6.	Process state is (New to Ready).	Process state is (Ready to Running)	-
7.	Select a good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	-

2.9.3 Context Switch

- A context switch is the switching of the CPU from one process or thread another. A context is the contents of a CPU's registers and program counter at a point in time.
- Switching the CPU to another process requires performing a state save of t current process and a state restore of a different process. This task is known as **context switch**.
- A context switch can mean a register context switch, a task context switch thread context switch or a process context switch.

- A register is a small amount of very fast memory inside of a CPU that is used to speed the execution of computer programs by providing quick access to commonly used values.
- A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and which holds either the address of the instruction being executed or the address of the next instruction to be executed, depending on the specific system.
- Context switching can be described in more detail as the Kernel performing the following activities with regard to processes (including threads) on the CPU :
  1. Suspending the progression of one process and storing the CPU's state (i.e., the context) for that process somewhere in memory.
  2. Retrieving the context of the next process from memory and restoring it in the CPU's registers and
  3. Returning to the location indicated by the program counter in order to resume the process.
- Context switches can occur only in Kernel mode (system mode). Kernel mode is a privileged mode of the CPU in which only the Kernel runs and which provides access to all memory locations and all other system resources.
- Other programs, including applications, initially operate in user mode, but they can run portions of the Kernel code via system calls. Software context switching can be used on all CPUs and can be used to save and reload only the state that needs to be changed.
- To use the hardware context switch you need to tell the CPU where to save the existing CPU state and where to load the new CPU state from. The CPU state is always stored in a special data structure called a TSS (Task State Segment).
- Context switch times are highly dependent on hardware support. Context switching represents a substantial cost to the system in terms of CPU time and it can be the most costly operation on an operating system.
- There are three situations where a context switch needs to occur. They are multitasking, interrupt handling, user and Kernel mode switching.

2.10 Process Scheduling

- In a multiprogramming environment, usually more programs to be executed than could possibly be run at one time. In CPU scheduling, it switches from one process to another process. CPU resource management is commonly known as scheduling.
- Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundamental operating system functions.

- User program contains combination of CPU burst cycle and I/O burst cycle. Process starts with CPU burst cycle then I/O burst cycle again CPU burst cycle again I/O burst cycle. In this way the process completes its executions. Process will terminates after final CPU burst cycle completions.

Program execution sequence : CPU burst cycle → I/O burst cycle → CPU burst cycle → I/O burst cycle → CPU burst cycle → Program terminates.

- A CPU bound process tends to use the processor time that the system allocates to it. An I/O bound process tends to use the processor only briefly before generating an I/O request. The CPU bound processes spend most of their time using the processor.

CPU Scheduler

- CPU scheduler is also called as short scheduler. System programmers use parameters like program size, resources required for program and any other special devices are needed to determine scheduling policies.
- Scheduling mechanism is the part of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Scheduler is responsible for multiplexing processes on the CPU.

2.10.1 Preemptive and Non-preemptive Scheduling

- The scheduling policy determines when it is time for process to be removed from the CPU and which ready process should be allocated the CPU next. The scheduling mechanism is composed of several different parts, depending on exactly how it is implemented in any particular operating system.
- CPU scheduling is divided into two types : Preemptive scheduling and non-preemptive scheduling.
- **Preemptive scheduling :** A scheduling method that interrupts the processing of a process and transfers the CPU to another process is called a preemptive CPU scheduling. The process switches from running state to the ready state and waiting state to the ready state.
- **Non-preemptive scheduling :** Non-preemptive operation usually proceeds toward completion uninterrupted. Once the system has assigned a processor to a process the system cannot remove that processor from the process. The process switches from running state to the waiting state and termination of process.
- Preemptive scheduling increases the cost and it has higher overheads. This scheduling method is useful only in the high priority processes require rapid response.

- Non-preemptive scheduling is simple to implement. It requires some hardware platform. This method is attractive because of its simplicity. Windows 3.1 and Apple Macintosh operating system uses this scheduling method.

2.10.2 Difference between Preemptive and Non-preemptive Scheduling

Sr. No.	Preemptive scheduling	Non-preemptive scheduling
1.	Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process.	Non-preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.
2.	Preemptive scheduling incurs a cost associated with access shared data.	Non-preemptive scheduling does not increase the cost.
3.	It also affects the design of the operating system Kernel.	It does not affects the design of OS Kernel.
4.	Preemptive scheduling is more complex.	Simple, but very inefficient.
5.	Example : Round robin method.	Example : First come first serve method.

2.10.3 CPU Scheduling Criteria

- Depending on the system, CPU scheduling criteria will change.
  1. **Throughput :** CPU scheduling should attempt to service the maximum number of processes per unit time. The higher is the number, the more work is done by the system.
  2. **Waiting time :** The average period of time a process spends waiting. Process is normally in the ready queue in waiting time.
  3. **Turnaround time :** Turnaround time start from process submission to completion of process.

     Turnaround time = Burst time + Waiting time
  4. **Response time :** It is the time from the submission of a request until the first response is produced.
  5. **CPU utilization :** It is average function of time during which the processor is busy.
  6. **Fairness :** Avoid the process from the starvation. All the processes must be given equal opportunity to execute.

7. **Priority :** If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher priority processes.
8. **Predictability :** A given process always should run in about the same amount of time under similar system loads.

- Depending upon the nature of operations the scheduling policy may differ. The CPU utilization and throughput are the system centered parameters. Fairness is affect by user and system.

2.11 Scheduling Algorithms

GTU : Winter-13, Summer-13,14

- CPU scheduling algorithms are as follows :
  1. First In First Out (FIFO) scheduling
  2. Shortest job first scheduling
  3. Priority scheduling
  4. Round robin scheduling

2.11.1 First Come First Serve Scheduling

- FIFO is also called first come first served scheduling method. The processes are dispatched according to their arrival time at the ready queue. FIFO is non-preemptive CPU scheduling algorithm.
- FIFO is simple to implement because it uses a FIFO queue. This algorithm is fine for most of the batch operating systems. FIFO is not useful in scheduling interactive processes because it cannot guarantee short response time.
- When new process enters into the system, its process control block is linked to the end of the ready queue and it is removed from the front of the queue.
- Turnaround time is unpredictable with the FIFO algorithm. Average waiting time of the FCFS is often quite long. Real life analogy is buying tickets.

Example 2.11.1 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time when the processes arrive in the following order :*

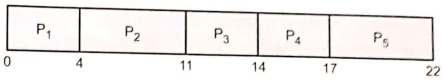
a. P_1, P_2, P_3, P_4, P_5

Provide the Gantt chart for the same.

Process	*Burst time*
P_1	4
P_2	7
P_3	3
P_4	3
P_5	5

Solution :

Gantt chart :



Such diagram is called Gantt charts, showing when each process burst uses the CPU.

Waiting time :

Process	Waiting time
P_1	0 – 0 = 0
P_2	4 – 0 = 4
P_3	11 – 0 = 11
P_4	14 – 0 = 14
P_5	17 – 0 = 17

Average waiting time $= \frac{0+4+11+14+17}{5} = \frac{46}{5} = 9.2$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	0 + 4 = 4
P_2	4 + 7 = 11
P_3	11 + 3 = 14
P_4	14 + 3 = 17
P_5	17 + 5 = 22

Average turnaround time $= \frac{4+11+14+17+22}{5} = \frac{68}{5} = 13.60$

Convey effect : To reduce I/O device utilization, all I/O bound processes will be waiting excessively long for processor bound ones. This is called as convey effect.

- If one process monopolizes the system, the extent of its overall effect of system performance depends on the scheduling policy and whether the process is processor bound or I/O bound.

2.11.2 Shortest Job First Scheduling

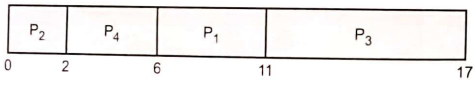
- Shortest job first scheduling algorithm is also known as Shortest Job Next (SJ scheduling algorithm. It handles the process based on length of their CPU cy time. It reduces average waiting time over FIFO algorithm.
- SJF is a non-preemptive CPU scheduling algorithm.
- It does not work in interactive system because users do not estimate in advan the CPU time required to run their processes.
- SJF scheduling algorithm is used frequently in long term scheduling.

Example 2.11.2 *Calculate the average waiting time and average turnaround time. Provide th Gantt chart for the same.*

Process	*Burst time*
P_1	5
P_2	2
P_3	6
P_4	4

Solution :

Gantt chart :



Waiting time :

Process	Waiting time
P_1	6 – 0 = 6
P_2	0 – 0 = 0
P_3	11 – 0 = 11
P_4	2 – 0 = 2

Average waiting time $= \frac{6+0+11+2}{4} = \frac{19}{4} = 4.75$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	6 + 5 = 11
P_2	0 + 2 = 2
P_3	11 + 6 = 17
P_4	2 + 4 = 6

Average turnaround time $= \frac{11+2+17+6}{4} = \frac{36}{4} = 9$

- The SJF scheduling algorithm is optimal only when all of the processes are available at the same time and the CPU estimates are available.
- SJF scheduling algorithm may be **preemptive or non-preemptive.**

2.11.3 Priority Scheduling

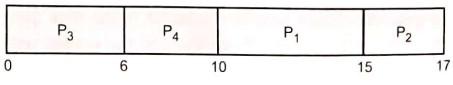
- Priority CPU scheduling algorithm is preemptive and non-preemptive algorithm. It is one of the most common scheduling algorithm in batch system. Priority can be assigned by a system admin using characteristics of the process.
- In this scheduling algorithm, CPU select higher priority process first. If the priority of two process is same then FCFS scheduling algorithm is applied for solving the problem.

Example 2.11.3 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same. (Time slice = 2)*

Process	*Burst time*	*Priority*
P_1	5	3
P_2	2	4
P_3	6	1
P_4	4	2

Solution : For non-preemptive priority scheduling

Gantt chart :



Waiting time :

Process	Waiting time
P_1	10 – 0 = 10
P_2	15 – 0 = 15
P_3	0 – 0 = 0
P_4	6 – 0 = 6

Average waiting time = $\frac{10+15+0+6}{4} = \frac{31}{4}$ = **7.75**

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	10 + 5 = 15
P_2	15 + 2 = 17
P_3	0 + 6 = 6
P_4	6 + 4 = 10

Average turnaround time = $\frac{15+17+6+10}{4} = \frac{48}{4}$ = **12**

Problem with Priority Scheduling

- Waiting time is more for lower priority process even if there required CPU burst time is less. Priority scheduling algorithm faces the starvation problem. Starvation problem can be solved by using **aging techniques.**
- In aging, the priority of the process will increase which is waiting for long time in the ready queue.

2.11.4 Round Robin Scheduling

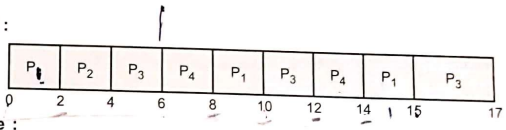
- Round robin is a preemptive scheduling algorithm. It is used in interactive system.
- Here process are given a limited amount of time of processor time called a time slice or time quantum. If a process does not complete before its quantum expires the system preempts it and gives the processor to the next waiting process. The system then places the preempted process at the back of the ready queue.
- Processes are placed in the ready queue using a FIFO scheme. With the RR algorithm, the principle design issue is the length of the time quantum to be used. For short time slice, processes will move through the system relatively quickly. It increases the processing overheads.

Example 2.11.4 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same. (Time slice = 2)*

Process	Burst time
P_1	5
P_2	2
P_3	6
P_4	4

Solution :

Gantt chart :



Waiting time :

Process	Waiting time
P_1	0 – 0 + 8 – 2 + 14 – 10 = 10
P_2	2 – 0 = 2
P_3	4 – 0 + 10 – 6 + 15 – 12 = 11
P_4	6 – 0 + 12 – 8 = 10

Average waiting time = $\frac{10+2+11+10}{4} = \frac{33}{4}$ = **8.25**

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	10 + 5 = 15
P_2	2 + 2 = 4
P_3	11 + 6 = 17
P_4	10 + 4 = 14

Average turnaround time = $\frac{15+4+17+14}{4} = \frac{50}{4}$ = **12.5**

2.11.5 Comparison between FCFS and RR

Sr. No.	FCFS	Round robin
1.	FCFS decision made is non-preemptive.	RR decision made is preemptive.
2.	It has minimum overhead.	It has low overhead.
3.	Response time may be high.	Provides good response time for short processes.
4.	It is troublesome for time sharing system.	It is mainly designed for time sharing system.
5.	The workload is simply processed in the order of arrival.	It is similar like FCFS but uses time quantum.
6.	No starvation in FCFS.	No starvation in RR.

2.11.6 Comparison of CPU Scheduling Algorithm

Algorithm	Policy type	Used in	Advantages	Disadvantages
FCFS	Non-preemptive	Batch	1. Easy to implement. 2. Minimum overhead.	1. Unpredictable turn around time. 2. Average waiting is more.
RR	Preemptive	Interactive	1. Provides fair CPU allocation. 2. Provides reasonable response times to interactive users.	1. Requires selection of good time slice.
Priority	Non-preemptive	Batch	1. Ensures fast completion of important jobs.	1. Indefinite postponement of some jobs. 2. Faces starvation problem.
SJF	Non-preemptive	Batch	1. Minimizes average waiting time. 2. SJF algorithm is optimal.	1. Indefinite postponement of some jobs. 2. Cannot be implemented at the level of short term scheduling. 3. Difficulty is knowing the length of the next CPU request.
Multilevel queues	Preemptive or Non-preemptive	Batch/Interactive	1. Flexible. 2. Gives fair treatment to CPUbound jobs.	1. Overhead incurred by monitoring of queues.

Example 2.11.5 *Consider the following five processes, with the length of the CPU burst time given in milliseconds.*

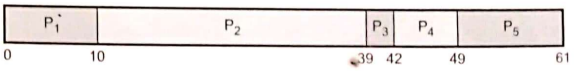
Process	Burst time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Consider the First Cum First Serve (FCFS), Non-Preemptive Shortest Job First (SJF), Round Robin (RR) (quantum = 10 milliseconds) scheduling algorithms. Illustrate the scheduling using Gantt Chart. Which algorithms will give the minimum average waiting time ? Discuss.

Solution :

i) FCFS

Gantt chart



Waiting time

$P_1 = 0$

$P_2 = 10$

$P_3 = 39$

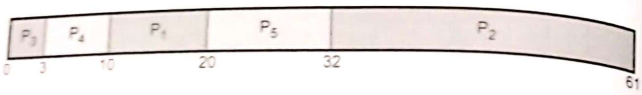
$P_4 = 42$

$P_5 = 49$

Average waiting time = $\frac{0+10+39+42+49}{5}$ = 28 milliseconds

ii) Non-preemptive SJF

Gantt chart :



Waiting time

$P_1 = 10$

$P_2 = 32$

$P_3 = 0$

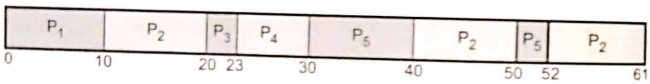
$P_4 = 3$

$P_5 = 20$

Average waiting time = $\frac{10+32+0+3+20}{5} = \frac{65}{5}$ = 13 millisecond

iii) RR

Gantt chart :



Waiting time

$P_1 = 0$

$P_2 = 10 + (40 - 20) + (52 - 50) = 32$

$P_3 = 20$

$P_4 = 23$

$P_5 = 30 + (50 - 40) = 40$

Average waiting time = $\frac{0+32+20+23+40}{5} = \frac{115}{5}$ = 23 millisecond

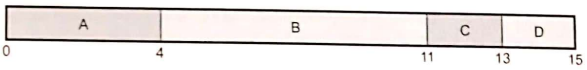
Non-preemptive SJF gives minimum average waiting time as compared to FCFS and RR method. In non-preemptive SJF, process assign to CPU that has the smallest next CPU burst.

Example 2.11.6 *Explain the various CPU scheduling techniques with Gantt charts clearly as indicated by (process name, arrival time, process time) for the following (A, 0, 4), (B, 2, 7) (C, 3, 2) and (D, 2, 2) for FCFS, SJF, SRT, RR with quantum 1 and 2.*

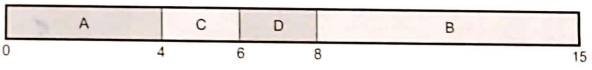
Solution :

Process Name	Arrival time	Process time
A	0	4
B	2	7
C	3	2
D	3	2

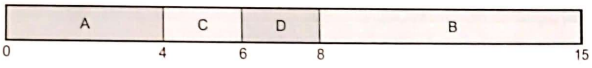
i) FCFS



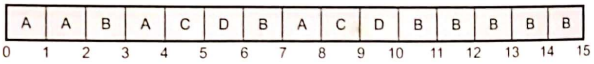
ii) SJF (Non-preemptive)



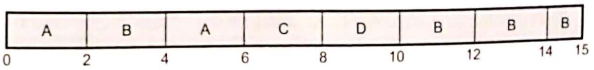
iii) SRT



iv) RR (quantum = 1)



v) RR (quantum = 2)



Example 2.11.7 *Consider the following set of processes, with the length of the CPU burst time given in milliseconds.*

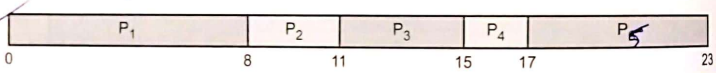
Process	Burst time	Priority
P_1	8	3
P_2	3	1
P_3	4	4
P_4	2	2
P_5	6	5

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4 and P_5 all at time '0'.

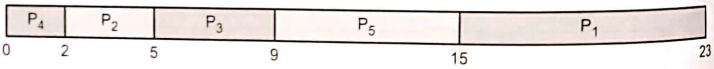
1) *Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, non-preemptive priority (a smaller priority number implies a higher priority and RR (quantum = 2) scheduling.*
2) *What is the turn around time of each process for each of the scheduling algorithms in part (1).*
3) *What is the waiting time of each process for each of the scheduling algorithms in part (1).*
4) *Which of the schedules in part (1) results in the minimal average waiting time (over all processes).*

Solution :

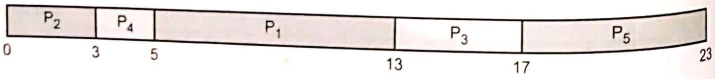
1. Gantt chart : a) FCFS



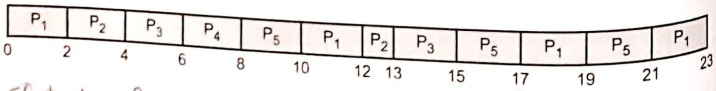
b) SJF



c) Non-preemptive priority



d) RR



2) Waiting time

a) FCFS

Process	Waiting time
P_1	00
P_2	8
P_3	11
P_4	15
P_5	17
Average waiting time	51/5 = 10.2

b) SJF

Process	Waiting time
P_1	15
P_2	00
P_3	13
P_4	03
P_5	09
Average waiting time	34/5 = 6.8

c) Non-preemptive priority

Process	Waiting time
P_1	05
P_2	00
P_3	13
P_4	03
P_5	17
Average waiting time	38/5 = 7.6

d) RR

Process	Waiting time
P_1	15
P_2	10
P_3	11
P_4	06
P_5	15
Average waiting time	57/5 = 11.4

3) Turn around time :

a) FCFS

Process	Waiting time
P_1	08
P_2	11
P_3	15
P_4	17
P_5	13

b) SJF

Process	Waiting time
P_1	23
P_2	05
P_3	09
P_4	05
P_5	15

c) Non-preemptive priority

Process	Waiting time
P_1	13
P_2	03
P_3	17

P_4	05
P_5	23

d) RR

Process	Waiting time
P_1	23
P_2	13
P_3	15
P_4	08
P_5	21

4) Average waiting time

Scheduling methods	Average waiting time
FCFS	10.2
SJF	6.8 (Minimal)
Non-preemptive priority	7.6
RR	11.4

Example 2.11.8 *Assume the following workload in a system. All jobs arrive at time 0 in the order given*

Job	*Burst time (ms)*	*Priority*
A	8	2
B	4	1
C	5	4
D	2	2
E	1	3

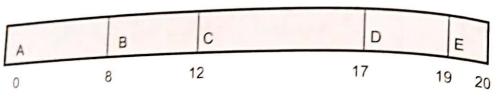
i) Draw a Gantt chart illustrating the execution of these job using FCFS, RR (quantum = 4), non-preemptive priority (a smaller priority number implies a higher priority) and SJF CPU scheduling.

ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm.

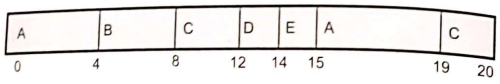
Solution :

i) Gantt chart

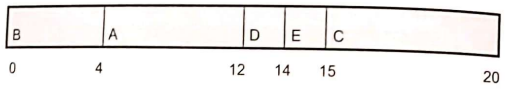
1) FCFS



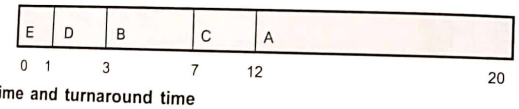
2) RR (quantum = 4)



3) Non - preemptive priority



4) SJF



ii) Waiting time and turnaround time

Job	Waiting time			Turnaround time		
	FCFS	RR	SJF	FCFS	RR	SJF
A	0	11	12	8	19	20
B	8	4	3	12	8	7
C	12	15	7	17	20	12
D	17	12	1	14	14	3
E	19	14	0	20	15	1

Non preemptive priority

Job	Waiting time	Turnaround time
A	4	12
B	0	4
C	15	20
D	12	14
E	14	15

Average waiting time :

i) FCFS = 11.20

ii) RR = 11.20

iii) SJF = 4.6

iv) Non preemptive priority = 9

Average turnaround time

i) FCFS = 14.20

ii) RR = 15.20

iii) SJF = 8.6

iv) Non preemptive priority = 13

Example 2.11.9 *Assume the following processes arrive for execution at the time indicated and also mention with the length of the CPU-burst time given in milliseconds.*

Job	*Burst time (ms)*	*Priority*	*Arrival time (ms)*
A	10	5	0
B	6	2	0
C	7	4	1
D	4	1	1
E	5	3	2

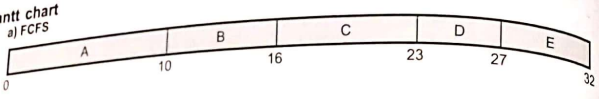
i) Give a Gantt chart illustrating the execution of these processes using FCFS, Round Robin (quantum = 5), and Priority (Preemptive and Non Preemptive).

ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm.

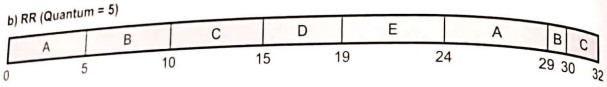
Solution :

i) Gantt chart

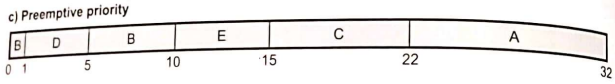
a) FCFS



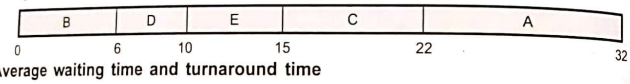
b) RR (Quantum = 5)



c) Preemptive priority



d) Non preemptive priority



ii) Average waiting time and turnaround time

Waiting time and turnaround time :

Job	Waiting time				Turnaround time			
	FCFS	RR	Preemptive Priority	Nonpreemptive priority	FCFS	RR	Preemptive priority	Nonpreemptive priority
A	00	19	22	22	10	29	32	32
B	10	24	04	00	16	30	10	06
C	15	24	14	14	22	31	21	21
D	22	14	00	05	26	18	04	09
E	25	17	08	08	30	22	13	13

Average waiting time and average turnaround time

Method	Average waiting time	Average turnaround time
FCFS	$= \frac{00+10+15+22+25}{5} = \frac{72}{5} = 14.4$	$\frac{10+16+22+26+30}{5} = \frac{104}{5} = 20.8$
RR	$= \frac{19+24+24+14+17}{5} = \frac{98}{5} = 19.6$	$\frac{29+30+31+18+22}{5} = \frac{130}{5} = 26$

Preemptive priority	$= \frac{22+4+14+0+8}{5} = \frac{48}{5} = 9.6$	$\frac{32+10+21+4+13}{5} = \frac{80}{5} = 16$
Non-preemptive priority	$= \frac{22+0+14+5+8}{5} = \frac{49}{5} = 9.8$	$\frac{32+06+21+9+13}{5} = \frac{81}{5} = 16.2$

Example 2.11.10 *Assume the following processes arrive for execution at the time indicated and also mention with the length of the CPU-burst time given in milliseconds.*

Job	*Burst time (ms)*	*Priority*	*Arrival time (ms)*
P_1	6	2	0
P_2	2	2	1
P_3	3	4	1
P_4	1	1	2
P_5	2	3	2

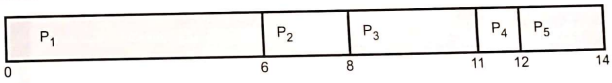
i) Give a Gantt chart illustrating the execution of these processes using FCFS, Round Robin (quantum = 1), and Priority (Preemptive and Non-preemptive).

ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm.

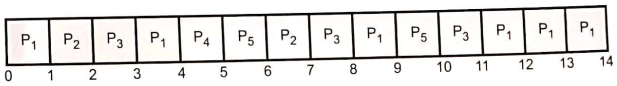
Solution :

i) Gantt chart

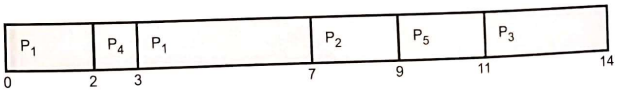
a) FCFS



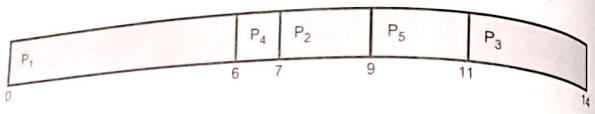
b) Round Robin (quantum = 1)



c) Preemptive priority



d) Non-preemptive priority



ii) Waiting time and turnaround time

Process	Waiting time			
	FCFS	RR	Preemptive priority	Non-preemptive priority
P_1	0	8	1	0
P_2	5	4	6	6
P_3	7	7	10	10
P_4	9	2	0	4
P_5	10	6	7	7
Average waiting time	6.2	5.4	4.8	5.2

Turnaround time

Process	Waiting time			
	FCFS	RR	Preemptive priority	Non-preemptive priority
P_1	6	14	7	6
P_2	7	6	8	8
P_3	10	10	13	13
P_4	10	3	1	5
P_5	12	8	9	9
Average waiting time	9	8.2	7.6	8.2

Example 2.11.11 *Consider the following set of processors with the length of CPU burst time given in milliseconds.*

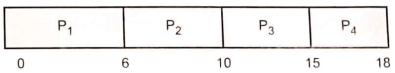
Process	*Arrival time*	*Burst time*
P_1	0	6
P_2	1	4
P_3	3	5
P_4	5	3

Draw the Gantt charts illustrating the execution of these processes using SJF (preemptive and non-preemptive) and FCFS. Calculate average turnaround time, average waiting time in each case. PU - May-14

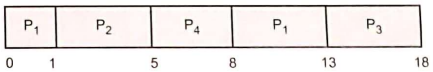
Solution :

i) FCFS

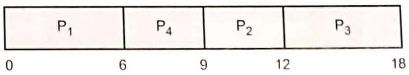
Gantt chart



ii) SJF (preemptive)



iii) SJF (Non-preemptive)



	FCFS		SJF Preemptive		SJF Non-preemptive	
Process	Waiting time	Turnaround time	WT	TT	WT	TT
P1	0	6	7	13	0	6
P2	5	9	0	4	8	12
P3	7	12	10	15	10	15
P4	10	13	0	3	1	4
Total	22	40	17	35	19	37

Average waiting time

$$\text{FCFS} = \frac{22}{4} = 5.5$$

$$\text{Preemptive SJF} = \frac{17}{4} = 4.25$$

$$\text{Non-preemptive SJF} = \frac{19}{4} = 4.75$$

Average turnaround time

$$\text{FCFS} = \frac{40}{4} = 10$$

$$\text{Preemptive SJF} = \frac{35}{4} = 8.75$$

$$\text{Non-preemptive SJF} = \frac{37}{4} = 9.25$$

University Questions

1. *Consider the following set of processes :* **PU : May-12**

Processes	Arrival time	Processing time
P_1	0	7
P_2	3	2
P_3	4	3
P_4	4	1
P_5	5	3

Draw a Gantt Chart and find out average waiting-time and average turn-around time for

i) FCFS ii) SJF iii) RR

2. *For the following snapshot of process calculate the TAT, WT* **PU : Dec.-13**

Process	AT	BT	Priority
A	0	3	2
B	1	6	1
C	4	4	3
D	6	2	4

Using FCFS, SJF (Both) RR Tq : 2 ms

3. *Draw Gantt chart and calculate turnaround time, wait time for following processes using FCFS, SJF nonpreemptive and Round Robin CPU scheduling algorithms. (Round robin time quantum = 2)* **PU : May-13**

Process	Arrival time	Burst time
P_1	0	8
P_2	1	5
P_3	3	3
P_4	4	1
P_5	6	4

4. *Consider the following set of processes with length of CPU burst time given in milliseconds.*

Process	Burst Time	Priority
P1	10	5
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Assume arrival order is : P1, P2, P3, P4, P5 at time 0, 1, 2, 3, 4 respectively and a smaller priority number implies a higher. Draw the Gantt charts for preemptive and non-preemptive priority scheduling. Calculate average turnaround time and average waiting time. **GTU : Winter-13, Marks 7**

5. *Explain scheduling of process with shortest process next policy.* **GTU : Summer-13, Marks 7**

6. *Find average waiting time for shortest job first scheduling and round robin scheduling algorithm.*

Process	CPU butst time
P1	6
P2	8
P3	5
P4	2

CPU burst time is given in millisecond and time quantum is 4. **GTU : Summer-14, Marks 7**

7. Consider the following set of processes with the length of CPU burst time given in
milliseconds.

Process	Arrival time	Burst time	Priority
P1	0	8	3
P2	1	1	1
P3	2	3	2
P4	3	2	3
P5	4	6	4

Calculate average turnaround time and average waiting time for First-come first served schedul
Shortest job first scheduling and Priority scheduling algorithm. **GTU : Summer-14, Marks**

2.12 Multiprocessor Scheduling

- Designs of processor scheduling parameters are changed when more than
processor is used.
- Classification of multiprocessor systems are as follows :

1. **Loosely coupled :** It is also called distributed multiprocessor or cluster. It
collection of autonomous systems with own main memory and I/O channels.
2. **Functionally specialized processors :** It consists of a master, general-purp
processor. Master processors are controlled the system and provides services.
example is an I/O processor
3. **Tightly coupled multiprocessor :** It consists of a set of processors that shar
common main memory. The popular multi-core architecture falls into this categ

2.12.1 Issue Relating to the Scheduling

1. **Granularity :** It is one of the characteristics of the multiprocessor system. It is us
with frequency of synchronization between processes in a system. We
distinguish five categories of parallelism that differ in the degree of granularity.

a. **Independent parallelism :** Each process in the system represents a separ
independent application. There is no synchronization among processes. Used
time sharing machine.

b. **Coarse and very coarse-grained parallelism :** Minimum synchronizati
among the processes. Set of concurrent processes running on
multi-programmed uni-processor system uses this type of parallelism.

c. **Medium-grained parallelism :** High degree of coordination is needed among those threads, which leads to a medium-grained parallelism.

d. **Fine-grained parallelism** represents a much more complex use of parallelism, and remains a very difficult area.

2.12.2 Various Design Issues

- Scheduling on a multiprocessor involves three issues:
  a. The assignment of processes to processors
  b. The use of multiprogramming on individual processors
  c. The actual dispatching of a process

1. Processor assignment

- Regarding the actual assignment, at least two approaches can be followed :
  a) With a master/slave approach, key kernel functions, including the scheduler of the operating system, always run on a "master" processor, while the rest of the processors are used to run user processes. When a slave process needs some service, it simply sends a request to the master processor, and waits for its response. This approach is very simple and does not need a conflicting resolution mechanism. But, the master can become a bottleneck, and it can even bring down the whole system when it fails.
  b) **In a peer structure,** the OS can execute on any processor, and each of them does its own scheduling among the available processes.

2. Use of multiprogramming on individual processors

- For coarse-grained, or independent processes, each process contains a large amount of instructions, thus, it is much more likely that a processor can be idle when processes are blocked for various service. It is necessity to processor to switch among processes to have a better performance. But, for a medium-grained applications running in an environment with many processors available, it is no longer that important to keep all processors busy all the time.
- A key design issue for a multiprocessor scheduling is the actual selection of a process for execution. In a multi-programmed environment, more sophisticated scheduling, based on such factors as priority and past usage, may lead to a better performance, compared with simpler scheduling algorithm such as FCFS.
- But, a sophisticated algorithm often leads to overhead, thus unnecessary or even counter-productive for the overall performance.
- In most traditional multiprocessor systems, processes are not associated with dedicated processors. But, a single queue for all the processors is used to serve all the competing processes.

- If priority is an important issue, then several queues will be used, each of takes care of a class of processes with the same priority.
- Simple FCFS strategy or the use of FCFS coupled with a static priority may suffice for a multiprocessor system.

2.12.3 Thread Scheduling

- An application can be implemented as a set of threads, which cooperate execute concurrently in the same address space. Four general approach multiprocessor thread scheduling are as follows :

1. Load sharing
2. Gang scheduling
3. Dedicated processor scheduling
4. Dynamic scheduling

1. Load sharing

- Threads are not assigned to a particular processor. When a processor is idle selects a thread from a global queue serving all processors.
- In this method, load is evenly distributed among processors. No centra scheduler is required.
- This policy is implemented in different ways: FCFS and priority method is u These are depends upon queue structure.
- Three different versions of load sharing :
  a. **First-Come-First-Served (FCFS) :** On arrival of the thread, it is placed at end of the shared queue. Next ready thread is selected when processor is Processor executes the thread until its completion or blocking.
  b. **Smallest number of threads first :** Shared ready queue is organized accord to the priority. Processor selects highest priority thread with the smal number of unscheduled threads. If the priority of the two thread is same, t FCFS method is applied.
  c. **Preemptive smallest number of threads first :** Highest priority is given to with the smallest number of unscheduled threads.

Advantages of Load sharing

1. No centralized scheduler is required.
2. Load distribution is equal.
3. Global queue is maintained by the system.

Disadvantages of Load sharing

1. It enforces mutual exclusion because of memory.

2. High degree of coordination is required between the threads of a program.
3. Caching becomes less efficient.

2. Gang scheduling

- With gang scheduling, a set of related threads is scheduled to run on a set of processors at the same time, on a one-to-one basis.
- Synchronization blocking may be reduced or increased.
- If closely related processes executes in parallel, synchronization blocking may be reduced.
- Set of related threads is scheduled to run on a set of processors.
- Gang scheduling has three parts -
  1. Group of related threads are scheduled as a unit, a gang.
  2. All members of a gang run simultaneously on different timeshared CPUs.
  3. All gang members start and end their time slices together.
- The trick that makes gang scheduling work is that all CPU are scheduled synchronously. This means that time is divided into discrete quanta.
- An example of how gang scheduling works is given in the Fig. 2.12.1. Here we have a multiprocessor with six CPU being used by five processes, A through E, with a total of 24 ready threads.

		CPU					
		0	1	2	3	4	5
Time slot	0	A_0	A_1	A_2	A_3	A_4	A_5
	1	B_0	B_1	B_2	C_0	C_1	C_2
	2	D_0	D_1	D_2	D_3	D_4	E_0
	3	E_1	E_2	E_3	E_4	E_5	E_6
	4	A_0	A_1	A_2	A_3	A_4	A_5
	5	B_0	B_1	B_2	C_0	C_1	C_2
	6	D_0	D_1	D_2	D_3	D_4	E_0
	7	E_1	E_2	E_3	E_4	E_5	E_6

Fig. 2.12.1 Gang scheduling

- During time slot 0, threads A_0 through A_5 are scheduled and run.
- During time slot 1, threads B_0, B_1, B_2, C_0, C_1, C_2 are scheduled and run.

- During time slot 2, D's five threads and E_0 get to run.
- The remaining six threads belonging to process E run in the time slot 3. The
- cycle repeats, with slot 4 being the same as slot 0 and so on.
- Gang scheduling is useful for applications where performance severly degra
- when any part of the application is not running.

3. Dedicated processor assignment

- When application is scheduled, its threads are assigned to a processor.
- Some processor may be idle and no multiprogramming of processors.
- Provides implicit scheduling defined by assignment of threads to processors. F
- the duration of program executation, each program is allocated a set of proces
- equal in number to the number of threads in the program. Processors are ch
- from the available pool.

4. Dynamic scheduling

- Number of threads in a process are altered dynamically by the application.
- Operating system and the application are involved in making scheduling decisi
- The OS is responsible for partitioning the processors among the jobs.
- Operating system adjusts load to improve the use of following parameters :
  1. Assign idle processors.
  2. New arrivals may be assigned to a processor that is used by a job curre
  - using more than one processor.
  3. Hold request until processor is available.
  4. New arrivals will be given a processor before executing running applicatio

2.13 Performance Evaluation of the Scheduling

GTU : Summer-1

- Performance evaluation of the CPU scheduling is required for selection of t
- algorithm. The maximizing CPU utilization and maximizing throughput are t
- parameters used for evaluation of the scheduling algorithm.
- Following are the different evaluation methods used for this :
  1. Deterministic Modeling
  2. Queueing Models
  3. Simulations
  4. Implementation

2.13.1 Deterministic Modeling

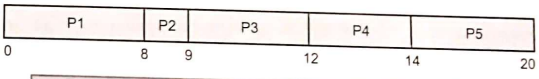
- Deterministic modeling uses an analytic method. It takes a particular predetermined workload and defines the performance of each algorithm for that workload. It gives real calculation of each case.

Example 2.13.1 *Consider the following set of processes with the length of CPU burst time given in the milliseconds.*

Process	Arrival time	Burst time	Priority
P1	0	8	3
P2	1	1	1
P3	2	3	2
P4	3	2	3
P5	4	6	4

Calculate average turnaround time and average waiting time for first come first served, Shortest job first and priority algorithm.

Solution : For the FCFS the processes are executed as follows :

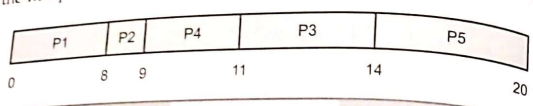


Process	Waiting time	Turnaround time
P1	0	8
P2	7	8
P3	7	10
P4	9	11
P5	10	16

$$\text{Average waiting time} = \frac{0+7+7+9+10}{5} = 6.6$$

$$\text{Average waiting time} = \frac{8+8+10+11+16}{5} = 10.6$$

- For the non-preemptive SJF the processes are executed as follows :



Process	Waiting time	Turnaround time
P1	0	8
P2	7	8
P3	9	12
P4	6	8
P5	10	16

$$\text{Average waiting time} = \frac{0+7+9+6+10}{5} = 6.4$$

$$\text{Average waiting time} = \frac{8+8+12+8+16}{5} = 10.4$$

- For the non-preemptive priority scheduling the processes are executed same FCFS algorithm. So average waiting is same as FCFS waiting time.
- The average waiting time of SJF is small as compared to FCFS and pri scheduling algorithm.
- Deterministic modeling is simple to implement and accurate also.

2.13.2 Queueing Models

- Each of us has spent a great deal of time waiting in lines. In this chapter, develop mathematical models for waiting lines, or queues. Queuing theory is mathematics of waiting lines. It is extremely useful in predicting and evaluat system performance.
- Queuing theory has been used for operations research. Traditional queuing the problems refer to customers visiting a store, analogous to requests arriving device.
- The theory of queuing systems was developed to provide models for forecast behaviors of systems subject to random demand. The first problems addres concerned congestion of telephone traffic. Erlang observed that a telephone sys can be modeled by Poisson customer arrivals and exponentially distributed ser times.

- A queuing system can be described as follows :
"Customers arrive for a given service, wait if the service cannot start immediately and leave after being served".
- The term "customer" can be men, products, machines, ...
- The Queuing models are very helpful for determining how to operate a queuing system in the most effective way if too much service capacity to operate the system involves excessive costs. The models enable finding an appropriate balance between the cost of service and the amount of waiting.
- Following Fig. 2.13.1 shows the queueing system.

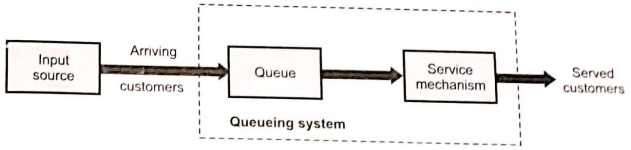


Fig. 2.13.1 Queueing system

- The basic queueing process is as follows :
1. Customers are generated over time by an input source.
2. The customers enter a queueing system.
3. A required service is performed in the service mechanism.
- **Distribution of arrivals :** Generally , the arrival of customers into the system is a random event. Frequently the arrival pattern is modeled as a Poisson process.
- **Distribution of service times :** Service time is also usually a random variable. A distribution commonly used to describe service time is the exponential distribution.
- **Queue discipline :** Most common queue discipline is first served (FCFS). Other disciplines assign priorities to the waiting units and then serve the unit with the highest priority first.

2.13.3 Simulations

- Simulations create a model of the system, then process statistical or real data. A component of the system is the major software data structures.
- Clock represents the variable in the system. Variable values are increases for reflecting system state.
- Simulator executes the algorithm and algorithm performances are gathered and printout is taken.

- Required data is generated in different ways. Random number generator one the method for generating input for the system. Random number generat generates the data for processes, CPU burst times, arrival times, departures time priority etc.
- The distribution can be defined **mathematically or empirically**.
- **Empirically :** Measurements of the actual system under study are taken. In system, results are defining the distribution of events. Simulation uses this re for calculating output.
- Simulation methods are expensive and time consuming.

Example 2.13.3 *Find average waiting time for shortest job first scheduling and round robin scheduling algorithm.*

Process	CPU burst time
P1	6
P2	8
P3	5
P4	2

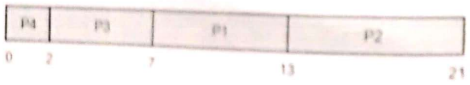
CPU burst time is given in millisecond and time quantum is 4.

GTU : Summer-14, Marks 7

Solution :

1) Shortest job first scheduling (Non-preemptive)

Gantt chart :



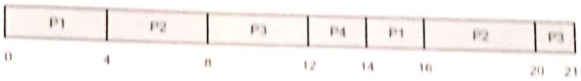
Average waiting time :

Process	Waiting time
P1	7
P2	13
P3	2
P4	0

$$\text{Average waiting time} = \frac{7+13+2+0}{4} = \frac{22}{4} = 5.5$$

2. Round Robin (time quantum = 4)

Gantt chart :



Average waiting time :

Process	Waiting time
P1	0 + (14 - 4) = 10
P2	4 + (16 - 8) = 12
P3	8 + (20 - 12) = 16
P4	12

Average waiting time : $\frac{10+12+16+12}{4} = \frac{50}{4} = 12.5$

❑❑❑

Notes

3 Interprocess Communication

Syllabus

Race conditions, Critical section, Mutual exclusion, Hardware solution, Strict alternation, Peterson's solution, The producer consumer problem, Semaphores, Event counters, Monitors, Message passing, Classical IPC problems : Reader's and writer problem, Dinning philosopher problem etc., Scheduling , Scheduling algorithms

Contents

3.1 Introduction of Concurrent Processes ..... **Winter-13, Summer-13,** ... Marks 4
3.2 Hardware Solution
3.3 Strict Alternation
3.4 Lock
3.5 Mutex
3.6 Peterson's Solution ..... **Summer-13,** ..... Marks 7
3.7 Producer - Consumer Problem ..... **Summer-14,** ..... Marks 7
3.8 Semaphore **Winter-13,**
..... **Summer-13, 14,** ..... Marks 7
3.9 Dekker's Solution
3.10 Classical IPC Problem
3.11 Monitors ..... **Winter-13, Summer-14,** ... Marks 7
3.12 Inter Process Communication Models

3.1 Introduction of Concurrent Processes

GTU : Winter-18, Summer-19

- Logical control flows are concurrent if they overlap in time. This gen... phenomenon is known as **concurrency**. Concurrency refers to any form... interaction among processes or threads. The familiar examples are hardw... exception handlers, processes and UNIX signal handlers.
- Modern operating systems can handle more than one process at a time. Sys... scheduler manages processes and their competition for the CPU. The rol... memory manager is to manage sharing of main memory between active proce... in the system. Look at how processes coexist and communicate with each other... modern computer.
- Concurrency is good for users because working on the same prob... simultaneous execution of programs, background execution.
- Concurrency creates problems also because of access to shared data structu... deadlock due to resource contention and enabling process interaction.
- Applications that use application-level concurrency are known as concur... programs. Modern operating systems provide three basic approaches for build... concurrent programs : Processes, I/O multiplexing and threads.
- **Concurrency** means that two or more calculations happen within the same t... frame and there is usually some sort of dependency between them. **Parallel**... means that two or more calculations happen simultaneously.
- But concurrency describes a problem, while parallelism describes a solut... Parallelism is one way to implement concurrency, but it is not the only o... Another popular solution is interleaved processing.

Resources :

- Processes compete for CPU, memory, I/O devices, secondary storage. C... resources may include data items in main memory such as message que... shared data structures.
- Resources can be reusable or consumable, reusable examples are CPU, m... memory. Consumable resource is created by one process and consumed... another example is message sent between one process and another.

3.1.1 Principle of Concurrency

- Concurrent access to shared data may result in data inconsistency. Maintain... data consistency requires mechanisms to ensure the orderly execution... cooperating processes.

- Concurrency arises in the same way at different levels of execution streams. Following are the example of concurrency in different types of operating systems :
  1. Concurrency in multiprogramming : An interaction between multiple processes running on one CPU
  2. Concurrency in multithreading : An interaction between multiple threads running in one process.
  3. Concurrency in multiprocessors : An interaction between multiple CPUs running multiple processes or threads.
  4. Concurrency in multi-computers : An interaction between multiple computers running distributed processes or threads.
- Java is a concurrent programming language.
- Process synchronization is required in uni-processor system, multiprocessor system and network. If more than one thread exists in a system at the same time, then the threads are said to be concurrent. Synchronization problems can occur whenever two or more concurrent processes use any shared resource.
- Cooperating process share the logical address space.

3.1.2 Race Condition

- **Race condition** occurs when two or more operations occur in an undefined manner. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race **condition.**
- There is a "race condition" if the outcome depends on the order of the execution. The outcome depends on the CPU scheduling or "interleaving" of the threads.
- **Race condition** should be avoided because they can cause fine errors in applications and are difficult to debug.
- In banking system **balance** is a shared variable. After depositing the amount, it update by bank employee **balance = balance + amount.** At the same time, some amount is transfer then it becomes **balance = balance – amount.** These two tasks are in a race to write variable **balance.** In this the process that updates last determines the final value of balance.
- The concurrent execution of two processes is not guaranteed to be determinate, since different executions of the same program on the same data may not produce the same result.

Operating system concerns

- Design and management issue for concurrency are as follows :

1. Track of various processes is kept by operating system.
2. Operating systems allocates and deallocate software and hardware resources active process.
3. Operating system must protect user data and physical resources un-authorized process.
4. Process execution speed do not depends upon other process execution speed.

3.1.3 Critical Section Problem

- A critical section is a block of code that only one process at a time can execute when one process is in its critical section, no other process may be in its section. The critical section problem is to ensure that **only one process at a ti allowed** to be operating in its critical section.
- Critical section means, process may change some common variable, writing updating memory location, updating a process table etc. When process is acce shared modifiable data, it is said to be in a critical section.
- Each process takes permission from operating system to enter into the c section. Structure of process is as follows:
  1. Entry section
  2. Remainder section
  3. Exit section
- **Entry section :** It is block of code executed in preparation for entering c section.
- **Exit section :** The code executed upon leaving the critical section.
- **Remainder section :** Rest of the code is remainder section.

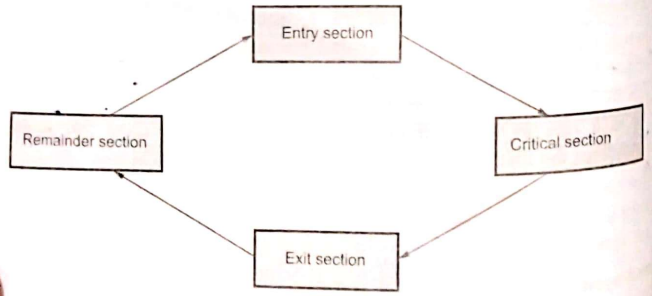


Fig. 3.1.1

- Each process cycles through remainder, entry, critical, exit sections in this order.
- Consider the following example :

  Int count = 5;

Process 1	Process 2
Count = count +1;	count = count -1;
Output : count = ?	(6 or 5)

- **Problem :** Design a protocol to solve this problem.
- **General framework :** Every solution will have the following layout :

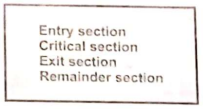


Fig. 3.1.2

- To prevent critical section problem, the system should ensure that only one process at a time can execute the instruction in its critical section for a particular resource. If one process attempts to enter its critical section while another is in its own critical section, the first process should wait until the executing process exits the critical section.
- If a process in a critical section terminates by any reasons, then the operating system must release mutual exclusion so that other process may enter their critical sections.

Solution of Critical Section

- Solution to critical section problem must satisfy the mutual exclusion, progress and bounded waiting parameters.
- Software and hardware solution to critical section problem is available.
- Process using Kernel is active in the system at any given time. Race condition is with implementing a Kernel code. Kernel data structure maintains a list of all open files in the system. System always modifies this list depending upon the process open files and closed files.
- Operating system handles critical section problem by using Kernel. Kernel is classified as **preemptive Kernel and non-preemptive Kernel.**
- **Preemptive Kernel :** If process running in kernel mode then also it is preempted.
- **Non-preemptive Kernel :** It does not allow process to preempt.

Requirements of mutual exclusion

1. At any time, only one process is allowed to enter in its critical section.
2. Solution is implemented purely in software on a machine.
3. A process remains inside its critical section for a bounded time only.
4. No assumption can be made about relative speeds of asynchronous conc processes.
5. A process cannot prevent any other process for entering into critical section.
6. A process must not be indefinitely postponed from entering its critical section.

3.1.4 Critical Region

- Writing set of data into a section of memory, or reading it out, generall some time and is performed through execution of a series of primitives opera
- Critical region is an area of a process which is sensitive to inter-p complications. To guarantee mutual exclusion only one process is allowed to its critical region at a time.
- OS need a system to ensure process cannot enter critical region if another pr is in it's.
- A process must be allowed to finish work on a critical part of the program other processes can have access to it. It is called a critical region because it critical section and its execution must be handled as a unit. Other processes wait before accessing critical region resources.
- Each conditional critical region is a "resource" that has a name and a s variables that can only be accessed in that region. All variables can on accessed within a **conditional critical region.**

3.1.5 Mutual Exclusion

- The need for mutual exclusion comes with concurrency. There are several kin concurrent execution :
  1. Interrupt handlers
  2. Interleaved preemptively scheduled processes/threads
  3. Multiprocessor clusters, with shared memory
  4. Distributed systems.
- Mutual exclusion methods are used in concurrent programming to avoid simultaneous use of a common resource, such as a global variable, by piece computer code called critical sections

- Requirement of mutual exclusion is that, when process P1 is accessing a shared resource R1 then on other process should be able to access resource R1 until process P1 has finished its operation with resource R1.
- Examples of such resources include files, I/O devices such as printers and shared data structures.
- Approaches to implementing mutual exclusion:
  1. Software method : Leave the responsibility with the processes themselves. These methods are usually highly error-prone and carry high overheads.
  2. Hardware method : Special-purpose machine instructions are used for accessing shared resources. This method is faster but cannot provide complete solution. Hardware solution cannot give guarantee of the absence of deadlock and starvation
  3. Programming language method : Provide support through the operating system or through the programming language.

Requirements of mutual exclusion

1. At any time, only one process is allowed to enter in its critical section.
2. Solution is implemented purely in software on a machine.
3. A process remains inside its critical section for a bounded time only.
4. No assumption can be made about relative speeds of asynchronous concurrent processes.
5. A process cannot prevent any other process for entering into critical section.
6. A process must not be indefinitely postponed from entering its critical section.

University Questions

1. *Define : Race condition, mutual exclusion, throughput.* **GTU : Winter-13, Marks 4**
2. *Define and explain following term :*
   Mutual exclusion **GTU : Summer-13, Marks 2**

3.2 Hardware Solution

3.2.1 Test and Set Operations

- Test and set is a single individual machine instruction. It is simply known as TS. It was introduced by IBM for its multiprocessing system.
- In a one machine cycle, it tests to see if the key is available and, if it is, sets it to unavailable. TS are a hardware solution for critical section problem.

- TS instruction eliminates the possibility of preemption occurring, during interval. The instruction :

```
testAndset ( p, q)
```

- The instruction reads the value of q, which may be either true or false. The value is copied into "p" and the instruction sets the value of "q" to true.
- A process P1 would test the condition code using TS instruction before entering critical section. If no other process was in this critical section, then process would be allowed to proceed and the condition code would be changed from zero to one.
- When process P1 exist the critical section, the condition code is reset to zero another process can enter into critical section.
- If process P1 finds a busy condition code then it is placed in a waiting loop where it continues to test the condition code and waits until it is free.
- The "TestandSet" instruction is a tool that programmers use to simplify software solution to mutual exclusion but the instruction itself does not enforce mutual exclusion.

Advantages :

1. Simple to implement.
2. It works well for a small number of processes.
3. It can be used to support multiple critical sections.

Drawbacks :

1. It suffers from starvation.
2. There is possibility of busy waiting.
3. There may be deadlock.

3.3 Strict Alternation

- Strict alternation is used to solve mutual exclusion problem. It is used for two process solution.
- In this method say, there are two processes Process P0 and Process P1 and we use an integer variable flag.
- Code for Process P0 can be written as:

```
while (TRUE)
{
    while (flag != 0)
    .....
```



```
    .....
    criticalsection( );
    .....
    .....
    flag = 1 ;
    non_criticalsection( );
    .....
    .....
}
```

- Code for Process P1:

```
while (TRUE)
  {
    while (flag != 1)
    .....
    .....
    criticalsection( );
    .....
    .....
    flag = 0 ;
    non_criticalsection( );
.....
.....
}
```

- In the above code, the integer variable flag is initially set to 0. Process P0 check the flag value. If flag value is 0 then process P0 enters its critical section and process P1 is waiting at this time.
- Process P1 is in busy waiting and it continuously checks the value of flag. When process P0 finish its critical section execution, it come out of critical section and set flag value equal to 1.
- When flag value becomes 1, then process P1 enters into critical section. By this method mutual exclusion is achieved as each process can go from its non critical region to the critical region by checking the value of flag.
- A few drawbacks with this solution :
  1. It requires to know in advance how many processes will want to share the memory.
  2. A process may be held up indefinitely waiting for its turn, even if no other process is in its critical section.
  3. It increases the CPU overhead

3.4 Lock

- Lock is a shared variable. Software mutual exclusion required that a proce a variable to determine that no other process is executing a critical section. set a variable known as lock. It indicates that the process is executing its c section.
- Initially lock variable is initialized with zero. The process set it to 1 and en critical section.

  Lock = 0 no process in the critical section

  Lock = 1 process in the critical section
- Using simple lock variable, process synchronization problem is not solve avoid this, spinlock is used. A lock that uses busy waiting is called a spin loc

3.5 Mutex

- Mutex is used to ensure only one thread at a time can access the res protected by the mutex. The process that locks the mutex must be the one unlocks it. Mutex is good only for managing mutual exclusion to some s resource.
- Mutex is easy and efficient for implement.
- Mutex can be in one of two states : locked or unlocked
- Mutex is represented by one bit. Zero (0) means unlock and any other represents locked. It uses two procedures.
- When a process needs access to a critical section, it checks the conditio mutex_locks. If the mutex is currently unlocked then calling process enters critical section.
- If the mutex is locked, the calling process is enters into blocked state and until the process in the critical section finishes its execution.
- Mutex variables have only two states so they are simple to implement. Ther is limited to guarding entries to critical regions.
- A segment of code in which a thread may be accessing some shared variab called a **critical region.**
- Mutex variable is like a binary semaphore. But both are not same.

3.6 Peterson's Solution

GTU : Summer-

- Peterson's solution is software based solution for critical section problem. T algorithm will not work correctly on the modern computer architecture. It simpler algorithm for two process mutual exclusion with busy waiting.

- This algorithm is less complicated than Dekker's algorithm.
- Peterson's algorithm solves critical section problem of two processes only. It does not require any special hardware. Two processes alternates execution between their critical section and remainder sections.
- Deadlock and indefinite postponement are impossible in Peterson's algorithm as long as no process or thread terminates unexpectedly.
- For indefinite postponement to occur, one process would have to be able to continually complete and reenter its critical section while the other process busy waited.

University Question

1. *Discuss the Peterson's solution for the race condition with algorithm.*

GTU : Summer-13, Marks 7

3.7 Producer - Consumer Problem

GTU : Summer-14

- Producer - consumer problem is example classic problems of synchronization. Producer process produce data item that consumer process consumes later.
- Buffer is used between producer and consumer. Buffer size may be fixed or variable. The producer portion of the application generates data and stores it in a buffer, and the consumer reads data from the buffer. Fig. 3.7.1 shows producer-consumer problem.

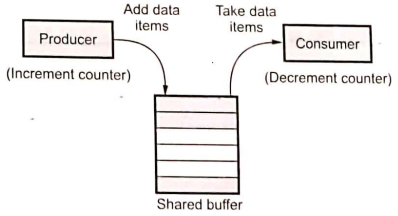


Fig. 3.7.1 Producer - consumer problem

- Producer-consumer is also called bounded buffer problem. There exist several producers and consumers.
- The producer-consumer problem shows the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process produces data items and deposits it in the buffer, while the other process consumes data items from the buffer.

- Producer-consumer problem is solved by using semaphore, mutex and monitor. Mutual exclusion must be enforced on the buffer itself.
- Each data item produce by producer to the shared buffer must be consumed exactly once by the consumer. Data item must be consumed in the same order (First in First Out Order) in which it is out into the buffer.
- If there is no synchronization between producer and consumer then data can be lost if the producer add new data item into the buffer before the consumer consumes the previous data item.
- We use variable counter to keep track of the number of data items in the buffer.
- Solution to single producer and single consumer is given below :

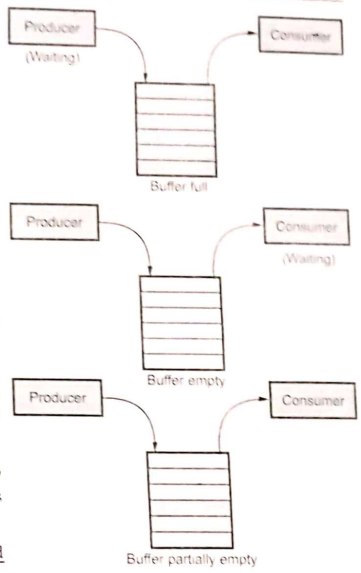


Fig. 3.7.2 Buffer states

1. Code for producer

```
producer (void)
  {
      int item;
          while (TRUE)
            {
                produce item(&item);
                 produce_if (counter  == N)
                  sleep( );
                  enter_item(item);
                   counter = counter + 1;
                       if (counter == 1)
                         wakeup(consumer);
            }
  }
```

2. Code for consumer process

```
consumer(void)
{
    int item;
    while (TRUE)
        {
            if (count == 0)
            sleep( );
            remove_item(&item);
            counter = counter - 1;
    if (count == N-1)
     wakeup(producer);
      consume_item(item);
          }
}
```

- Initially we assume that the buffer is empty. Consumer read the counter and context switch occurs. The producer process produce data item and increments the counter by one. After incrementing the counter, it sends a wakeup signal to consumer. Here one more context switching occurs and consumer process runs.
- Solution to producer consumer problem by using binary semaphore :

```
void main ( )
{
   count = 0;
}
 int t;
binarysemaphore s = 1;
int producer ( )
{
  While (true)
{
     produce_data_item ( );
     semwaitB(s);
     append ( );
     count = count + 1;
     if ( count == 1)
     semSignalB(delay);
     semSignalB(s);
     }
}
int consumer ( )
{
   int p;
semWaitB(delay);
while (true)
{
```



```
  semWaitB(s);
consume( );
count = count - 1;
p = count;
semSignalB(s);
consumedata( );
if ( p == 0)
semWaitB(delay);
}
}
```

University Question

1. *Explain the Problem of Critical Section (CSP) through producer consumer problem. Explain any one solution in detail.* **Summer-14, Marks 7**

3.8 Semaphore

GTU : Winter-13, Summer-13,14

- Semaphore is described by Dijkstra. Semaphore is a nonnegative integer variable that is used as a flag. Semaphore is an operating system abstract data type. It takes only integer value. Its used to solve critical section problem.
- Dijkstra introduced two operations (P and V) to operate on semaphore to solve process synchronization problem. A process calls the P operation when it wants to enter its critical section and calls V operation when it wants to exit its' critical section. The P operation is called as wait operation and V operation is called as signal operation.
- A wait operation on a semaphore decease its value by one.

```
waits : while S < 0
        do loops;
      S:= S - 1;
```

- A signal operation increments its value:

```
signal:
  S:= S + 1;
```

- A proper semaphore implementation requires that P and V be indivisible operations. A semaphore operation is atomic. This may be possible by taking hardware support. The operation P and V are executed by the operating system in response to calls issued by any one process naming a semaphore as parameter.
- There is no guarantee that no two processes can execute wait and signal operations on the same semaphore at the same time.

3.8.1 Binary Semaphore

- Binary semaphore is also known as mutex locks. It deals with the critical s... for multiple processes.
- Binary semaphore value is only between 0 and 1.

Counting semaphore

- Used with that resource which has a finite number of instances.
- It works over unrestricted domain.
- Nonbinary semaphore often referred to as either a counting semaphore a... general semaphore.
- Process waiting for semaphore is stored in the queue for binary and co... semaphore. Queue uses first in first out policy.
- A semaphore that does not specify the order in which processes are removed ... the queue is called as **weak semaphore.** The process that has been blocked ... longest is released from the queue first is called a **strong semaphore.**
- Semaphore can be implemented in user applications and in the kernel. ... operations can be implemented in the kernel by blocking waiting process to a... busy waiting.

3.8.2 Busy Waiting

- Busy waiting is a situation in which a process is blocked on a resource but ... not yield the processor. A busy wait keeps the CPU busy in executing a pr... even as the process does nothing.
- Busy waiting is also called as spin waiting.
- Busy waiting waste CPU cycles that some other process might be able to ... productively.
- To avoid busy waiting, a process waiting for critical section, they put into ... blocked state. When process allowed entering into critical section, then proces... kept in ready state.
- To overcome the need for bust waiting :

In wait( ) operation :

a. When process reads semaphore value and it is not positive value then proc... blocked itself for some time.

b. Semaphore keeps all blocked process in the waiting queue so process is kept in ... queue.

c. CPU scheduler selects another process for execution.

In signal ( ) operation :

a. The wakup ( ) operation is used for restating the blocked processes. When any other process executes signal ( ) operation, then blocked process state changed from blocked state to ready state.

b. This process is kept in the ready queue.

Properties of semaphore :

1. Semaphores are machine independent.
2. Semaphores are simple to implement.
3. Correctness is easy to determine.
4. Semaphore acquire many resources simultaneously.
5. Can have many different critical sections with different semaphores.

3.8.3 Drawbacks of Semaphore

1. They are essentially shared global variables.
2. Access to semaphores can come from anywhere in a program.
3. There is no control or guarantee of proper usage.
4. There is no linguistic connection between the semaphore and the data to which the semaphore controls access.
5. They serve two purposes, mutual exclusion and scheduling constraints.

3.8.4 Semaphore Programming

1. Semget ( )

- To create a semaphore or gain access to one that exists, the *semget* system call is used.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget (key_t key ,intnsems, int semflg);
```

- The semget system call takes three arguments :
  a. The first argument, *key*, is used by the system to generate a unique semaphore identifier.
  b. The second argument, *nsems*, is the number of semaphores in the set.
  c. The third argument, *semflg*, is used to specify access permission and/or special creation conditions.
- If the semget system call fails, it returns a – 1 and sets the value stored in errno.

- When a semaphore is first created, the kernel assigns to it an associated :
  a. Semaphore Control Block (SCB),
  b. A unique ID,
  c. A value (binary or a count), and
  d. A task - waiting list
- A Kernel can support many different types of semaphores, including bin counting, and mutual-exclusion (mutex) semaphores.

2. semctl ( ) function

- The *semctl* system call allows the user to perform a variety of generalized com operations on the system semaphore structure, on the semaphores as a set, and individual semaphores.
- The function prototype is as follows :

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, union semun arg);
```

- The *semctl* system call takes four arguments :

1. The first argument, *semid* is a valid semaphore identifier that was returned b previous *semget* system call.
2. The second argument, *semnum*, is the number of semaphores in the semaphore (array), 0 means this number (index) is not relevant.
3. The third argument to *semctl*, *cmd*, is an integer command value. The *cmd* va directs *semctl* to take one of several control actions. Each action requires spec access permissions to the semaphore control structure.
4. The fourth argument to *semctl*, *arg*, is a union of type *semun*. Given the ac specified by the preceding *cmd* argument, the data in *arg* can be one of any of following four values :
   a. An integer already was set in the val member of sem_union that used w SETVAL to indicate a change of specific value for a particular semaph within the semaphore set.
   b. A reference to a semid_ds structure where information is returned wh IPC_STAT or IPC_SET is specified.
   c. A reference to an array of type unsigned short integers; the array is used eit to initialize the semaphore set or as a return location when specifi GETALL.
   d. A reference to a seminfo structure when IPC_INFO is requested.

- If semctl fails, it returns a value of – 1 and sets errno to indicate the specific er

3. semop ( ) function

- Additional operations on individual semaphores are accomplished by using the *semop* system call. Its syntax is :

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
```

- The semop system call takes three arguments :
  a. The first argument, *semid*, is a valid semaphore identifier that was returned by a previous successful *semget* system call.
  b. The second argument, sops, is a reference to the base address of an array of semaphore operations that will be performed on the semaphore set associated with by the *semid* value.
  c. The third argument, nsops, is the number of elements in the array of semaphore operations.
- If *semop* fails, it returns a value of – 1 and sets errno to indicate the specific error.
- If a *semop* call must block after completing some of its component operations, the kernel *rewinds* the operation to the beginning to ensure atomicity of the entire call.
- When the *sem_op* value is negative, the process specifying the operation is attempting to decrement the semaphore. The decrement of the semaphore is used to record the acquisition of the resource affiliated with the semaphore.
- When a semaphore value is to be modified, the accessing process must have altered permission for the semaphore set.
- When the *sem_op* value is positive, the process is adding to the semaphore value. The addition is used to record the return (release) of the resource affiliated with the semaphore.
- Again, when a semaphore value is to be modified, the accessing process must have alter permission for the semaphore set.
- When the *sem_op* value is zero, the process is testing the semaphore to determine if it is at 0.
- When a semaphore is at 0, the testing process can assume that all the resources affiliated with the semaphore are currently allocated.

4. sem_post ( )

Syntax :

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

- The sem_post( ) function unlocks the specified semaphore by performing a semaphore unlock operation on that semaphore. When this operation results in a positive semaphore value, no threads were blocked waiting for the semaphore to be unlocked; the semaphore value is simply incremented.
- When this operation results in a semaphore value of zero, one of the threads waiting for the semaphore is allowed to return successfully from its invocation of the sem_wait( ) function. The sem_post( ) interface is reentrant with respect to signals and may be invoked from a signal-catching function.

5. sem_init ( )

Syntax :

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- The sem_init( ) function is used to initialize the semaphore's value. The pshared argument must be 0 for semaphores local to a process. The value argument specifies the initial value for the semaphore.
- The pshared argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.
- If pshared has the value 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads.
- If pshared is nonzero, then the semaphore is shared between processes, and should be located in a region of shared memory.

University Questions

1. What is semaphore ? Give the implementation of readers-writers problem using semaphore. **GTU : Winter-13, Marks 7**
2. What is semaphore ? How can we achieve the synchronization using semaphore for producer consumer problem ? **GTU : Summer-13, Marks 7**
3. Write short note : Semaphores **GTU : Summer-14, Marks 7**

3.9 Dekker's Solution

- Dekker's algorithm was the first correct solution to the critical section problem for two processes/threads. It uses an array of Boolean values and an integer variable.
- Dekker's algorithm is purely **software solution with no special purpose hardware instruction.** It uses flag to indicate a process desire to enter its critical section.

- Processes are loops indefinitely, repeating entering and reentering its critical section. Here flag and turn are the two global variables. Variable flag indicates the position of the process with respect to mutual exclusion. The turn variable is used for resolving the conflicts.

```
var flag1, flag2: boolean = false;
turn: 1..2 = 1;
```

P1: **begin**	P2: **begin**
flag1 = true;	flag2 = true;
turn = 2;	turn = 1;
while flag2	**while** flag1
and turn = 2 **do;**	and turn = 1 **do;**
<critical section 1>	<critical section 2>
flag1 = false;	flag2 = false;
... ...	... ...
end P1	**end** P2

- Dekker's algorithm is correct and it satisfies mutual exclusion and it is free from deadlock and starvation.
- The process P1 indicates its desire to enter its critical section by setting its flag to true. The process then proceeds to the while test and determine whether P2 also wants to enter its critical section.
- When process P1 performs the while test and it finds that P2 flag is set to true. So there is a contention between processes attempting to enter their respective critical sections. The process P1 enters the body of its while loop, where the process examines the value of variable **turn** which is used to resolve such conflicts.

Limitation of Dekker's Algorithm :

1. Dekker's algorithm does not provide strict alternation.
2. Dekker's algorithm will not work with many modern CPUs.
3. This algorithm won't work on Symmetric Multi-Processors (SMP) CPUs.
4. Additionally, Dekker's algorithm can fail regardless of platform due to many optimizing compilers. Compiler may remove writes to flag since never accessed in loop. Creating an infinite loop.

Drawback of software solutions

1. There is possibility of busy waiting.
2. Makes difficult assumptions about the memory system.
3. Programming part is complicated.

3.10 Classical IPC Problem

3.10.1 Dinning Philosopher Problem

- Dinning philosophers problem is one of classical process synchronization problem. Here five philosophers are seated around a circular table. They spend their lives in thinking and eating. Five plates with five forks are kept on the circular table.
- While philosophers think, they ignore the eating and do not require fork. When a philosopher decides to eat, then he/she must obtain a two fork, one from left side fork and another from right side fork. Philosophers can pick up only a single fork at one time.

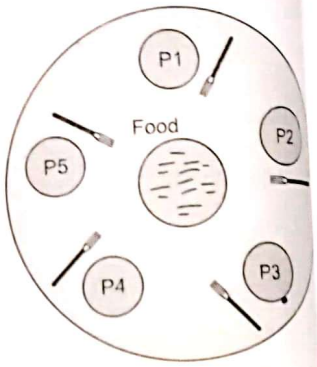


Fig. 3.10.1 Dinning philosophe

- After consuming a food, the philosophers replace the fork and resumes thinking. A philosophers to the left or right dinning philosopher cannot eat while the dinning philosophers is eating, forks are a shared resource.
- Fig. 3.10.1 shows seating arrangement of five philosophers.
- Consider the following code :

```
void dinning_Philosopher ( )
  {
     while (true)
     {
       thinking ( );
       eating ( );
     }

  }
void eating ( )
{
   takeleftfork ( );
```



```
            eatfood(delay);
            putrighttfork ( );
      putleftfork ( );
}
```

- Here philosophers operates asynchronously and concurrently, it is possible for each philosophers to enter into eating mode. The procedure takeleftfork ( ) waits until the specified fork is available and then seizes it. This is not possible because of the following reasons.
- Each philosopher will hold exactly one fork, and no forks will remain available on the table. The philosophers will all deadlock.
- To solve this problem, write some extra code in takerightfork ( ) procedure. User can specify the philosophers to put down the left fork if that philosophers cannot obtain the right fork.

Problem analysis :

- **Shared resource :** Here each fork is shared by two philosophers so we called it is a shared resource.
- **Race condition :** we do not want a philosopher to pick up a fork that has already been picked up by his neighbor. This is a race condition.
- **Solution :** we consider each fork as a shared item and protected by a mutex lock. Before eating, each philosopher first lock left fork and then right fork. If philosopher acquires both locks successful, then philosophers have two locks (two fork) so he can eat a food. After finishes easting, this philosopher releases both fork, and thinks.

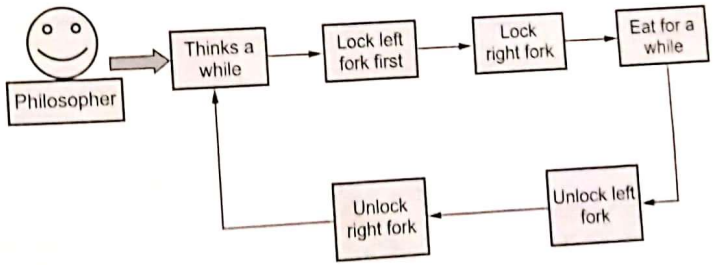


Fig. 3.10.2 Philosopher states

Some other alternatives :

1. Pick up the left fork, if the right fork is not available for a given time, put the fork down, wait and try again. Even if each philosopher waits a different ran time, an unlucky philosopher may starve.

2. Require all philosophers to acquire a binary semaphore before picking up a... forks. This guarantees that no philosopher starves but limits parallel... dramatically.

- Because we need to lock and unlock a chopstick, each chopstick is associated w... a mutex lock. Since we have five philosophers who think and eat simultaneou... we need to create five threads, one for each philosopher. Since each philosop... must have access to the two mutex locks that are associated with its left and r... chopsticks, these mutex locks are global variables.

3.10.2 Reader-Writer Problem

- When two types of processes need to access a shared resource such as a file ... database. They called these processes reader and writers.
- For example in railway reservation system, readers are those who want to... schedule information. They are called reader because readers do not change t... content of the database. Many readers may access the database at once. There ... no need to enforce mutual exclusion among readers.
- The writers are those who making reservations on a particular train. Writer c... modify the database, so it must have exclusive access. When writer is active, n... other readers or writers may be active. Therefore, it must enforce mutual exclusi... if there are groups of readers and a writer.
- Reader writer problem is similar to one which a file is to be shared among a set ... processes. If a process want only to read the file, then it may share the file wi... any other process that also wants to read the file. If the writer wants to appe... the file, then no other process should have access to the file when the writer h... access to it.
- If reader having higher priority than the writer, then there will be starvation wi... writers. For writer having higher priority than reader then starvation with reader...

3.11 Monitors

GTU : Winter-13, Summer-14

- Monitor is an object that contains both data and procedures needed to perfor... allocation of a shared resource. It is a programming language construct th... support both data access synchronization and control synchronization.
- Monitor is implemented in programming languages like Pascal, java and C++. Ja... makes extensive use of monitors to implement mutual exclusion. Fig. 3.11.1 show... structure of monitor.
- Monitor is an abstract data type for which only one process may be executing ... procedure at any given time. Monitor is a collection of procedure, variables a...

data structure. Data inside the monitor may be either global to all routines within the monitor or local to a specific routine.

- Monitor data is accessible only within the monitor. A shared data structure can be protected by placing it in a monitor. If the data in a monitor represents some resource, then the monitor provides a mutual exclusion facility for accessing the resource. A procedure defined within a monitor can access only those variable declared locally within the monitor and its formal parameters.
- When a process calls a monitor procedure, the first few instruction of the procedure will check to see if any other process is currently active within the monitor. If process is active then calling process will be suspended until the other process has left the monitor. If no other process is using the monitor, the calling process may enter.

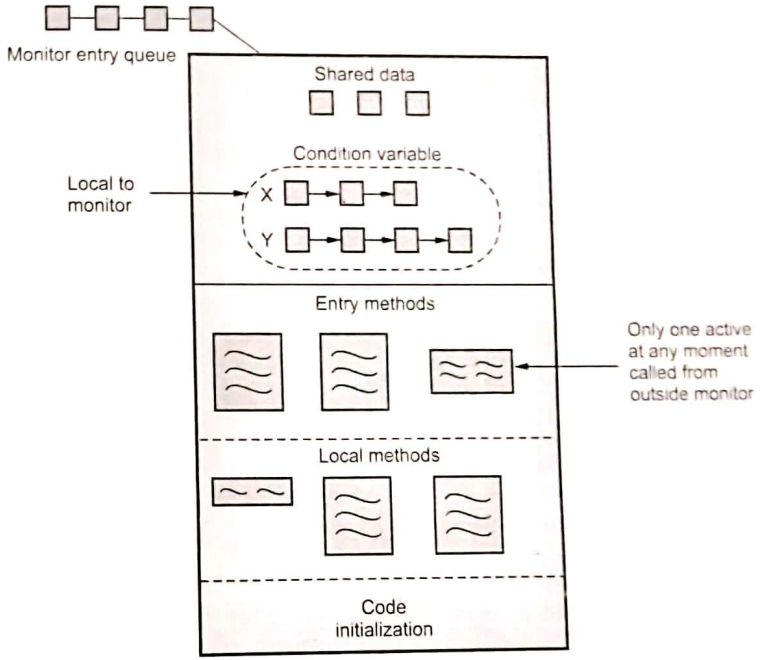


Fig. 3.11.1 Monitor structure

- Monitor supports synchronization by the use of condition variables that are contained within the monitor and accessible only within the monitor. Every conditional variable has an associated queue. Condition variables operates on two functions :

cwait(condition variable)

csignal (condition variable)

- cwait : It suspend execution of the calling process on condition.
- csignal : Resume execution of some process blocked after a cwait on the condition.
- The cwait must come before the csignal.
- Fig. 3.11.2 shows monitor structure with condition variables.

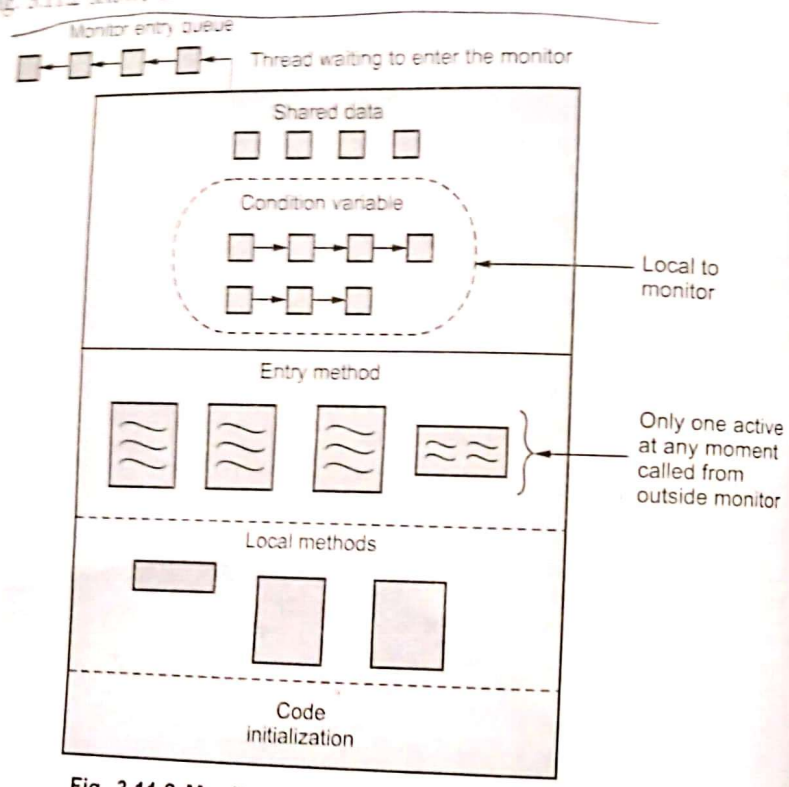


Fig. 3.11.2 Monitor structure with condition variable

- CPU is a resource that must be shared by all processes. The part of the k that apportions CPU time between processes is called the scheduler.
- A condition variable is like a semaphore, with two differences :
  1. A semaphore counts the number of excess up operations, but a s operation on a condition variable has no effect unless some process is wai A wait on a condition varibale always blocks the calling process.
  2. A wait on a condition variable automatically does an up on the monitor m and blocks the caller.

Example : Solve producer consumer problem by using monitors

Bounded Buffer problem using monitors UoP : Dec.-22

```
monitor Bounded Buffer
{
        private Buffer b = new Buffer (20);
        private int count = 0;
        private condition nonfull, nonempty;
        public void add (object item)
}
        if(count == 20)
        nonfull.wait ( );
        b.add (item);
        count ++;
        nonempty. signal ( );
}
public object remove ( )
{
        if (count == 0)
        nonempty.wait ( );
        item result = b.remove( );
        count = count-1;
        nonfull.signal( );
        return result ;
}
```

- Each condition variable is associated with some logical condition on the state o the monitor. Consider what happens when a consumer is blocked on the nonempty condition variable and producer calls add.
  1. The producer adds the item to the buffer and calls nonempty signal ( ).
  2. The producer is immediately blocked and the consumer is allowed to continue.
  3. The consumer removes the item from the buffer and leaves the monitor.
  4. The producer wakes up and since the signal operation wait the last statemen in add, leaves the monitor.
- Monitors are a higher level concept than P and V. They are easier and safer to use but less flexible. Many languages are not supported by the monitor. Java is making monitors much more popular and well known.

Example 3.11.1 *Solve the reader-writer problem using monitor with readers priority.* UoP : Dec.-01

Solution :

```
readers-writers : monitor;
begin
        integer readercount;
        condition okread, okwrite;
        boolean busy;
```

```
procedure startread;
        begin
                if busy then ok read .wait;
                readercount := readercount+1;
                okread signal;
        end startread;
        procedure endread;
        begin
                reader count := readercount-1;
        if      reader count = 0 then okwrite.signal;
                end endread;

procedure startwrite;
        begin if busy OR readercount ≠ 0 then okwrite.wait;
        busy := true;
        end startwrite;

procedure endwrite;
        begin
        busy := false;
        if okread queue then okread signal
        else okwrite signal
        end endwrite;
begin (*Initialisation*)
readercount :=0;
busy :=false;
end;
end readers-writers;
```

Example 3.11.2 *Solve procedure-consumer problem with monitors.*

Solution :

```
monitor procedure-consumer
condition full empty;
integer count;

Procedure insert (item : integer);
        begin
                if count = N then wait (full);
                insert-item(item);
                count := count + 1;
                if count = 1 then signal (empty)
        end;
function remove integer
        begin
                if count = 0 then wait (empty);
                remove = remove_item;
                count := count-1;
                if count = N-1 then signal (full);
```



```
                end;
                count := 0;
        end monitor;
        procedure producer;
                begin
                while true do
           begin
                item = produce_item;
                        producer_consumer.insert(item)
                end
    end;
                procedure consumer;
                        begin
                        while true do
                                begin
                                item = procedure_consumer.remove;
                                consume_item(item)
                        end
                end;
```

- By making the mutual exclusion of critical regions automatic, monitors make parallel programming much less error prone than with semaphores.

3.11.1 Drawbacks of Monitors

1. Major weakness of monitors is the absence of concurrency if a monitor encapsulates the resource, since only one process can be active within a monitor at a time.
2. There is the possibility of deadlocks in the case of nested monitors calls.
3. Monitor concept is its lack of implementation most commonly used programming languages.
4. Monitors cannot easily be added if they are not natively supported by the language.

3.11.2 Difference between Monitors and Semaphore

- A semaphore is an operating system abstract data type whereas monitors are based on abstract data types.
- Semaphore-level synchronization primitives are difficult to use for complex synchronization situations. Monitors were derived to simplify the complexity of synchronization problems by abstracting away details.
- Semaphores provide a general-purpose mechanism for controlling access to critical sections. Their use does not guarantee that access will be mutually exclusive or

deadlock will be avoided. A monitor is a programming language construct that provides appropriate access to critical sections.

- Monitor uses condition variables. A condition variable is like a semaphore but is different.
  1. A wait operation causes the number of excess up operations, but a wait operation on a condition variable has no effect unless some process is waiting. A wait on a condition variable always blocks the calling process and blocks the caller.
  2. A wait on a condition variable automatically does an up on the monitor lock

University Questions

- What is monitor ? Give the implementation of bounded buffer producer consumer problem using monitor — GTU : Winter-13, 7 Marks
- Write short note : Monitors — GTU : Summer-14, 7 Marks

3.12 Inter Process Communication Models

- Exchange of data between two or more separate, independent processes/threads is possible using IPC. Operating systems provide facilities/resources for Inter-Process Communications (IPC), such as message queues, semaphores, and shared memory.
- A complex programming environment often uses multiple cooperating processes to perform related operations. These processes must communicate with each other and share resources and information. The Kernel must provide mechanisms to make this possible. These mechanisms are collectively referred to as **interprocess communication.**
- Distributed computing systems make use of these facilities/resources to provide Application Programming Interface (API) which allows IPC to be programmed at a higher level of abstraction. (e.g., send and receive).
- Five types of inter-process communication are as follows :
  1. Shared memory permits processes to communicate by simply reading and writing to a specified memory location.
  2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
  3. Pipes permit sequential communication from one process to a related process.
  4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.

5. Sockets support communication between unrelated processes even on different computers.

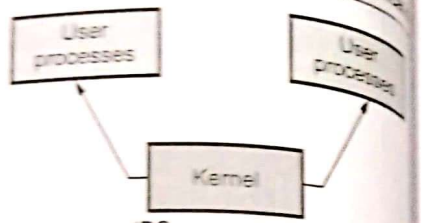
Name	Description	Scope	Use
File	• Data is written to and read from a typical UNIX file. • Any number of processes can interoperate.	Local	Sharing large data sets.
Pipe	• Data is transferred between two processes using dedicated file descriptors. • Communication occurs only between a parent and child process.	Local	Simple data sharing, such as producer and consumer.
Named pipe	• Data is exchanged between processes via dedicated file descriptors. • Communication can occur between any two peer processes on the same host.	Local	Producer and consumer or command-and-control as demonstrated with MySQL server and its command-line query utility.
Signal	• An interrupt alerts the application to a specific condition.	Local	Cannot transfer data in a signal, so mostly useful for process management.
Shared memory	• Information is shared by reading and writing from a common segment of memory.	Local	Cooperative work of any kind, especially if security is required.
Socket	• After special setup, data is transferred using common input/output operations.	Local or remote	Network services such as FTP, ssh, and the Apache web server.

Table 3.12.1 Interprocess communication in UNIX

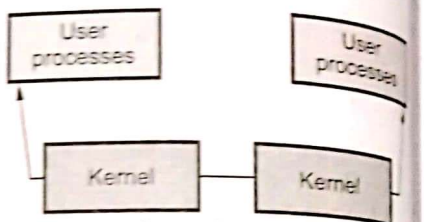
- Purposes of IPC
  1. **Data transfer :** One process may wish to send data to another process.
  2. **Sharing data :** Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it.
  3. **Event modification :** A process may wish to notify another process or set of processes that some event has occurred.
  4. **Resource sharing :** The Kernel provides default semantics for resource allocation; they are not suitable for all application.

5. **Process Control** : A process such as a debugger may wish to assume complete control over the execution of another process.

- IPC has two forms : IPC on same host and IPC on different hosts



IPC on same host



IPC on different hosts

Fig. 3.12.1 IPC on different hosts

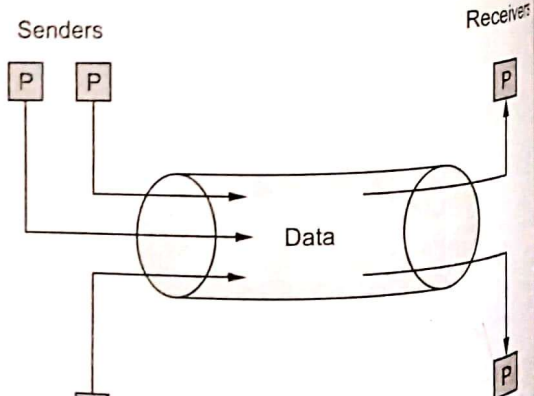
IPC is used for 2 functions :

1. **Synchronization** : Used to coordinate access to resources among processes and also to coordinate the execution of these processes. They are record locking, semaphores, mutexes and condition variables.

2. **Message passing** : Used when processes wish to exchange information. Message passing takes several forms such as : Pipes, FIFOs, message queues and shared memory.

3.12.1 Pipes

- A pipe is a unidirectional, first-in first-out, unstructured data stream of fi maximum size. Writers add data to the end of the pipe; readers retrieve data fr the front of the pipe.
- Once read, the data is removed from the pipe and is unavailable to other read A pipe provides a simple flow control mechanism.
- A process attempting to read from empty pipe blocks until more data is writte the pipe. A process trying to write to a full pipe lock until another process re data from pipe.
- The pipe system call creates a pipe and returns two file descriptors: one for reading and one for writing. These descriptors are inherited by child processes, which thus share access to the file.



Each pipe can have several readers and writers. A give process may be a reader or writer

- Pipes can be used only between processes that have a common ancestor. Normally, a pipe is created by a process, that process calls fork and the pipe is used between the parent and the child.
- Example to show how to create and use a pipe :

```
main( )
{
   int pipefd[2], n;
   char buff[100];

   if (pipe(pipefd) < 0 )
  err_sys("pipe error");
  printf("read fd = %d, write fd = %d\n", pipefd[0], pipefd[1]);

 if (write(pipefd[1], "hello world\n", 12) != 12)
  err_sys("write error");
 if ( (n=read(pipefd[0], buff, sizeof(buff))) < =0)
  err_sys("read error");
 write (1, buff, n); /*fd=1=stdout*/
 }
```

Properties of pipe :

1. Pipes do not have a name. For this reason, the processes must share a parent process. This is the main drawback to pipes. However, pipes are treated as file descriptors, so the pipes remain open even after fork and exec.
2. Pipes do not distinguish between messages; they just read a fixed number of bytes.
3. Pipes can also be used to get the output of a command or to provide input to a command

- The most common use of pipes is to let the output of one program become the input for another. Users typically join two programs by a pipe using the shell's pipe operator (|).

Limitations of pipes :

1. Reading data removes it from the pipe, a pipe cannot be used to broadcast data to multiple receivers.
2. Data in a pipe is treated as a byte stream and has no knowledge of message boundaries.
3. If there are multiple readers on a pipe, a writer cannot direct data to a specific reader.

- **Program for sending data from parent process to child process over a pipe**

```
#include <stdio.h>
```

```
{
int n;
int fd[2];
pid_t pid;
char line[MAXLINE];

if (pipe(fd) < 0)
err_sys("pipe error");

if ((pid = fork( )) < 0)
{
  err_sys("fork error");

} else if (pid > 0)
 { /* parent */
  close(fd[0]);
 write(fd[1], "hello world\n", 12);

 } else { /* child */
  close(fd[1]);

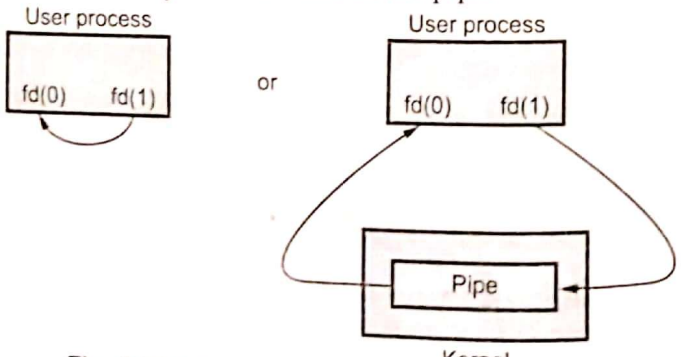
n = read(fd[0], line, MAXLINE);
write(STDOUT_FILENO, line, n);

}
exit(0);
}
```

- Pipe is created by calling the pipe function.

```
#include < unistd.h>
int pipe ( int filedes[2]);
```

- Two file descriptor are returned through the *filedes* argument : *filedes[0]* is op reading and *filedes[1]* is open for writing. The output of *filedes[1]* is the inp *filedes[0].*
- Fig. 3.12.3 shows two ways to view the UNIX pipe.



- The *fstat* function returns a file type of FIFO for the file descriptor of either end of a pipe. A pipe in a single process is not useful. Normally, the process that calls pipe then calls fork, creating an IPC channel from the parent to the child or vice versa. Fig. 3.12.4 shows half-duplex pipe after a fork.

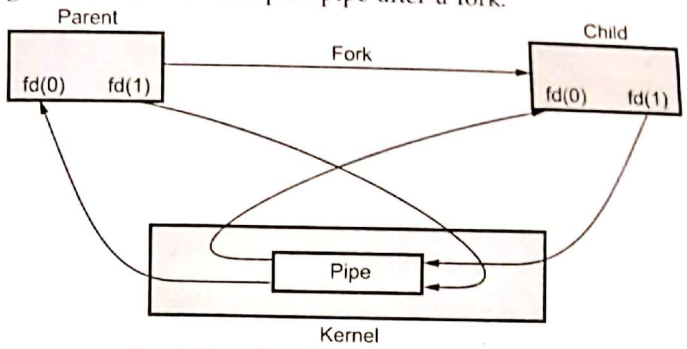


Fig. 3.12.4 Half-duplex pipe after a fork

- Fig. 3.12.5 shows pipe from parent to child. For a pipe from the parent to th child, the parent closes the read end of the pipe (fd[0]), and the child closes th write end (fd[1]).

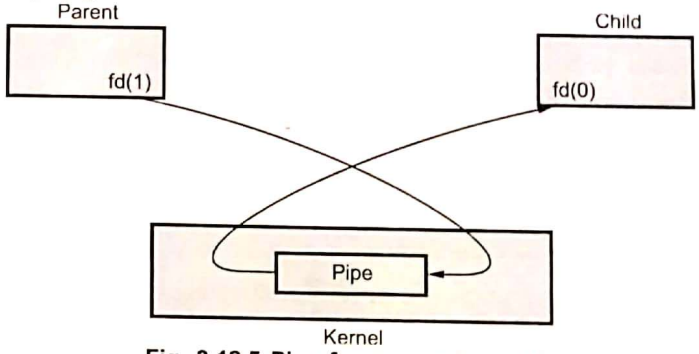


Fig. 3.12.5 Pipe from parent to child

- For a pipe from the child to the parent, the parent closes fd[1], and the chi closes fd[0]. When one end of a pipe is closed, the following two rules apply.
  1. If we read from a pipe whose write end has been closed, read returns 0 t indicate an end of file after all the data has been read.
  2. If we write to a pipe whose read end has been closed, the signal SIGPIPE i generated. If we either ignore the signal or catch it and return from the signal handler, write returns 1 with errno set to EPIPE.

a) Send (destination_name, message)

b) Receive (source_name, message).

- Send primitive is used for sending a message to destination. Process sends information in the form of a message to another process designated by a destination. A process receives information by executing the receive primitive, which indicates the source of the sending process and the message.
- Design characteristics of message system for IPC
  1. Synchronization between the process
  2. Addressing
  3. Format of the message
  4. Queueing discipline

Issues in IPC by Message Passing

- Message is a block of information.
- A message is a meaningful formatted block of information sent by the sender process to the receiver process.
- The message block consists of a fixed length header followed by a variable size collection of typed data objects.
- The header block of a message may have the following elements :
  1. **Address** : A set of characters that uniquely identify both the sender and receiver.
  2. **Sequence Number** : It is the message identifier to identify duplicate and lost messages in case of system failures.
  3. **Structural Information** : It has two parts. The type part that specifies whether the data to be sent to the receiver is included within the message or the message only contains a pointer to the data. The second part specifies length of the variable-size message.
- Fig. 3.12.6 shows the typical message format.

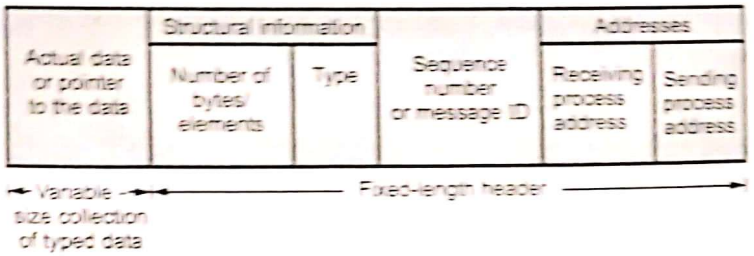


Fig. 3.12.6 Message structure

- Some important issues to be considered for the design of an IPC protocol b message passing system :
  i. The sender's identity
  ii. The receiver's identity
  iii. Number of receivers
  iv. Guaranteed acceptance of sent messages by the receiver
  v. Acknowledgment by the sender
  vi. Handling system crashes or link failures
  vii. Handling of buffers
  viii. Order of delivery of messages

3.12.4 IPC Synchronization

- Send operation can be synchronous or asynchronous. Receive operation can blocking or nonblocking.
- Sender and receiver process can be blocking mode or nonblocking mode. Differ possibility of sender and receivers are as follows:
  1. Blocking send, blocking receive
  2. Nonblocking send, blocking receive
  3. Nonblocking send, Nonblocking receive

Blocking send, blocking receive

- Blocking send must wait for the receiver to receive the message. Synchron communication is an example of blocking send. Both processes (sender receiver) are blocked until the message is delivered.
- **Rendezvous** : sending a message to another process leaving the sender pro suspended until the message is received and processed.

Nonblocking send, blocking receive

- Sender is free to send the messages but receiver is blocked until the reque message arrives.
- Asynchronous communication is an example of nonblocking send. Asynchron communication with nonblocking sends increases throughput by reducing the that processes spend waiting.
- Natural concurrent

3.12.5 Shared Memory

- A region of memory that is shared by co-operating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer. Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of Kernel intervention.
- In contrast, in shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the Kernel is required.
- Fig. 3.12.7 shows client/server with shared memory.

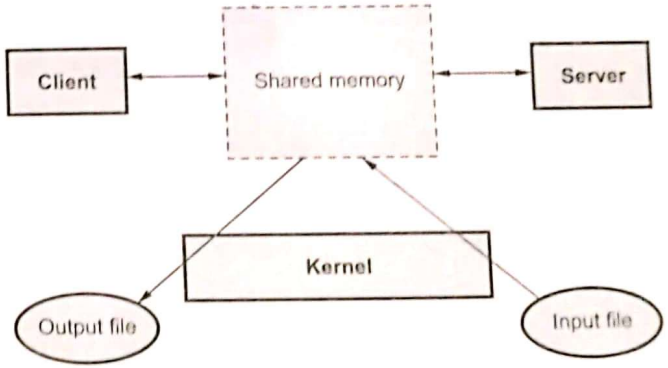


Fig. 3.12.7 Client / server with shared memory

Advantages

1. Good for sharing large amount of data.
2. Very fast.

Limitations

1. No synchronization provided - applications must create their own.
2. Alternative to shared memory is *mmap* system call, which maps file into th address space of the caller.

❑❑

4 Deadlock

Syllabus

Definition, Deadlock characteristics, Deadlock prevention, Deadlock avoidance : Banker's algorithm, Deadlock detection and recovery

Contents

4.1 *Introduction to Deadlock* ..... **Winter-13, Summer-13,14,** ..... Marks 7
4.2 *Deadlock Characteristics*
4.3 *Deadlock Solution*
4.4 *Deadlock Prevention*
4.5 *Deadlock Avoidance* ..... **Winter-13, Summer-13, 14** ..... Marks 7
4.6 *Deadlock Detection*
4.7 *Deadlock Recovery*
4.8 *Comparison between Detection, Prevention and Avoidance Methods of Handling Deadlock*

4.1 Introduction to Deadlock

GTU : Winter-13, Summer-14

- A lack of process synchronization can result in two extreme conditions deadlock or starvation. Deadlock is the problem of multiprogrammed system.
- Deadlock can be defined as the permanent blocking of a set of processes either complete for system resources. Deadlock is more serious than starvation
- Deadlock can occur on sharable resources such as files, printers, database, tape drives, memory, CPU cycles etc.
- A process is in deadlock state if it was waiting for a particular event that will occur. In a system deadlock, one or more processes are deadlocked.

Deadlock Example

- Computer system is collection of limited/finite number of resources. resources are distributed among a number of competing processes.
  Resources are of two types :
  1. Reusable resources
  2. Consumable resources
- **Reusable resource** is used only by one process at a time. Process can resource after use. Processors, I/O channel, I/O device, files, database, and secondary memory, semaphore are example of the reusable resource.
- Consumable resource is one that can be created and destroyed. There is on the number of consumable resources of a particular type.
- An interrupt, messages, signals and messages in I/O buffers are examp consumable resources.
- Process utilize the resources in the following sequence :
  1. Request for resource
  2. Use of resource
  3. Resource release
- Process uses system call for requesting the resource. After allocating reso the process, it use or operate the resource.
- The process releases the resource after use.
- If the resource is not available when it is requested, the requesting pro forced to wait.

Process 1	Process 2	Process 3	Process 4
Printf(" ")	int a;	Procedure ( )	sort ( )
- - - - -	- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -	- - - - -
request (resource 1); /* Holding res 1 */	request (resource 2); /* Holding res 2 */	request (resource 3); /* Holding res 3 */	request (resource 4) /* Holding res 4*/
- - - - -	- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -	- - - - -
request (resource 2);	request (resource 3);	request (resource 4);	request(resource 1);

Fig. 4.1.1

- Fig. 4.1.1 shows the deadlock with 4 process.
- Process 1 is holding resource 1 and requesting resource 2; process 2 is holdi resource 2 and requesting resource 3; process 3 is holding resource 3 a requesting resource 4. Process 1 is holding resource 4 also.
- Here deadlock occurs because none of the processes can proceed because all a waiting for a resource held by another blocked process.
- To break this situation, one of the process release the resource.
- Operating system must be handle the deadlock situation. OS detect the deadl and solve the deadlock problem.

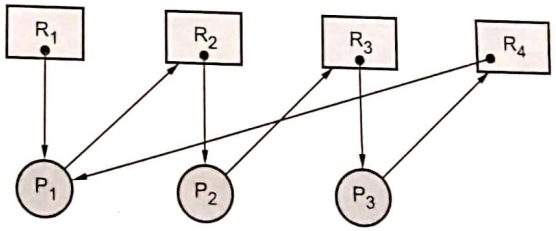


Fig. 4.1.2 Deadlocked processes

4.1.1 Resource Allocation Graphs

- Resource allocation graph is introduced by Holt. It is a directed graph that de a state of the system of resources and processes.
- Process and resource are represented by node in directed graph. Graph consists of a set of vertices (V) and set of edges (E).



Fig. 4.1.3 Pro

- A process node is graphically represented by circle.
- A resource node is graphically represented by a rectangle and represents one of resources.

- The number of bullet symbols in a resource node indicates how many units of that resource class exist in the system.



Fig. 4.1.4 Resource with 8 instances

- An arrow from the process to resource indicates the process is requesting the resource. An arrow from resource process shows an instance of the resource has been allocated to the process.
- Claim edge $P \rightarrow R$ indicated that process P may request resource R in the fu represented by a dashed line. Claim edge converts to request edge when a p requests a resource.
- Request edge converted to an assignment edge when the resource is allocat the process. When a resource is released by a process, assignment edge reconv to a claim edge.



(a) Request edge (b) Claim edge

Fig. 4.1.5 Edges

- Fig. 4.1.6 shows a resource allocation graph. System consists of proces resources.

Process : P_1, P_2

Resource : R_1, R_2

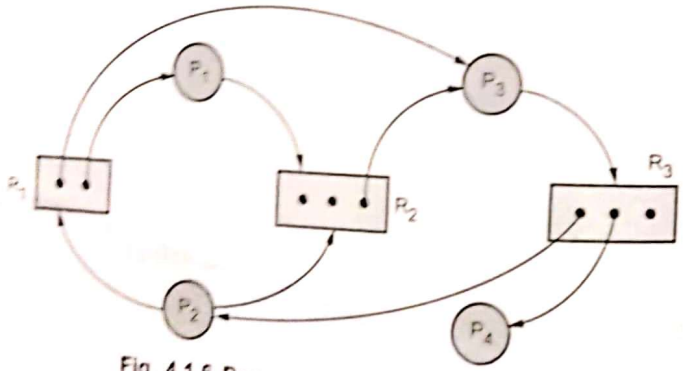


Fig. 4.1.6 Resource allocation graph

Resource	Number of instance
R_1	2
R_2	3
R_3	3

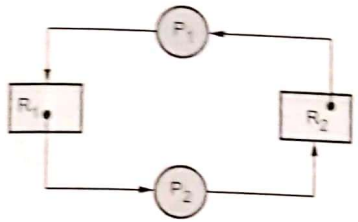


Fig. 4.1.7 Circular wait with deadlock

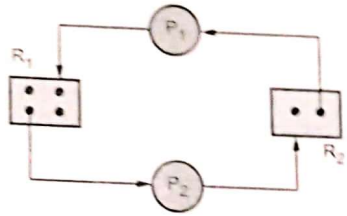


Fig. 4.1.8 Circular wait but no deadlock

Fig. 4.1.9 shows the resource allocation graph. System consists of four processes (P_1, P_2, P_3, P_4) and four resources (R_1, R_2, R_3, R_4).

Resource	Number of instance
R_1	2
R_2	2
R_3	3
R_4	6
R_5	3

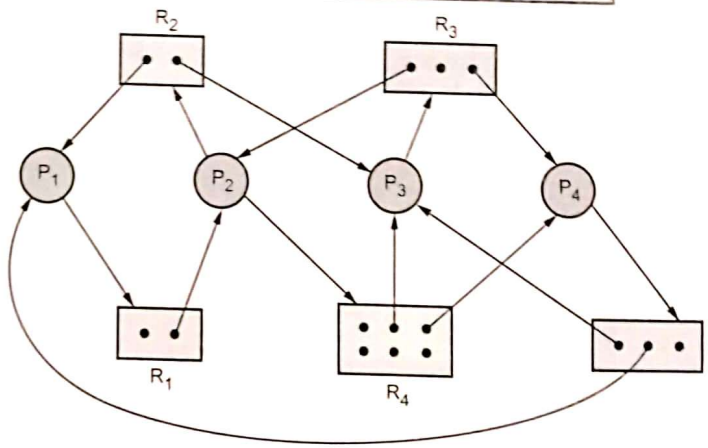


Fig. 4.1.9 Resource allocation graph

- Resource managers and other operating system processes can be involved in a deadlock. Deadlock is a global condition rather than a local one.
- An individual program cannot generally detect a deadlock. It is blocked and unable to use the processor to do any work. Deadlock detection must be handled by the operating system.

University Questions

1. What is Deadlock ? List the conditions that lead to deadlock. How Deadlock can be prevented ? **GTU : Winter-13, Summer-14, W**
2. Define and explain following terms : **GTU : Summer-13, W**

Deadlock

4.2 Deadlock Characteristics

4.2.1 Necessary Condition for Deadlock

- Following four conditions are necessary for deadlock to exist.
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait

1. **Mutual exclusion :** A resource may be acquired exclusively by one process at a time.
2. **Hold and wait :** Processes currently holding resources that were granted earlier can request new resources.
3. **No preemption :** Once a process has obtained a resource, the system cannot remove it from the process control until the process has finished using the resource.
4. **Circular wait :** A circular chain of hold and wait condition exists in the system.

- All four of these conditions must be present for a resource deadlock to occur. If one of them is absent, no resource deadlock is possible.

4.3 Deadlock Solution

- There are four approaches for deadlock solution.
  1. Deadlock prevention
  2. Deadlock avoidance
  3. Deadlock detection
  4. Deadlock recovery

- In deadlock **prevention,** aim is to condition a system to remove any possibility of deadlock occuring. Poor resource utilization may be possible.
- **Avoidance :** Deadlocks can be avoided by clearly identifying safe states and unsafe states.
- **Detection :** Deadlock detection methods are used in systems in which deadlocks can occur.
- **Recovery :** Used to resolve the deadlock from a system.
- Deadlock prevention and detection algorithm is used for ignoring the deadlock.

4.4 Deadlock Prevention

- To prevent a deadlock, the OS must eliminate one of the four necessary conditions.
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait

1. **Mutual exclusion :** It is necessary in any computer system because some resources (memory, CPU) must be exclusively allocated to one user at a time. No other process can use a resource while it is allocated to a process.
2. **Hold and wait :** If a process holding certain resources is denied a further request, it must release its original resources and if required request them again.
3. **No preemption :** It could be bypassed by allowing the operating system to deallocate resources from process.
4. **Circular wait :** Circular wait can be bypassed if the operating system prevents the formation of a circle.

- A deadlock is possible only if all four of these conditions simultaneously hold in the system.
- Prevention strategies ensure that at least one of the conditions is always false.

4.5 Deadlock Avoidance

GTU : Winter-13, Summer-13, 14

- Deadlock avoidance depends on additional information about the long term resource needs of each process. The system must be able to decide whether granting a resource is safe or not and only make the allocation when it is safe.
- When a process is created, it must declare its maximum claim, i.e. the maximum number of unit resource. The resource manager can grant the request if the resources are available.

- For example, if process 1 requests a resource held by process 2 then make sure that process 2 is not waiting for resource held by first process 1.

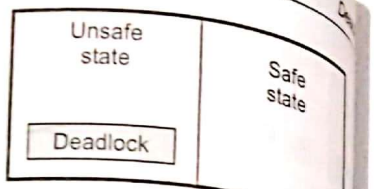


Fig. 4.5.1 Safe and unsafe state

4.5.1 Banker's Algorithm

- Banker's algorithm is the deadlock avoidance algorithm. Banker's is named because the algorithm is modeled after a banker who makes loads from a pool of money and receives payments that are returned to that pool.
- Algorithm is check to see if granting the request leads to an unsafe state. If it does, the request is denied. If granting the request leads to a safe state, it is carried out.
- The Dijkstra proposed an algorithm to regulate resource allocation to avoid deadlocks. The banker's algorithm is the best known of the avoidance method.
- By using avoidance method, the system is always kept in a safe state.
- It is easy to check if a deadlock is created by granting a request, deadlock avoidance method is used for this.
- Deadlock avoidance uses the worst-case analysis method to check for the deadlocks.
- **Safe state :** There is at least one sequence of resource allocations to processes that does not result in a deadlock.
- System is in a safe state only if there exist a safe sequence. A safe state is not a deadlocked state.
- Deadlocked state is an unsafe state. It does mean the system is in a deadlock.
- As long as the state is safe, the resource manager can be guaranteed to avoid deadlock.
- Initially the system is in a safe state. When process requests a resource and the resource is available then the system must decide whether the resources can be allocated immediately or process must wait.
- If system remains in safe state after allocating resource then only OS allocate resources to process.
- Banker algorithm uses following data structures.

1. Allocation : Allocation is a table in which row represents process and column represents resources (R).

alloc [i, j] = Number of unit of resource R_j held by process P_i.

2. Max : Max be the maximum number of resources that process requires during its execution.

- **Need (claim) :** It is current claim of a process, where a process's claim is equal to its maximum need minus its current allocation.

  Need = Max – Allocation

- **Available :** There will be number of resources still available for allocation. This is equivalent to the total number of resources minus the sum of the allocation to all processes in the system.

  Available = Number of resources – Sum of the allocation to all process

  $$= \text{Number of resources} - \sum_{i=1}^{n} \text{Allocation}(P_i)$$

- Each process cannot request more that the total number of resources in the system. Each process must also guarantee that once allocated a resource, the process will return that resource to the system within a finite time.

Weakness of Banker's algorithm

1. It requires that there be a fixed number of resources to allocate.
2. The algorithm requires that users state their maximum needs (request) in advance.
3. Number of users must remain fixed.
4. The algorithm requires that the bankers grant all requests within a finite time.
5. Algorithm requires that process returns all resource within a finite time.

Examples on Banker's algorithm

1. System consists of five processes $(P_1, P_2, P_3, P_4, P_5)$ and three resources (R_1, R_2, R_3). Resource type R_1 has 10 instances, resource type R_2 has 5 instances and R_3 has 7 instances. The following snapshot of the system has been taken :

Process	Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	0	7	5	3	3	3	2
P_2	2	0	0	3	2	2			
P_3	3	0	2	9	0	2			
P_4	2	1	1	2	2	2			
P_5	0	0	2	4	3	3			

Content of the matrix need is calculated as Need = Max – Allocation.

Process	Need		
	R_1	R_2	R_3
P_1	7	4	3
P_2	1	2	2
P_3	6	0	0
P_4	0	1	1
P_5	4	3	1

Currently the system is in safe state.

Safe sequence : Safe sequence is calculated as follows :

1) Need of each process is compared with available. If $need_i \leq available_j$, the resources are allocated to that process and process will release the resource.

2) If need is greater than available, next process need is taken for comparison.

3) In the above example, need of process P_1 is (7, 4, 3) and available is (3, 3, 2)

need ≥ available → False

So system will move for next process.

4) Need for process P_2 is (1, 2, 2) and available (3, 3, 2), so

need ≤ available (work)

(1, 2, 2) ≤ (3, 3, 2) = True

then Finish [i] = True

Request of P_2 is granted and processes P_2 is release the resource to the system

Work := Work + Allocation

Work := (3, 3, 2) + (2, 0, 0)

:= (5, 3, 2)

This procedure is continued for all processes.

5) Next process P_3 need (6, 0, 0) is compared with new available (5, 3, 2).

Need > Available = False

(6 0 0) > (5 3 2)

6) Process P_4 need (0, 1, 1) is compared with available (5, 3, 2).

Need < Available

(0 1 1) < (5 3 2) = True

Available = Available + Allocation

= (5 3 2) + (2 1 1)

= (7 4 3) (New available)

7) Then process P_5 need (4 3 1) is compared with available (7 4 3)

Need < Available

(4 3 1) < (7 4 3) = True

Available = Available + Allocation

= (7 4 3) + (0 0 2) = (7 4 5) (New available)

8) One cycle is completed. Again system takes all remaining process in sequence. So process P_1 need (7 4 3) is compared with new available (7 4 5).

Need < Available = True

(7 4 3) < (7 4 5)

Available = Available + Allocation

= (7 4 5) + (0 1 0) = (7 5 5) (New available)

9) Process P_3 need is (6 0 0) is compared with new available (7 5 5).

∴ Need < Available = True

(6 0 0) < (7 5 5) = True

Available = Available + Allocation

= (7 5 5) + (3 0 2)

= (10 5 7) = (New available)

Safe sequence is < P_2 P_4 P_5 P_1 P_3 >

Example 4.5.1 *Consider the following snapshot of a system :*

Process	Allocation				Request				Available			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4

P_1	0	0	1	2	0	0	1	2	1	5	2	0
P_2	1	0	0	0	0	1	7	5	0			
P_3	1	3	5	4	2	3	5	6				
P_4	0	6	3	2	0	6	5	2				
P_5	0	0	1	4	0	6	5	6				

Answer the following questions using Banker's algorithm :

i) What is the content of 'need' matrix ?

ii) Is the system in a safe state ? (Show calculations)

iii) If a request from P_2 arrives for (0, 4, 2, 0), can the request be granted immediately

Solution :

i) Content of need matrix is

Process	R_1	R_2	R_3	R_4
P_1	0	0	0	0
P_2	0	7	5	0
P_3	1	0	0	2
P_4	0	0	2	0
P_5	0	6	4	2

ii) Safe state

a) Need of P_1 = (0, 0, 0, 0)

Available = (1, 5, 2, 0)

Need < Available

(Required resources are allocated to P_1)

∴ New available = Allocation + Available

= (0, 0, 1, 2) + (1, 5, 2, 0)

= (1, 5, 3, 2)

b) For P_2 process

Need = (0, 7, 5, 0)

Available = (1, 5, 3, 2)

Need > Available

(Request is not granted)

New available = Available = (1, 5, 3, 2)

c) For P_3 process

Need = (1, 0, 0, 2)

Available = (1, 5, 3, 2)

Need < Available

(Request is granted)

New available = Allocation + Available = (1, 3, 5, 4) + (1, 5, 3, 2)

= (2, 8, 8, 6)

d) For process P_4

Need of P_4 = (0, 0, 2, 0)

Available = (2, 8, 8, 6)

Need < Available

New available = (2, 14, 11, 8)

e) Process P_5

Need of P_5 = (0, 6, 4, 2)

Available = (2, 14, 11, 8)

Need < Available

New available = (2, 14, 11, 8) + (0, 0, 1, 4)

= (2, 14, 12, 12)

f) Again for P_2 process

Need of P_2 = (0, 7, 5, 0), Available = (2, 14, 12, 12)

Need < Available

New available = (1, 0, 0, 0) + (2, 14, 12, 12)

= (3, 14, 12, 12)

So the safe sequence is P_1, P_3, P_4, P_5, P_2

iii) Request from P_2 = (0, 4, 2, 0)

Available = (1, 5, 2, 0)

After granting the request of P_2, available resource is (1, 1, 0, 0)

a) Need of P_1 = (0, 0, 0, 0)

P_1	0	0	1	2	0	0	1	2	1	5	2	0
P_2	1	0	0	0	1	7	5	0				
P_3	1	3	5	4	2	3	5	6				
P_4	0	6	3	2	0	6	5	2				
P_5	0	0	1	4	0	6	5	6				

Answer the following questions using Banker's algorithm :

i) What is the content of 'need' matrix ?

ii) Is the system in a safe state ? (Show calculations)

iii) If a request from P_2 arrives for (0, 4, 2, 0), can the request be granted immediately ?

Solution :

i) Content of need matrix is

Process	R_1	R_2	R_3	R_4
P_1	0	0	0	0
P_2	0	7	5	0
P_3	1	0	0	2
P_4	0	0	2	0
P_5	0	6	4	2

ii) Safe state

a) Need of P_1 = (0, 0, 0, 0)

Available = (1, 5, 2, 0)

Need < Available

(Required resources are allocated to P_1)

∴ New available = Allocation + Available

= (0, 0, 1, 2) + (1, 5, 2, 0)

= (1, 5, 3, 2)

b) For P_2 process

Need = (0, 7, 5, 0)

Available = (1, 5, 3, 2)

Need > Available

(Request is not granted)

New available = Available = (1, 5, 3, 2)

c) For P_3 process

Need = (1, 0, 0, 2)

Available = (1, 5, 3, 2)

Need < Available

(Request is granted)

New available = Allocation + Available = (1, 3, 5, 4) + (1, 5, 3, 2)

= (2, 8, 8, 6)

d) For process P_4

Need of P_4 = (0, 0, 2, 0)

Available = (2, 8, 8, 6)

Need < Available

New available = (2, 14, 11, 8)

e) Process P_5

Need of P_5 = (0, 6, 4, 2)

Available = (2, 14, 11, 8)

Need < Available

New available = (2, 14, 11, 8) + (0, 0, 1, 4)

= (2, 14, 12, 12)

f) Again for P_2 process

Need of P_2 = (0, 7, 5, 0), Available = (2, 14, 12, 12)

Need < Available

New available = (1, 0, 0, 0) + (2, 14, 12, 12)

= (3, 14, 12, 12)

So the safe sequence is P_1, P_3, P_4, P_5, P_2

iii) Request from P_2 = (0, 4, 2, 0)

Available = (1, 5, 2, 0)

After granting the request of P_2, available resource is (1, 1, 0, 0)

a) Need of

New available = (1, 1, 0, 0) + (0, 0, 1, 2)

= (1, 1, 1, 2)

b) P_2 need is greater than available.

c) P_3 need is (1, 0, 0, 2)

Need < Available

New available = (1, 1, 1, 2) + (1, 3, 5, 4)

= (2, 4, 6, 6)

d) P_4 Need is (0, 0, 2, 0)

Need < Available

New available = (2, 4, 6, 6) + (0, 6, 3, 2)

= (2, 10, 9, 8)

e) P_5 Need is (0, 6, 4, 2)

Available (2, 10, 9, 8)

Need < Available

∴ New available = (2, 10, 9, 8) + (0, 0, 1, 4)

= (2, 10, 10, 12)

f) Again P_2 need (0, 7, 5, 0)

Need < Available

If a request from P_2 arrives for (0, 4, 2, 0), then **the request is granted immediately**

Example 4.5.2 *The operating system contains 3 resources, the number of instance of each resource type are 7, 7, 10. The current resource allocation state is as shown below.*

Process	Current allocation			Maximum need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8
P_2	2	0	3	4	3	3
P_3	1	2	4	3	4	4

i) Is the current allocation in a safe state ?

ii) Can the request made by process P_1 (1 1 0) be granted ?

Solution : **i)** First find the available resource in the system.

Available : = Number of instance – Sum of allocation

Process	Current allocation		
	R_1	R_2	R_3
P_1	2	2	3
P_2	2	0	3
P_3	1	2	4
	5	4	10

⇐ Sum

Available = (7 7 10) – (5 4 10)

Available resources = (2 3 0)

Content of need matrix is

Process	Need		
	R_1	R_2	R_3
P_1	1	4	5
P_2	2	3	0
P_3	2	2	0

Safe sequence < P_3, P_2, P_1 >.

The system is in safe state.

ii) Process P_1 request (1 1 0), this request is less than need. Need for process P_1 is (1 4 5). Available resource is (2, 3, 0) and request is (1 1 0).

∴ Request < Available

(1 1 0) < (2 3 0)

After allocating (1 1 0) to process P_1 the need becomes as follows.

Process	Need		
	R_1	R_2	R_3
P_1	0	3	5
P_2	2	3	0
P_3	2	2	0

And available resource is (1 2 0).

Need of any process is never satisfied after granting the process P_1 request (1 1 0). So the system will be blocked. Therefore request of process P_1 (1 1 0) cannot be granted.

Example 4.5.3 *Consider following snapshot.*

Process	Allocation		Max		Available	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	1	2	4	2	1	1
P_2	0	1	1	2		
P_3	1	0	1	3		
P_4	2	0	3	2		

Whether the system is safe or unsafe ?

Solution : First calculate Need

Need = Max – Allocation

Process	Need	
	R_1	R_2
P_1	3	0
P_2	1	1
P_3	0	3
P_4	1	2

System is in safe state with safe sequence < P_2, P_4, P_1, P_3 >. System will comp its operation in this sequence.

Example 4.5.4 *Consider following snapshot.*

Process	Allocation		Max		Available	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	7	2	9	5	2	1
P_2	1	3	2	6		
P_3	1	1	2	2		
P_4	3	0	5	0		

i) Calculate content of need matrix. ii) System is safe or unsafe.

Solution : i) Need = Max – Allocation

Process	Need	
	R_1	R_2
P_1	2	3
P_2	0	3

P_3	1	3
P_4	2	0

iii) System is unsafe state. Resources are not available for processes P_1 and P_2 to complete their operation.

University Questions

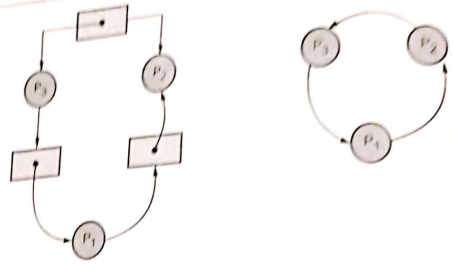
1. *Explain the use of Banker's algorithm for multiple resources for deadlock avoidance with illustration.* **GTU : Winter-13, Marks 7**
2. *How resource trajectories can be helpful in avoiding the deadlock ?* **GTU : Summer-13, Marks 7**
3. *Write short note on : Banker's algorithm for a single resource.* **GTU : Summer-13, Marks 4**
4. *What do you mean by deadlock avoidance ? Explain the use of Banker's algorithm for deadlock avoidance with illustration.* **GTU : Summer-14, Marks 7**

4.6 Deadlock Detection

- Deadlock detection is the process of determining that a deadlock exists and identifing the processes and resources involved in the deadlock.
- If processes are blocked on resources for an inordinately long time, the detection algorithm is executed to determine whether the current state is a deadlock.
- Deadlock detection algorithms can incur significant runtime overhead.

4.6.1 Wait for Graph

- Any resource allocation graph with a single copy of resources can be transferred to a **wait for graph.**
- Fig. 4.6.1 shows resource allocation graph with corresponding wait for graph. The state of the system can be modeled by directed graph, called a wait for graph. (See Fig. 4.6.1 on next page).
- Wait for graph is a graph where each node represents a process. An edge $P_i \rightarrow P_j$ means that process P_i is blocked and waiting for process P_j to release a resource.
- The wait for graph of a system is always smaller than the resource allocation graph of that same system. There is a deadlock in a system if and only if there is a loop in the wait for graph of that system.
- Deadlock detection involves two issues :
  1. Maintenance of the wait for graph.
  2. Searching of the wait for graph for the presence of cycles.



(a) Resource allocation graph (b) Wait for graph

Fig. 4.6.1

- Deadlock detection requires overhead for run time cost of maintaining necessary information and executing the detection algorithm.
- For multiple instances of a resource type use an algorithm similar to banker's algorithm.

4.7 Deadlock Recovery

- Once deadlock has been detected in the system, the deadlock must be broken by removing one or more of the four necessary conditions.
- Here one or more processes will have to be preempted, thus releasing the resources so that the other deadlocked processes can become unblocked.

1. Process termination

- Deadlock is removed by aborting a process. But aborting process is not easy. All deadlocked processes are aborted.
- Circular wait is eliminated by aborting one by one process. There will be lot of overhead. Deadlock detection algorithm must rerun after each process kill.
- Selection of process for aborting is difficult. Following parameters are considered.
  1. Priority of the process.
  2. What percentage the process finished its execution ?
  3. Resource used by process.
  4. Need of resources to complete process remaining operation.
  5. How many processes will need to be terminated ?
  6. Process type : Batch or interactive.

2. Resource preemption

- Some times, resource temporarily take away from its current process and allocate it to another process.
- For selecting victim, following factors are considered.
  1. Priority of the process. Higher priority process are usally not selected.
  2. CPU time used by process. The process which is close to completion are usually not selected.
  3. The number of other process that would be affected if this process were selected as the victim.

3. Recovery through rollback

- When a process in a system terminates, the system performs a rollback by undoing every operation related to the terminated process.
- Checkpointing a process means that its state is written to a file so that it can be restarted later.
- Risk in this method is that the original deadlock may recover but the nondeterminancy of concurrent processing may ensure that this does not happen.

4. Starvation

- Starvation is one type situation in which a process waits for an event that might never occur in the system.
- Select the victim only for finite number of times. Use rollback method for selecting victim process.

4.8 Comparison between Detection, Prevention and Avoidance Methods of Handling Deadlock

Parameters	Avoidance	Detection	Prevention
Resource allocation policy	Midway between that of detection and prevention.	Very liberal.	Conservative under commit resources.
Different schemes	Manipulate to find at least one safe path.	Invoke periodically to test for deadlock.	Preemption, resource ordering, requesting all resources at once.
Advantages	No preemption necessary.	Never delays process initiation.	No preemption necessary.
Disadvantages	Process can be blocked for long period.	Inherent preemption losses.	Delays process initiation.

Example 4.8.1 *Consider a system with five processes and three resource types and at time the following snapshot of the system has been taken :*

	Allocated			Maximum			Available		
Process Id	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	1	2	4	3	3	3	1	
P2	2	1	2	3	2	2			
P3	4	0	1	9	0	2			
P4	0	2	0	7	5	3			
P5	1	1	2	11	2	3			

i) Determine the total amount of resources of each type.

ii) Compute the need matrix

iii) Determine if the state is safe or not using Banker's algorithm.

iv) Would the following request be granted in the current state?

(a) P1 <3, 3, 1>

(b) P2<2, 1,0>

MMTU : 2013

Solution :

i) The total amount of resources of each type : Sum of allocated + Available

	Allocated			Available		
Process Id	R1	R2	R3	R1	R2	R3
P1	1	1	2	3	1	0
P2	2	1	2			
P3	4	0	1			
P4	0	2	0			
P5	1	1	2			
Total	8	5	7			

	R1	R2	R3
e total amount of resources of each type =	8	5	0 (Allocated)
	3	1	0 (Available)
e total amount of resources of each type =	11	6	7

R1 = 11
R2 = 6
R3 = 7

ii) The need matrix

Need Matrix = Maximum – Allocated

	Maximum				Allocated				Need Matrix		
Process Id	R1	R2	R3		R1	R2	R3		R1	R2	R3
P1	4	3	3		1	1	2		3	2	1
P2	3	2	2	–	2	1	2	=	1	1	0
P3	9	0	2		4	0	1		5	0	1
P4	7	5	3		0	2	0		7	3	3
P5	11	2	3		1	1	2		10	1	1

iii) State is safe or not using Banker's algorithm

Steps for safe state :

First Iteration :

1. Process P1 need is (3, 2, 1) and resource available with system is (3, 1, 0).

Here need is greater than available i.e.

Need (3, 2, 1) > Available (3, 1, 0)

Condition is false and algorithm selects next process for resource allocation.

2. Process P2 need is (1, 1, 0) and resource available with system is (3, 1, 0).

Here need is less than available i.e.

Need (1, 1, 0) < Available (3, 1, 0)

Condition is true and algorithm allocates required resources to that process (P2 Process). After completion of work, process P2 releases all the resources. So resources available with system are increased. So,

New Available = Previous available + Allocated of particular process

= (3, 1, 0) + (2, 1, 2)

New Available = (5, 2, 2)

3. Process P3 need is (5, 0, 1) and resource available with system is (5, 2, 2).

Here need is less than available i.e.

Need (5, 0, 1) < Available (5, 2, 2).

Condition is true and algorithm allocates required resources to that process (P3 Process). After completion of work, process P3 releases all the resources. So resources available with system are increased. So,

New Available = Previous available + Allocated of particular process

= (5, 2, 2) + (4, 0, 1)

New Available = (9, 2, 3)

4. Process P4 need is (7, 3, 3) and resource available with system is (9, 2, 3).

Here need is greater than available i.e.

Need (7, 3, 3) > Available (9, 2, 3)

Condition is false and algorithm selects next process for resource allocation.

5. Process P5 need is (10, 1, 1) and resource available with system is (9, 2, 3).

Here need is greater than available i.e.

Need (10, 1, 1) > Available (9, 2, 3)

Condition is false and algorithm selects next process for resource allocation.

One loop is completed; again we start for remaining three processes P1, P4 and P5

Second Iteration: for process P1, P4 and P5

1. Process P1 need is (3, 1, 2) and resource available with system is (9, 2, 3).

Here need is less than available i.e.

Need (3, 1, 2) < Available (9, 2, 3).

Condition is true and algorithm allocates required resources to that process (P1 Process). After completion of work, process P1 releases all the resources. So resources available with system are increased. So,

New Available = Previous available + Allocated of particular process

= (9, 2, 3) + (1, 1, 2)

New Available = (10, 3, 5)

2. Process P4 need is (7, 3, 3) and resource available with system is (10, 3, 5).

Here need is less than available i.e.

Need (7, 3, 3) < Available (10, 3, 5).

Condition is true and algorithm allocates required resources to that process (P4 Process). After completion of work, process P4 releases all the resources. So resources available with system are increased. So,

New Available = Previous available + Allocated of particular process

= (10, 3, 5) + (0, 2, 0)

New Available = (10, 5, 5)

3. Process P5 need is (10, 1, 1) and resource available with system is (10, 5, 5)

Here need is less than available i.e.

Need (10, 1, 1) < Available (10, 5, 5)

Condition is true and algorithm allocates required resources to that process (P5 Process). After completion of work, process P5 releases all the resources. So resources available with system are increased. So,

New Available = Previous available + Allocated of particular process

= (10, 5, 5) + (1, 1, 2)

New Available = (11, 6, 7)

Safe sequence = P2, P3, P1, P4, P5

iv) Would the following request be granted in the current state ?

a) P1 <3, 3, 1>

We assume that requested is granted for process P1. Then the need matrix will be

	Maximum			Allocated			Need Matrix		
Process Id	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	4	3	3	4	4	3	0	0	0
P2	3	2	2	2	1	2	1	1	0
P3	9	0	2	4	0	1	5	0	1
P4	7	5	3	0	2	0	7	3	3
P5	11	2	3	1	1	2	10	1	1

But available with the system is less that the required one. So P1 requested is not granted.

b) P2<2, 1, 0>

We assume that requested is granted for process P2. Then the need matrix will be

Process Id	Maximum R1	Maximum R2	Maximum R3		Allocated R1	Allocated R2	Allocated R3		Need Matrix R1	Need Matrix R2	Need Matrix R3
P1	4	3	3		1	1	2		3	2	1
P2	3	2	2	–	4	2	2	=	0	0	0
P3	9	0	2		4	0	1		5	0	1
P4	7	5	3		0	2	0		7	3	3
P5	11	2	3		1	1	2		10	1	1

New Available = (3, 1, 0) - (2, 1, 0) = (1, 0, 0)

Compute the safe sequence :

1. Need of P1 (3, 2, 1) is greater than available (1, 0, 0). Condition is false. Select process.
2. Need of P2 (0, 0, 0) is less than available (1, 0, 0). Condition is true. Then

   New available = (1, 0, 0) + (4, 2, 2) = (5, 2, 2).
3. Need of P3 (5, 0, 1) is greater than available (5, 2, 2). Condition is false. Select process.
4. Need of P4 (7, 3, 3) is greater than available (5, 2, 2). Condition is false. Select process.
5. Need of P5 (10, 1, 1) is greater than available (5, 2, 2). Condition is false. S next process.

Again repeat for second iteration

1. Need of P1 (3, 2, 1) is greater than available (5, 2, 2). Condition is true. Then

   New available = (5, 2, 2) + (1, 1, 2) = (6, 3, 4)

There is no safe sequence because need of process P3, P4 and P5 is greater the available. So request is not granted immediately.

5 Memory Management

Syllabus

Basic memory management : *Definition, Logical and physical address map, Memory allocation : Contiguous memory allocation - Fixed and variable partition - Internal and external fragmentation and compaction, Paging : Principle of operation - Page allocation - Hardware support for paging - Protection and sharing - Disadvantages of paging.*

Virtual memory : *Basics of virtual memory - Hardware and control structures - Locality of reference, Page fault , Working set, Dirty page/Dirty bit - Demand paging (Concepts only) - Page replacement policies : Optimal (OPT), First in First Out (FIFO), Second Chance (SC), Not Recently Used (NRU) and Least Recently Used (LRU).*

Contents

5.1 Basic Memory Management . . . . . **Summer-13,** . . . . . Marks 7
5.2 Swapping
5.3 Contiguous Memory Allocation with Fixed Partitions
. . . . . **Winter-14,** . . . . . Marks 7
5.4 Contiguous Memory Allocation with Variable Partitions
. . . . . **Summer-14,** . . . . . Marks 7
5.5 Paging
5.6 Segmentation . . . . . **Summer-13,** . . . . . Marks 2
5.7 Paged Segmentation
5.8 Virtual Memory Concepts . . . . . **Winter-13,** . . . . . Marks 7
5.9 Demand Paging
5.10 Page Replacement Algorithms . . . . . **Summer-13, Winter-13,** . . . . Marks 7
5.11 Thrashing

5.1 Basic Memory Management

GTU : Summer-10

- Memory is used to store information. Secondary storage memory is long t persistent memory that is held in storage device such as disk drive.
- Primary memory is faster than secondary memory. Memory manage responsible for allocating primary memory to processes.
- Memory management is performed by both software and special pur hardware. The memory manager is an operating system component. Managing sharing of primary memory and minimizing memory access time are the goals of the memory manager.

Primary memory requirements

1. Access time : It should be as small as possible. This need influences software and hardware design.
2. Size : Size must be as large as possible. It can accomodate many program memory.
3. Cost : Cost of the memory is less than the total cost of the computer.

5.1.1 Memory Management Function

1. Allocate primary memory space to processes.
2. Minimize access time.
3. Determining allocation policy for memory.
4. Deallocation technique and policy.

5.1.2 Basic Hardware of Memory

- CPU can access content of main memory and register directly. If the data available into the memory, it load into memory from disk.
- Registers are built on the processor. Using one cycle of the CPU clock, p access data from register.
- Accessing memory may take many CPU clock cycle. Mismatch of speed be CPU and memory is overcome by using cache memory.
- The use of base and bound (limit) registers are restrict a process references upto a certain limit. Hardware is used to protect user address spa
- Each process requires its own address space operating system define legal for each process. Maximum and minimum limit is also decided so that proc access only these legal address.

- Fig. 5.1.1 shows the protection of process by using registers.
- An address space is the set of addresses that a process/program can use to address main memory. Each process has its own address space.
- User programs are loaded into consecutive memory locations by using base and limit register. When process is executing, the base register is loaded with the physical address where its program begins in memory and the limit register is loaded with the length of the program.

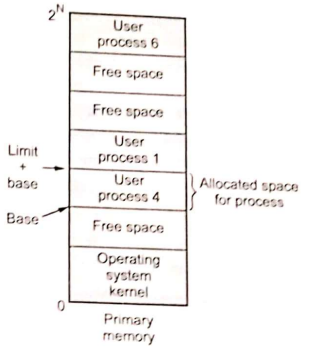


Fig. 5.1.1 Address space and memory

- Memory protection is used to avoid interference between programs existing in main memory. The memory protection hardware compares every memory address used by the program with the contents of two registers (base and limit) to ensure that it lies with the allocated memory area.
- Multiple hardware memories are used to provide a larger address space.
- The simplest method of memory protection is adding two registers to the CPU. This works good for all memory is allocated contiguously. Non-contiguous memory is harder to protect.
- When a process reads from or writes to address, the memory decoder adds on the value of the base register. The actual operation of read or write to address = base register + limit register.
- If the input address is higher than limit or lower than zero, then the memory hardware generates error. This is informed to the operating system by using interrupt. Processes can only access memory within these limits.
- Each process has its own pair of base register and limit register.

5.1.3 Address Space Mapping

- Secondary storage device stores program in binary executable format. Before executing, the program is loaded into the main memory.
- Most of the operating systems allow a user process to store in any section of the main memory. Source program uses symbolic addresses.

- Binding of instruction and data to main memory address is following ways :
  1. Compile time 2. Load time 3. Execution time
- **Compile time** : Source program is translated at compile time to prod relocatable object module. At compile time, the translator generates co allocate storage for the variable. This storage address is used for code ref Target address is unknown at compile time, it cannot be bound at compil Example of compile time binding is MS DOS.com programs.
- **Load time** : Compiler generates relocatable code if compile time binding performed. The loader modifies the addresses in the load module at load produce the executable image stored in main memory. Final binding is d until program load time.
- **Execution time** : Memory address of the program is changed at execution time, then execution time binding is used. Binding is delayed until the run time of the program. Normally all operating system uses execution time binding. Special hardware is used for execution time binding.

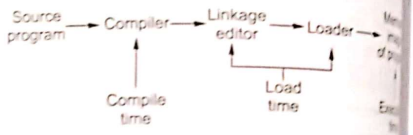


Fig. 5.1.2 Processing of user program

- Memory allocation and deallocation is done using run-time support programming language in which a program is coded. Allocation and dea requests are made by calling appropriate routines of the run time library.
- Kernel is not involved in this kind of memory management.

5.1.4 Concept of Memory Address

- **Logical address** is generated by the CPU. This address is also called address.
- Main memory address uses **physical address**. This address also calle address.
- **Logical address space** : Set of all logical addresses generated by a program
- Logical address and physical address is identical when load time and compi address binding is performed. The execution time address binding pr different physical and logical address.
- Memory Management Unit (MMU) is responsible for run time address m from virtual to physical address.

Dynamic Relocation

- Base register is sometimes called as a relocation register. The value of the relocation register is added to every address generated by a user process at the time it is sent to main memory.
- User can load a process with only absolute addresses for instructions and data, only when those specific addresses are free in main memory. Program's instruction, data and any other data structure required by the process can be accessed easily if the addresses are relative.
- Fig. 5.1.3 shows dynamic relocation. User programs never reads the main memory physical address.
- Dynamic relocation requires extra hardware. It is mapping of the virtual address space to the physical address space at run time.
- Dynamic relocation makes it possible to move a partially executed process from one area of main memory into another without affecting other process.

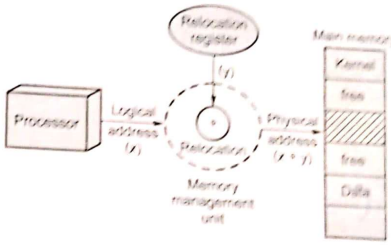


Fig. 5.1.3 Dynamic relocation

- Problem with relocation is that, it is necessary to perform an addition and a comparison on every memory reference.
- For good memory management, logical address space is bound with a separate physical address space.

5.1.5 Dynamic Loading

- Dynamic loading is used for better memory space utilization. User program size is large as compared to the memory size. Program or process are dynamically loaded into memory as per required.
- With dynamic loading, a routine is not loaded until it is called. All routines are kept on storage disk in a relocatable load format. The main program is loaded into memory and is executed.

Advantages

1. Unused routine is never loaded.
2. Special support is not required from OS.
3. Used with error routines.

University Question

1. Establish the necessity for memory management. Explain the memory management with the [illegible] linked lists. **GTU : Summer-13, Mark**

5.2 Swapping

- Keeping all processes in memory all the time requires a large amount of main memory. For execution of the process, it must be swapped.
- Swapping is method of temporarily removing inactive process/programs from the physical memory.
- Inactive program means, which is neither executing on the CPU, nor performing an I/O operation.

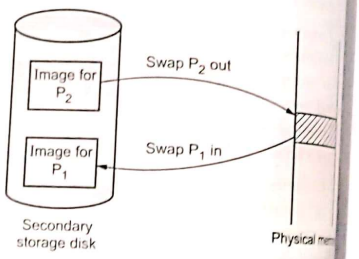


Fig. 5.2.1 Swapping

- Fig. 5.2.1 shows swapping.
- When process P_2 send requests for an I/O operation, OS blocked P_2 proces[s] will not return to the ready state for long period of time. Process P_2 is in b[lo]state. Process manager place the process P_2 in blocked state and inform [the] memory manager regarding this action.
- Process manager moves a blocked process P_1 into the ready state and infor[ms] memory manager. Process P_1 is loaded into main memory by memory ma[nager] when there is space available.
- Swapped out process is stored on the secondary storage disk. It co[ntains] executable image with code, data and stack.
- Relocation hardware supports for swapping. Because of address binding [at] swapping is difficult without relocation hardware.

- Round robin and priority based CPU scheduling algorithm uses the swapping concept.
- Time sharing system uses swapping. In time sharing, kernel can estimate when a program is likely to be scheduled next.
- A swapped out process will occupy same memory location when process swap in by OS. But this totally depends on address binding. If address binding is load time or assembly time, then swapped out process uses same memory location when swap-in. But execution-time binding uses different memory location.
- Secondary storage is required to implement the swapping technique.
- Content switching time increases when swapping increases. Major part of the swap time is transfer time. Swapping increases the operating system overhead due to the disk I/O involved.
- When swapping creates multiple holes in memory, it is possible to combine them all into one by moving all the processes to one side of memory. This technique is known as **memory compaction.**
- Compaction requires a lot of CPU time, so most operating systems not used this technique.
- Compaction means movement of processes in the memory during their execution. It is sometimes called dynamic relocation of a program.

5.3 Contiguous Memory Allocation with Fixed Partitions

GTU : Winter-14

- In a fixed size partitioning of the main memory all partitions are of the same size. The operating system divides main memory into a number of fixed size partition. Each partition holds a single program.
- Fixed sized partitions are relatively simple to implement. Degree of multiprogramming depends on the number of partitions.
- Normally main memory is divided into two partitions :
  1. For resident program
  2. For user processes
- Batch operating system uses the fixed size partition scheme. The operating system keeps the record of memory allocation and deallocation in the form of table. Initially all the memory is available for user processes.
- When the process arrives in the system and needs memory, operating system search for large hole for this process. Hole is one large block of free available memory. If any free hole is found, process is allocated to the free hole of memory as is needed.

- After allocating number of holes for the processes, a set of various size holes is scattered throughout memory at any given time. When a process arrives and searches for (memory) set of holes. But the holes must be large enough to accomodate the process.
- Fixed sized partitions are simple to implement. Any process whose size is less than or equal to the partition size can be loaded into any available partition.

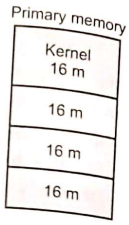


Fig. 5.3.1 Fixed size partition

- Fig. 5.3.1 shows an example of fixed partitioning of a memory.
- There are two difficulties with the use of equal size fixed partitions :
  1. A program may be too big to fit into a partition.
  2. Memory utilization is extremely inefficient.
- First difficulty is solved by using overlays. Overlays involves moving data a program segments in and out of memory.

Advantages and disadvantages of fixed partition size

1) Advantages

1. Simple to implement.
2. Does not require expertise to understand and use such a system.
3. Less overhead.

2) Disadvantages

1. Memory is not fully utilized.
2. Poor utilization of processors.
3. User's process being limited to the size of available main memory.
4. Requires contiguous loading of entire program.

5.3.1 Dynamic Memory Partitions

- Memory partitions are of variable length and number. Processes are loaded the size nearest to its requirements.
- Fig. 5.3.2 shows the effect of dynamic memory partitioning. Initially in memory, only OS is loaded and rest of the memory is free. Memory size 64 Mbytes. How the process is loaded with dynamic partitions is shown be After loading all the process some free space remains at one side of the memor

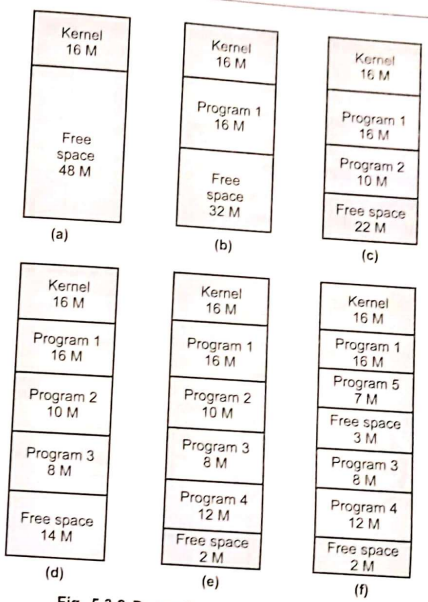


Fig. 5.3.2 Dynamic partitioning example

- Four process/programs are loaded into memory. These program starts loading where the OS ends. After loading four programs, it leaves a hole at the end of memory that is too small for a next process.
- Operating system swap out program 2 and leaves sufficient room to load new program into memory. The size of newly loaded program is smaller than the program in memory. Again another small hole is created.

5.3.2 Memory Protection and Mapping

- Memory allocated to the process need to be protected against interference by other processes. Memory protection provides this facility.
- Relocation register and limit register are used for memory protection. Physical address is stored in the relocation register and range of virtual address is stored in the limit register.

- Fig. 5.3.3 shows logical address space.
- Fig. 5.3.4 shows memory protection by using relocation and limit registers.
- Use of registers method for memory protection requires proper arrangement. This method has to be used individually for each memory area allocated to a process.

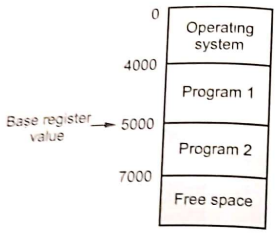


Fig. 5.3.3 Logical address space

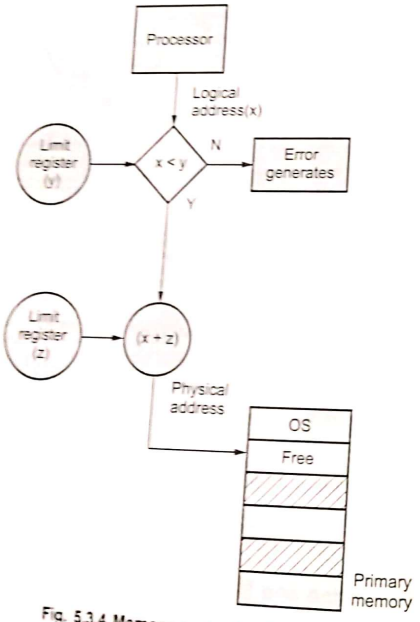


Fig. 5.3.4 Memory protection hardware

- Every logical address generated by the CPU is checked against these registers. protection violation interrupt is raised if any of these checks fail.

- Operating system load a base register and limit register. OS uses privileged instruction for this purpose. Operating system executes in monitor mode and privilege instruction also executes in privilege mode.
- Operating system prevents the user programs from changing the content of the registers. While executing in the monitor mode, operating system is given unrestricted access to both monitor and user's memory.
- In a multiprogramming environment, protection of main memory is essential. Paging or segmentation or both in combination provides an effective means of managing main memory.
- An example of the hardware support that can be provided for memory protection is that of the IBM system/370 family of machines, on which VMS runs. Microsoft Windows 3.1/95 offer memory protection. Microsoft WinNT also offers memory protection. In UNIX, almost impossible to corrupt another process memory.

University Question

1. *Explain the following in brief : Multiprogramming with fixes partitions and Multiprogramming with varibale partitions.*

GTU : Winter-13, Marks 7

5.4 Contiguous Memory Allocation with Variable Partitions

GTU : Summer-14

- The use of unequal size partitions provides a degree of flexibility to fixed partitioning. In dynamic partitioning, the partitions are of variable length and number.
- In noncontiguous memory allocation, a program is divided into blocks that the system may place in nonadjacent slots in main memory.
- This allocation method do not suffer from internal fragmentation, because a process partition is exactly the size of the process.
- Fig. 5.4.1 shows noncontiguous memory allocation method. Operating system maintain the table which contains the memory areas allocated to process and free memory. Memory management unit use this information for allocating processes.
- Table contains information about memory starting address for process/program and their size.
- CPU sends the logical address of the process to the MMU and the MMU uses the

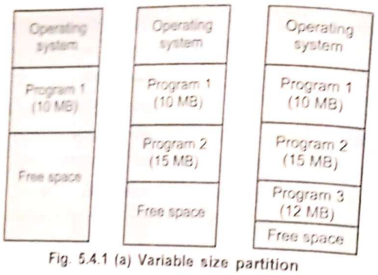


Fig. 5.4.1 (a) Variable size partition

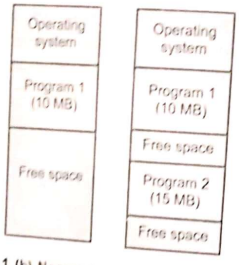


Fig. 5.4.1 (b) Noncontiguous memory allocation

5.4.1 Difference between Contiguous and Noncontiguous Memory Allocation

Sr. No.	Contiguous allocation	Noncontiguous allocation
1.	Program execution take place without overhead.	Address translation is overhead.
2.	Swapped-in processes are placed in the original area.	Swapped-in processes can be placed any where in memory.
3.	Suffer from internal fragmentation.	Only paging suffers from internal fragmentation.
4.	Allocates single area of memory for process.	Allocates more than one block of memory for process.
5.	Wastage of memory.	No wastage of memory.

5.4.2 Fragmentation

- Fragmentation are of two types :
  1. Internal fragmentation.
  2. External fragmentation.
- In fragmentation, OS cannot use certain area of available memory.

Internal fragmentation

- There is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition is called as internal fragmentation.
- Fig. 5.4.2 shows an internal fragmentation.
- To keep track of this free hole is overhead for system.

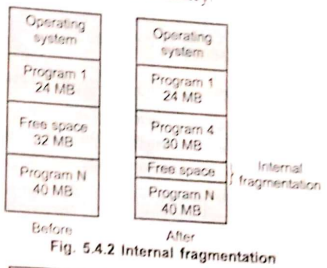


Fig. 5.4.2 Internal fragmentation

External fragmentation

- It occurs when enough total main memory space exists to satisfy a request, but it is not contiguous, storage is fragmentated into a large number of small holes.
- Fig. 5.4.3 shows external fragmentation.

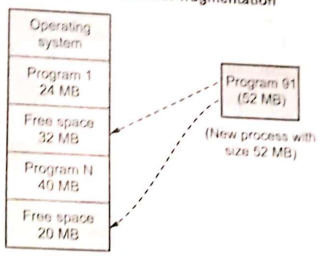


Fig. 5.4.3 External fragmentation

- Following are the solution for external fragmentation :
  1. Compaction
  2. Logical address space of a process/program

5.4.3 Difference between Internal and External Fragmentation

Sr. No.	Internal fragmentation	External fragmentation
1.	Memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation.	External fragmentation exists when there is enough total memory space to satisfy a request, but available spaces are not contiguous.

2.	First-fit and best-fit memory allocation does not suffer from internal fragmentation.	First-fit and best-fit memory allocation from external fragmentation.
3.	Systems with fixed sized allocation units, such as the single partitions scheme and paging suffer from internal fragmentation.	Systems with variable sized allocation such as the multiple partitions scheme segmentation suffer from fragmentation.

5.4.4 Compaction

- Compaction solves problem of external fragmentation.
- Operating system moves all the free holes to one side of main memory and creates large block of free size.
- It must be performed on each new allocation of process to memory or completion of process for memory. System must also maintain relocation information.
- All free blocks are brought together as one large block of free space. Compaction requires dynamic relocation.

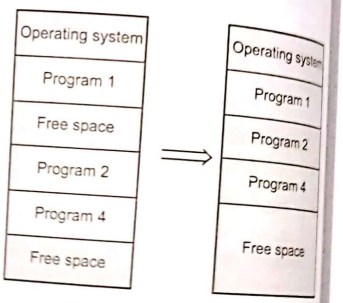


Fig. 5.4.4 Compaction

- Fig. 5.4.4 shows compaction.
- Compaction has a cost and selection of an optimal compaction strategy is di
  One method for compaction is swapping out those processes that are to be m
  within the memory and swapping them into different memory locations.
- Compaction is not always possible. Compaction is time consuming method
  wastage of the CPU time.

Garbage collection

- Some programs use dynamic data structures. These programs dynamically use
  discard memory space. Technically, the deleted data items release me
  locations.
- But, in practice the OS does not collect such free space immediately for allo
  This is because that affects performance. Such areas, therefore are called garbag
- When such garbage exceeds a certain threshold the OS would not have
  memory available for any further allocation.

5.4.5 Placement Algorithms

- In an environment that supports dynamic memory allocation, the memory manager must keep a record of the usage of each allocatable block of memory. This record could be kept by using almost any data structure that implements linked lists.
- An obvious implementation is to define a free list of block descriptors, with each descriptor containing a pointer to the next descriptor, a pointer to the block and the length of the block.
- The memory manager keeps a free list pointer and inserts entries into the list in some order conductive to its allocation strategy. A number of strategies are used to allocate space to the processes that are competing for memory.
- Fig. 5.4.5 shows the placement algorithm.

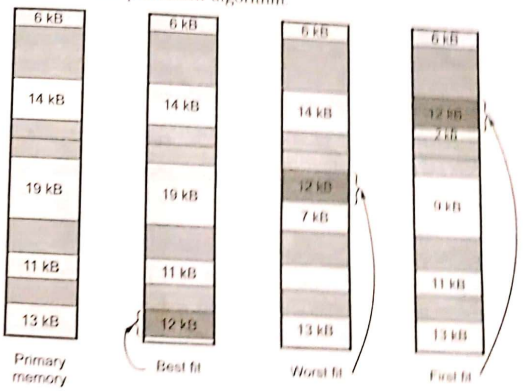


Fig. 5.4.5 Placement algorithm

1. **Best fit** : The allocator places a process in the smallest block of unallocated memory in which it will fit. For example, suppose a process requests 12 kB of memory and the memory manager currently has a list of unallocated blocks of 6 kB, 14 kB, 19 kB, 11 kB and 13 kB blocks. The best-fit strategy will allocate 12 kB of the 13 kB block to the process.

✔ Best fit chooses the block that is closest in the size to the request.

Problems of best fit

- It requires an expensive search of the entire free list to find the best hole.

- More importantly, it leads to the creation of lots of little holes that are not enough to satisfy any requests. This situation is called *fragmentation* and is a problem for all memory-management strategies, although it is particularly bad for best-fit.

Solution : One way to avoid making little holes is to give the client a bigger block than it asked for. For example, we might round all requests up to the next larger multiple of 64 bytes. That doesn't make the fragmentation go away, it just hides it. Unusable space in the form of holes is called *external fragmentation*.

2. **Worst fit :** The memory manager places a process in the largest block of unallocated memory available. The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that compared to best fit ; another process can use the remaining space. Using the same example as above, worst fit will allocate 12 kB of the 19 kB block to the process, leaving a 7 kB block for future use.

3. **First fit :** There may be many holes in the memory, so the operating system, to reduce the amount of time it spends analyzing the available spaces, begins at the start of primary memory and allocates memory from the first hole it encounters large enough to satisfy the request. Using the same example as above, first fit will allocate 12 kB of the 14 kB block to the process.

4. **Next fit :** The first fit approach tends to fragment the blocks near the beginning of the list without considering blocks further down the list. Next fit is a variant of the first-fit strategy. The problem of small holes accumulating is solved with next fit algorithm, which starts each search where the last one left off, wrapping around to the beginning when the end of the list is reached (a form of one-way elevator).

- Notice in the diagram above that the best fit and first fit strategies both leave a tiny segment of memory unallocated just beyond the new process.

Example 5.4.1 *Given memory partitions of 100 K, 500 K, 200 K, 300 K and 600 K (in order) how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 K, 417 K, 112 K and 426 K (in order) ? Which algorithm makes the most efficient use of memory ?* **AU : May-04**

Solution : First-fit :

212 K is put in 500 K partition

417 K is put in 600 K partition

112 K is put in 288 K partition (new partition 288 K = 500 K – 212 K)

426 K must wait.

Best-fit :

212 K is put in 300 K partition

417 K is put in 500 K partition

112 K is put in 200 K partition

426 K is put in 600 K partition

Worst-fit :

212 K is put in 600 K partition

417 K is put in 500 K partition

112 K is put in 388 K partition (600 K – 212 K)

426 K must wait

In this example, Best-fit turns out to be the best.

Example 5.4.2 *Memory partitions with used and unused block are given in the following Fig. 5.4.6. Additional requests 200 K, 100 K and 50 K (in that order) are received. At what starting address will each of the additional requests be allocated for following allocation ?*

i) First fit ii) Best fit.

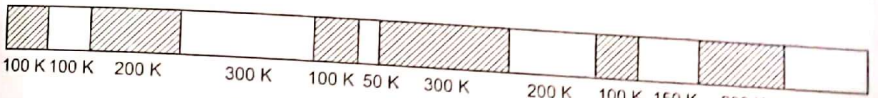


Fig. 5.4.6

Solution : i) First fit :

It searches for the first hole larger than or equal to 200 K. But the first hole (free block) size is 100 K, so it is too small. Next free block (hole) is of size 300 K. Request of 200 K can easily accommodate in second hole. So the first request would be allocated at the start of the second hole.

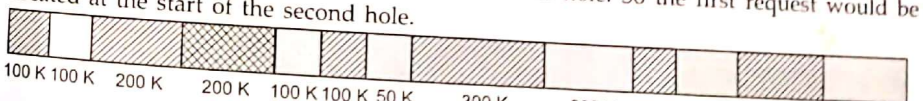


Fig. 5.4.7

After allocating 200 K, memory looks like

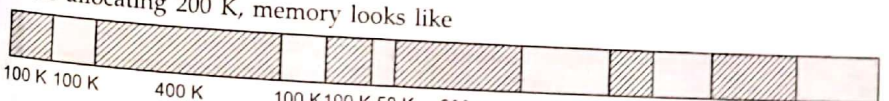


Fig. 5.4.8

For second request of 100 K, it allocates in the first hole of size 100 K. Memory would now look as follows.

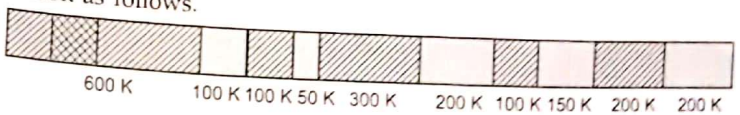


Fig. 5.4.9

For third request of 50 K, memory would now look as follows.

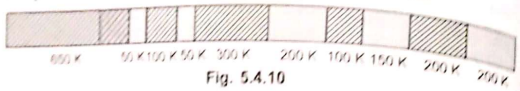


Fig. 5.4.10

iii) **Best fit** : It searches for the smallest hole larger than or equal to 200 K. The hole is exactly 200 K, making it best fit. Memory map is as follows.

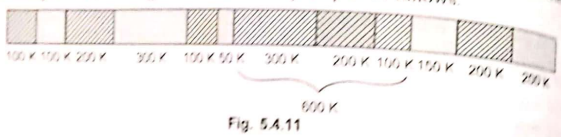


Fig. 5.4.11

Second request of 100 K, first hole is best fit for it. Memory looks like



Fig. 5.4.12

Final request of 50 K would be allocated to the hole of 50 K. The memory looks

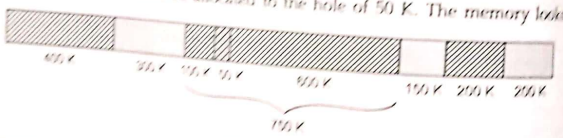


Fig. 5.4.13

University Questions

1. Explain the following allocation algorithms : 1) First-fit 2) Best-fit 3) Worst-fit.
2. What is fragmentation ? What is the need of fragmentation ? Explain the difference between internal and external fragmentation.

GTU : Summer-14, Marks 7

5.5 Paging

GTU : Summer-14, Marks 7

- In paging, operating system divides each incoming programs into pages of equal size. The sections of a disk are called block or sectors. The sections of main memory are called page frames. One sector will hold one page of job instructions and fit into one page frame of memory.

- In paging, logical address space of a program can be noncontiguous. It solves external fragmentation problem.
- The relation between virtual addresses and physical memory addresses given by page table.

- Fixed sized blocks are called frames and breaking of logical memory into blocks of same size called pages.
- Memory manager prepares following things before executing a program :
  1. Find out the number of pages in the program.
  2. Free space in the main memory.
  3. Loading of all the programs pages into memory.
- Pages are not loaded continuously in main memory. Each page can be stored in any available page frame anywhere in main memory. Memory manager keeps the track of pages of program. Paging avoids external fragmentation and need for compaction.
- Memory manager uses three tables to keep track of process and memory.
  1. **Job table** : It stores the size of the active job and memory location where its page table is stored.
  2. **Page map table** : It contains vital information about each page. PMT contains apage number and corresponding page frame memory address. It includes only one entry per page. Page numbers are in sequential order.
  3. **Memory map table** : MMT is used to store page frame location and its status.
- Page size depends upon underlying hardware. Operating system maintains a page table for each process. The page table shows the frame location for each page of a process. Fig. 5.5.1 shows the virtual address format for paging system.

Page number (P)	Displacement or offset (D)

Fig. 5.5.1 Virtual address format

- Processor hardware performs the logical to physical address translation. Logical address contains page number and offset. The physical address contains frame number and offset. Offset is a relative factor. It is used to locate that line within its page frame.
- Suppose a system uses n-bit for representing both physical and virtual address. The page number is represented by the most significant bit (n - m bits) and the displacement is represented by m bit. The table, which holds virtual address to physical address translations is called page table. As displacement is constant, so only translation of virtual page number to physical page is required. Fig. 5.5.2 shows address

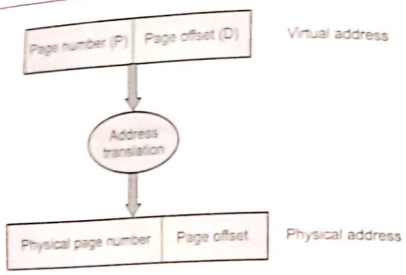


Fig. 5.5.2 Address translation

- Fig. 5.5.3 shows paging hardware. CPU generates logical addresses containing page number and an offset. This page number is used to retrieve the frame number from a page table which gives the base address so the physical address is calculated as base + offset.

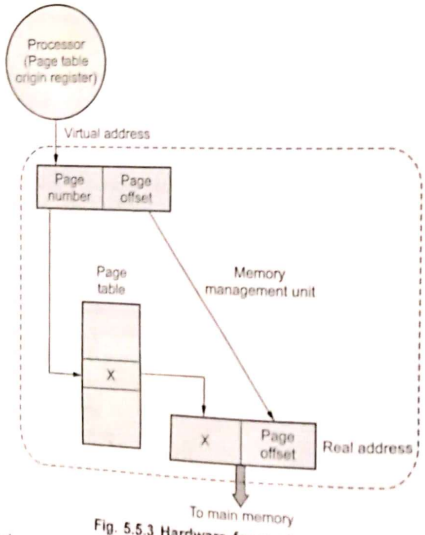


Fig. 5.5.3 Hardware for paging

- Paging implementation requires CPU and operating system support. Page table may itself be resident in main memory.

- Active process is loaded into memory and it occupies number of pages. This set of pages forms its resident set. Hardware defined the page size.
- The processor generates virtual (logical) address and it consists of two parts :
  1. Page number 2. Page offset
- Page table contains page number and index frame number. Page number is used as an index into a page table.
- In case of simple partition, a logical address is the location of a word relative to the beginning of the program the processor translates that into a real address.
- But in the paging method the logical to real address translation is still done by processor hardware. The processor must know how to access the page table of the current process.
- The operating system partitions the memory into areas called **page frames.** Each page frame size is equal. Page size is normally power of 2.
- Fig. 5.5.4 shows a paging model. Operating system maintains the free frame list. During execution of process the memory management unit consults the page table to perform address translation.

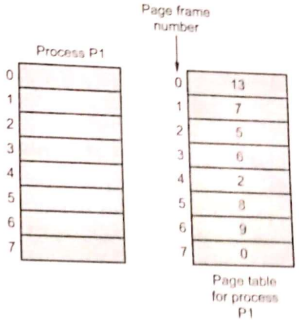


Fig. 5.5.4 Paging model

- The physical address is the portion of the primary memory allocated to the process. The page frames allocated to the process need not be contiguous.
- The aim of the paging system is to identify the set of pages needed for the process current locality and then to load only those pages into page frames in primary memory.
- The logic of the address translation process in paged system is shown in the following Fig. 5.5.5. For example, the virtual address is 03200H. This virtual address is split by hardware into the page number and offset within that page. High order 12 bit is used as page number i.e. 003H and lower order 12 bit is used for the offset i.e. (200H).

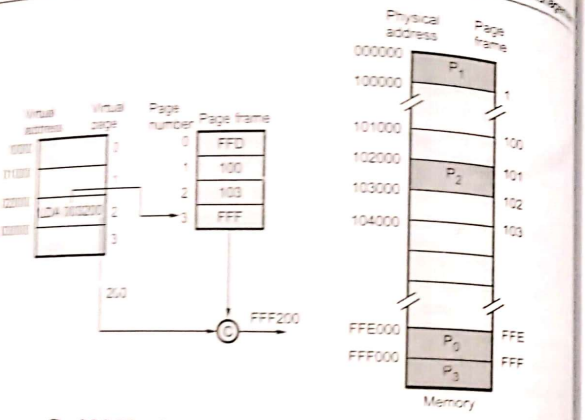


Fig. 5.5.5 (Virtual address to physical address) Paging

- Page number is used to index the page table and to obtain the corresponding physical frame number (FFFH). This value is then concatenated with the offset to produce the physical address (FFF200H) which is used to reference the target item in memory.
- The operating system keeps track of the status of each page frame by means of a physical memory map that may be structured as a static table. **The logical address space may be the same size as the physical address space or it may be smaller.**
- If logical address space is small, it prevents one process from monopolizing all the memory. If the logical address space is larger than the physical address space, all pages could not be resident in physical memory. Simple paging has no capability for dealing with references to pages that are not in memory.

5.5.1 Protection and Sharing

- Protection bit is used for provide protection to the memory. Page table adds one more column for providing protection bit.
- One bit field define a page to be read only or read-write operation. This field is read at the time of calculating physical address of memory.
- Fig. 5.5.6 shows protection bit with page table. (See Fig. 5.5.6 on next page)

Index	Frame number	Protection bit
0	4	Valid
1	2	Invalid
2	5	Valid
3	7	Valid
4	1	Valid
5	3	Invalid



Fig. 5.5.6 Protection bit

- When valid bit is set, the required page is in the process logical address space and the page is valid. If invalid bit is set, the page is not in the process logical address space.

5.5.2 Paging with TLB

- Set of registers are used for implementing page table. Registers are suitable only for small page table. If page table size increases then special purpose hardware cache is used for page table.
- Special cache memory called a Translation Lookaside Buffer (TLB) is used with the address translation hardware. The full page table is kept in primary memory.
- TLB is very effective in improving the performance of page frame access. The cache buffer is implemented in a technology which is faster than the primary memory technology.
- Fig. 5.5.7 shows paging with TLB.
- The TLB contents may be controlled by the operating system or by hardware, depending on the architecture.
- When a page is first translated to a page frame, the map is read from primary memory into the TLB. The TLB entry contains the page number, page frame's physical address etc.

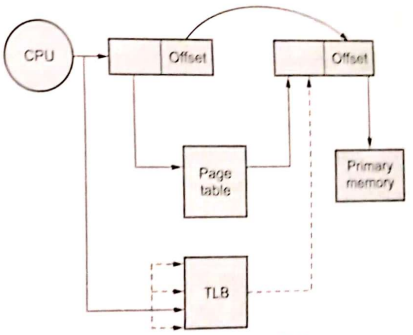


Fig. 5.5.7 Paging with TLB

- The page mapping mechanism first tries to find page number in the TLB. If the TLB contains the page number, then it is called as **page hit** or **TLB hit.**
- If the TLB does not contain an entry for page p, then it called as page miss or TLB miss. In case of TLB miss, the operating system locates the page table entry in the primary memory, which increases execution time.
- The TLB is associative, high-speed memory. A translation buffer is used to store a few of the translation table entries. It is very fast, but only for a small number of entries. On each memory reference.
  1. First ask TLB if it knows about the page. If so, the reference proceeds fast.
  2. If TLB has no information for page, must go through page and segment table to get information. Reference takes a long time, but gives the info for this page to TLB so it will know it for next reference.
- The greatest performance improvement is achieved when the associative memory is significantly faster than the normal page table lookup and the hit ratio is high. The hit ratio is the ratio between accesses that find a match in the associative memory and those that do not.
- **Disadvantage of TLB is that :** If two pages use the same entry of the memory, only one of them can be remembered at once. If process is referencing both pages at same time, TLB does not work very well.
- Each entry in the TLB must include the page number as well as the complete page table entry. The processor is equipped with hardware that allows it to simultaneously interrogate a number of TLB entries to determine if there is a match on page number. This technique is referred to as *associative mapping*.

5.5.3 Page Table Structure

5.5.3.1 Multilevel or Hierarchical Page Table

- Page tables can consume a significant amount of memory.
- For example : A 32-bit virtual address space using 4 kB pages.

  Page size = 2^{12} bytes

  Space for page numbers = $2^{32} - 2^{12}$

  = 2^{20} bytes
- So page table may consists of upto 1 million entries, one for each page, for a total address capacity of about four billion bytes.
- Hierarchical page table is also called as forward mapped page table. This approach is so effective that many modern operating systems employ it.

- In this method, each level containing a table that stores pointers to tables in the level below. Each table in the hierarchy is the size of one page. It enables the operating system to transfer page tables between main memory and secondary storage device easily.

For two levels of page table

- Virtual address contains page number and displacement into that page.

  Virtual address (v) = (p, t, d)

  where (p, t) = Page number

  d = Displacement
- Here p is an index into the outer page table.
- Fig. 5.5.8 shows hierarchical page table.
- Intel IA - 32 architecture support two levels of page tables.

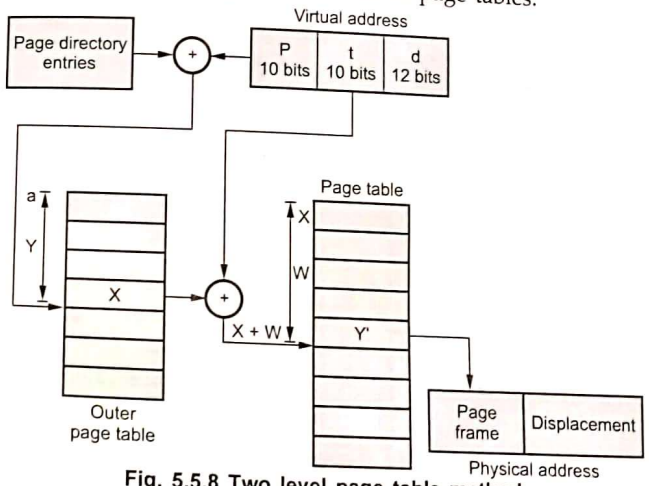


Fig. 5.5.8 Two level page table method

Advantages

1. It is compact and support sparse address space.
2. It easily manage the memory.
3. It reduce table fragmentation.

Disadvantages

1. Overhead due to one more page table.
2. On TLB miss, two loads from memory will be required to get the right address translation information from the page table.

5.5.5 Hashed Page Tables

- Hashed page table support address space of 32 bits and more. Hash value is same as virtual page number.
- Hash table contains a link list of elements. Each elements consists of following fields :
  1. Virtual page number
  2. Mapped page frame value.
  3. Pointer to the next element.
- Hash table improves search speed.
- Fig. 5.5.9 shows block diagram of hashed page table.

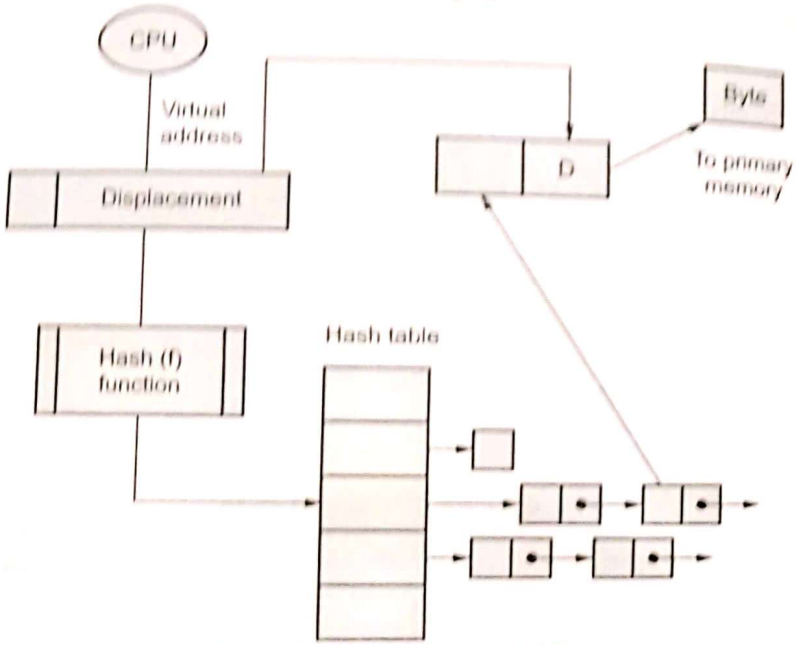


Fig. 5.5.9 Hashed page table

rking

1. Virtual page number is taken from virtual address.
2. Virtual page number is hashed into the hash table.
3. Virtual page number is compared with the first element of linked list.
4. Both the value is matched, that value is (i.e. page frame) used for calculating physical address.
5. If the both value is not matched, the entire linked list is searched for a matching
6. Clustered page table is same as hashed page table but only difference is that entry in the hash table refers to several pages.

5.5.6 Inverted Page Table

- It uses a page table that contains an entry for each physical frame. This means that the page table occupies a fixed fraction of memory. The size is proportional to the physical memory.
- Page table overhead increases with address space size. Page table get too big to fit in memory.
- Inverted page table has one entry for each real page of memory. Lookup time is increased because it requires a search on the inverted table.
- Logical address is as follows :

Process id	Page number	Offset

- Inverted page table is used by Itanium, Power PC and Ultra SPARC.

5.5.7 Advantages of Paging

1. Paging eliminates fragmentation.
2. Support higher degree of multiprogramming.
3. Paging increases memory and processor utilization.
4. Compaction overhead required for the relocatable partition scheme is also eliminated.

5.5.8 Disadvantages of Paging

1. Page address mapping hardware usually increases the cost of the computer.
2. Memory must be used to store the various tables like page table, memory map table etc.
3. Some memory will still be unused if the number of available block is not sufficient for the address spaces of the jobs to be run.

5.6 Segmentation

GTU : Summer-18

- In segmentation, a program's data and instructions are divided into blocks called **segments.** A segment is a logical entity in a program.
- **Logical view :** A process consists of a set of segments.
- **Physical view :** It consists of non-adjacent areas of memory allocated to segments.
- All segment size may be equal or may not be equal. Segmentation support user view of memory. Fig. 5.6.1 shows programmer's view of a program.

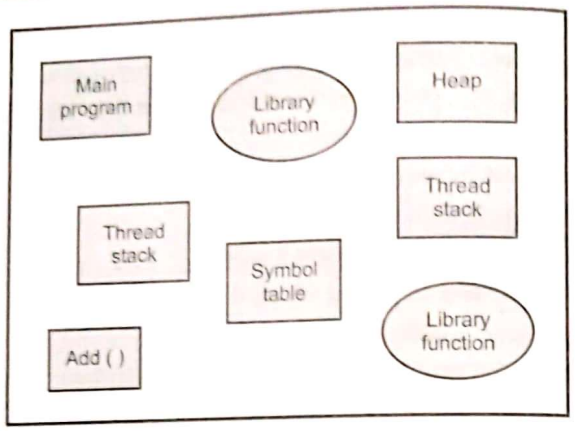


Fig. 5.6.1 Programmer's view

- Segmentation can be implemented with or without paging.
- Collection of segment is called as logical address space. Each segment is identified by its name.
- Processor generates logical addresses. These addresses consists of a segment number and an offset into the segment. Segment number is used as an index to segment page table.

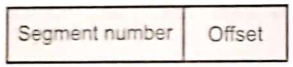


Fig. 5.6.2 Logical address

- Segment names are normally symbolic names.
- Operating system maintains a segment table for each process. It is usually stored in main memory as a segment that is not to be loaded as long as the process can run.
- Each entry in the segment table has a segment base and a segment limit. The base field contains the segment relocation register for the target segment. The segment limit field contains the length of the segment.
- Fig. 5.6.3 shows an address translation in segmentation.
(See Fig. 5.6.3 on next page)
- Segment table contains the physical address of the start of the segment. Then add the offset to the base and generate the physical address.
- If the required reference is not found in one of the segment registers, then error is generated. At the same time, operating system does lookup in segment table and loads new segment descriptor into the register.

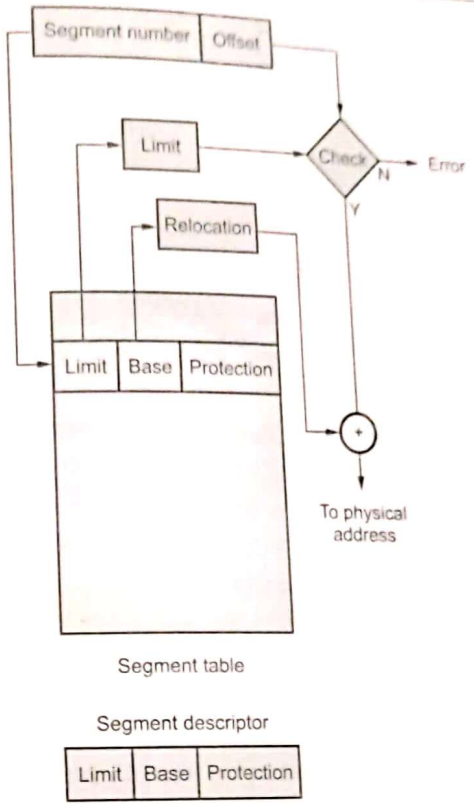


Fig. 5.6.3 Address translation in segmentation

5.6.1 Protection and Sharing

- Sharing of segment provides less overhead than the pure paging segment. Multiple processes can share a segment.
- Two processes share a segment when their segment table entries point to the same segment in main memory. Operating system maintains global segment table for everyone and also used by OS.
- Access rights for segment are usually included in table entry. An illegal access of memory is protected by memory mapping hardware. It checks the protection bits associated with each segment table.

- Fig. 5.6.4 shows sharing in segmentation. Access rights for segments are : Read, write, append and execute.
- Segmentation suffer from external fragmentation because segment sizes vary. Entire segment is either memory or on secondary storage disk.

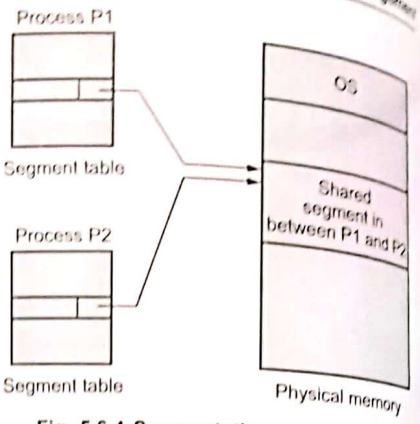


Fig. 5.6.4 Segmentation with sharing

5.6.2 Advantages

1. It provides virtual memory.
2. Allows dynamic segment growth.
3. Segmentation assists dynamic linking.
4. Segmentation is visible.

5.6.3 Disadvantages

1. Maximum size of a segment is limited by the size of main memory.
2. Difficulty to manage variable size segments on secondary storage.
3. Fragmentation and complicated memory management.

5.6.4 Difference between Segmentation and Paging

Sr. No.	Segmentation	Paging
1.	Program is divided into variable size segments.	Program is divided into fixed size pages.
2.	User (or compiler) is responsible for dividing the program into segments.	Division into pages is performed by the operating system.
3.	Segmentation is slower than paging.	Paging is faster than segmentation.
4.	Segmentation is visible to the user.	Paging is invisible to the user.
5.	Segmentation eliminates internal fragmentation.	Paging suffers from internal fragmentation.

6.	Segmentation suffers from external fragmentation.	There is no external fragmentation.
7.	Processor uses page number, offset to calculate absolute address.	Processor uses segment number, offset to calculate absolute address.
8.	OS maintain a list of free holes in main memory.	OS must maintain a free frame list.

University Question

1. *Define and explain following terms : Segmentation.* **GTU : Summer-13, Marks 2**

5.7 Paged Segmentation

- Most of the architecture support paging and segmentation. All the pages of segment need not be in main memory.
- It simplify the memory allocation and speed increases. It requires a high speed register to store the base address of the segment map table.
- For each segment, page table is created by OS. A pointer to the page table is kept in the segment's entry in the segment table.
- Segments are typically larger than pages.
- The base address get from the segment descriptor table is concatenated with the offset. This new address is referred to as a **linear address.** Linear address is generated by paging hardware.

Advantages

Combines all advantages of paging and segmentation.

Disadvantages

1. It increases hardware cost. 2. It increases processor overheads.
3. Dangers of thrashing.

Example 5.7.1 *Paging system consists of physical memory 2^{24} bytes, pages of logical address space is 256. Page size of 2^{10} bytes, how many bits are in a logical address.*

Solution : Logical address space = $256 = 2^8$

Page size = 2^{10} bytes

So, Total logical address space = $2^8 \times 2^{10} = 2^{18}$ bytes

For 2^{18} byte address space - 18-bit address is required.

Example 5.7.2 *On a system using simple segmentation, compute the physical address for each of the logical addresses, logical address is given in the following segment table. If the address generates a segment fault, indicate so.*

Segment	Base	Length
0	330	124
1	876	211
2	111	99
3	498	302

a) 0, 99 b) 2, 78 c) 1, 265 d) 3, 222 e) 0, 111

Solution : **a) 0, 99**

Offset = 99

Segment length = 124

Segment = 0

* Offset 99 is less than segment length 124.

* Starting location of segment 0 is start from 330.

* Physical address = Offset + Segment base

= 99 + 330 = 429

b) 2, 78

Segment = 2

Offset = 78

Segment length = 99

* Offset 78 is less than segment length 99

* Starting location of segment 2 is 111

* Physical address = Segment base + Offset

= 111 + 78 = 189

c) 1, 265

Segment = 1

Offset = 265

Segment length = 211

* Offset 265 is greater than segment length 211.

* This address results in a segment fault.

d) 3, 222

Segment = 3

Offset = 222

Segment length = 302

* Offset 222 is less than segment length 302.

* Starting location of segment 3 is 498

* Physical address = Segment base + Offset

= 498 + 222 = 720

e) 0, 111

Segment = 0

Offset = 111

Segment length = 124

* Offset 111 is less than segment length 124.

* Starting address of segment 0 is 330

* Physical address = Segment base + Offset

= 330 + 111 = 441

Example 5.7.3 *System using a paging and segmentation, the virtual address space consists of upto 8 segments where each segment can be up to 2^{29} byte long. The hardware pages each segment into 256 bytes pages. How many bits in the virtual address specify the*

i) Segment number

ii) Page number

iii) Offset within page

iv) Entire virtual address

Solution : **i) Segment number :** Virtual address space consists of upto 8 segments. So

$$8 = 2^3$$

$\therefore$ 3 bits are needed to specify segment number.

ii) Page number : Hardware pages each segment into 256 byte pages. So

$$256 = 2^8 \text{ byte pages}$$

Size of segment is 2^{29} bytes.

$\therefore$ $2^{29}/2^8 = 2^{29-8} = 2^{21} = 21$ pages

$\therefore$ 21 bits are required to specify the page number.

iii) Offset within the page : For 2^8 byte page, 8 bits are needed.

iv) Entire virtual address :

= Segment number + Page number + Offset

= 3 + 21 + 8

= 32

5.8 Virtual Memory Concepts

GTU : Winter-11

- Virtual memory is a method of using hard disk space to provide extra memory. It simulates additional main memory. In Windows operating system, the amount of virtual memory available equals the amount of free main memory plus the amount of disk space allocated to the swap file. Fig. 5.8.1 shows logical view of virtual memory concept.

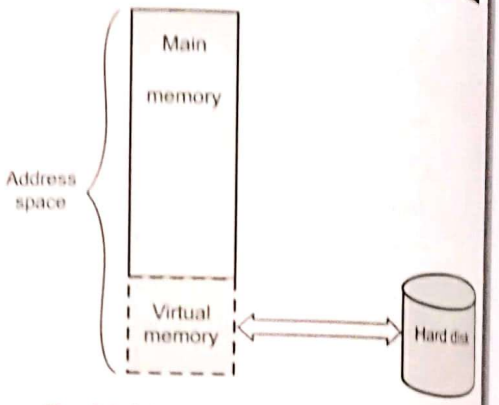


Fig. 5.8.1 Logical view of virtual memory

- A swap file is an area of your hard disk that is set aside for virtual memory. Swap files can be either temporary or permanent.
- Virtual memory is stored in the secondary storage device. It helps to extend additional memory capacity and work with primary memory to load applications. The virtual memory will reduce the cost of expanding the capacity of physical memory. The implementations of virtual memory will different for operating systems to operating system.
- Each process address space is partitioned into parts that can be loaded into primary memory when they are needed and written back to secondary storage otherwise. Address space partitions have been used for the code, data and stack identified by the compiler and relocating hardware. The portion of the process that is actually in main memory at any time is defined to be the **resident set** of the process.
- The logical addressable space is referred to as virtual memory. The virtual address space is much larger than the physical primary memory in a computer system. The virtual memory works with the help of secondary storage device and its speed is low compared to the physical storage location.

- Virtual memory uses two types of addresses : ***Virtual address and physical address.***
- Virtual address is referred by process and physical address is actual address of the main memory.
- Whenever a process accesses a virtual address, the system must translate it to a physical address. Virtual memory system uses special purpose hardware. This hardware is called as memory management unit.
- Most virtual memory system use a technique called paging. The virtual address space is divided into fixed size units called pages. The corresponding units in the physical memory are called page frames. The pages and page frames are normally the same size. The area on a hard disk that stores page frames is usually called the paging file or the swap file.
- Fig. 5.8.2 shows concept of virtual memory.

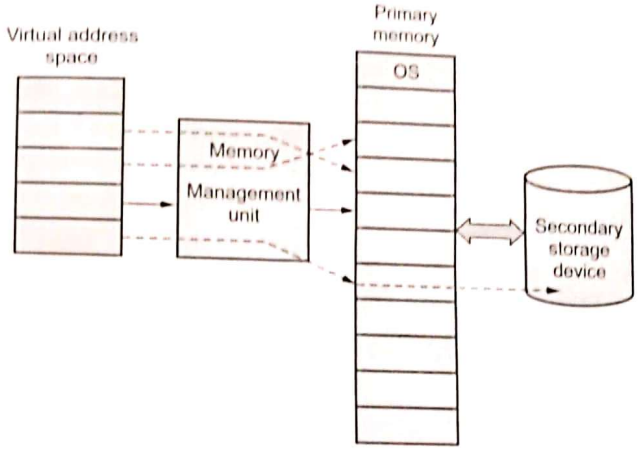


Fig. 5.8.2 Concept of virtual memory

- When the system is ready to run a process, the system loads the process code and data from secondary storage into primary memory. Only a small portion of these needs to be in primary memory at once for the process to execute. The success of implementing virtual memory system is how to map virtual addresses to physical addresses. Because processes reference only virtual addresses.
- Virtual memory can be implemen in one of two ways :
  1. Paging
  2. Segmentation

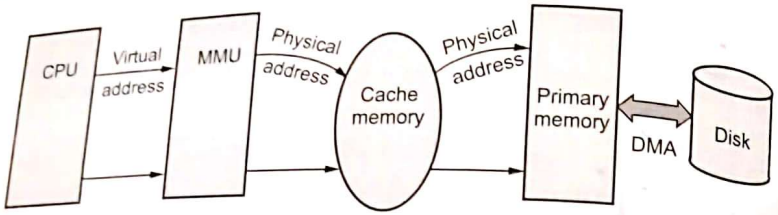


Fig. 5.8.3 Virtual memory organization

- Following are the situations, when entire program is not required to load fully.
  1. User written error handling routines are used only when an error occurs in the data or computation.
  2. Certain options and features of a program may be used rarely.
  3. Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
  4. Many routines are commonly used at mutually exclusive times during a run.
- Virtual memory makes the task of programming much easier. Virtual memory is commonly implemented by **demand paging**. Virtual memory mechanism bridges the size and speed gaps between the main memory and secondary storage and is usually implemented in part by software technique.

Benefits

1. Less I/O needed to load/swap a process.
2. Degree of multiprogramming can be increased.
3. A program's logical address space can be larger than physical memory.

University Question

1. *Explain virtual memory management with paging in detail. Explain how virtual address is translated into physical address. Also compare paging with segmentation.*

GTU : Winter-13, Marks 7

5.9 Demand Paging

- In demand paging, a page is loaded into primary memory only when the process references it. It is the simplest fetch policy implemented in virtual memory.
- Demand paging guarantees that the system brings into primary memory only those pages that processes actually need. It involves interaction between hardware and software component of the virtual memory.

- Paging with swapping method will become as demand paging concept. Process's logical address space is stored on the secondary storage device.
- Demand paging requires that the page map table for each process keep track of each page as it is loaded or removed from primary memory.
- When user want to execute a process, user swap a process into a memory. **Lazy swapper** is used for swapping the process into memory. Lazy swapper swaps the pages whenever needed.

Working of demand paging

1. Program attempts to find a page.
2. If the page is located in the primary memory, then the program executes.
3. If the page cannot be found in the primary memory, then the page fault occurs.
4. Memory reference is checked for required page to determine it is a valid reference to a location on a secondary storage, if it is then the page will be required page. If not, the process is terminated.
5. Schedule disk operation to read the required page into primary memory.
6. Restart the instruction that was interrupted by the OS trap.

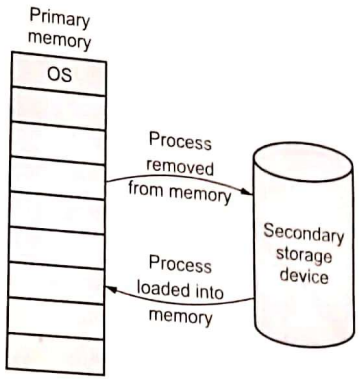


Fig. 5.9.1 Demand paging

- Fig. 5.9.1 shows the concept of demand paging.
- Demand paging takes support from hardware to distinguish the pages. The valid-invalid bit method is used. This method gives the idea about page location. Page may be in the primary memory or in the secondary storage.

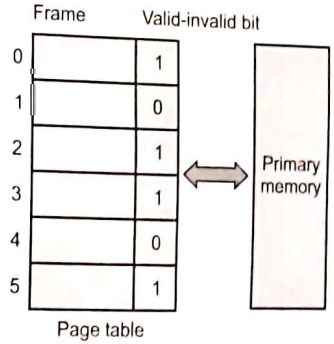


Fig. 5.9.2 Page table entry with valid-invalid

- If valid bit is set, then required page is stored in the primary memory. When an invalid bit is set, then page is on the secondary storage disk.
- Valid-invalid bit field is attached with page table. Fig. 5.9.2 shows page table entry.

- The valid-invalid bit field contain a boolean value to indicate whether the page exists in primary memory.
  0 = Not in memory
  1 = In memory
- Initially all valid-invalid entries are set to 0. This information is used for page replacement algorithm. Page fault leads to the page replacement operation to load the required pages in primary memory.
- A failure to find a page in memory is often called a page fault.
- Demand paging is most efficient when users are aware of the page size used by operating system.

5.9.1 Steps in Handling a Page Fault

- When page fault occurs, the system performs following steps :
  1. Check the process control block table for reference which bit is set i.e. valid bit or invalid bit for memory access.
  2. If invalid bit is set, user will terminate the process. If valid bit is set, then the page is loaded into memory.
  3. Check for free frame for new **page loading.**
  4. Start reading secondary storage device for desired page for allocated frame.
  5. Required page is loaded into the memory after completion of read operation PCB table is updated.
  6. System restart the instruction that was interrupted by the error. Page is loaded in memory so process can use it.
- Fig. 5.9.3 shows page fault handling steps in demand paging.

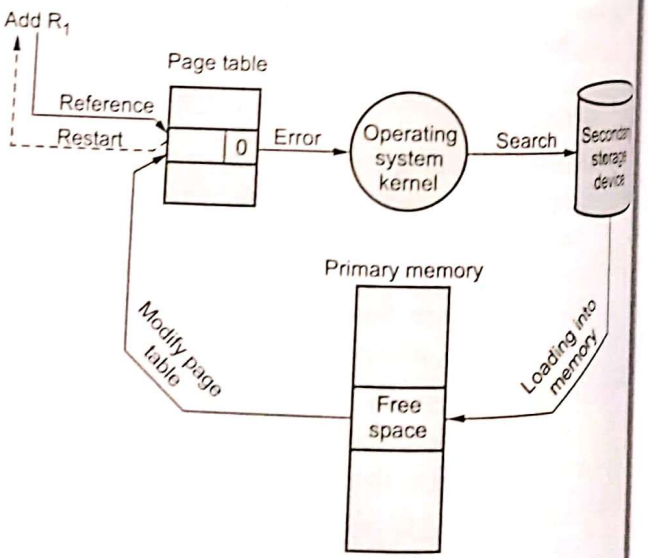


Fig. 5.9.3 Handling of page fault

5.9.2 Demand Paging Performance

- Demand paging is a solution to inefficient memory utilization. Performance of computer system is affected by demand paging. Effective access time for demand paged memory is calculated as follows :
- Since demand paging like caching, can compute average access time. Memory access time (m) for most computers now ranges from 10 to 200 nanoseconds if there is no page fault, then

  Effective access time = Memory access time
- If there is page fault, then

  Effective access time = Hit rate × Hit time + Miss rate × Miss time
  = Hit time + Miss rate × Miss penalty
  Effective access time = Memory access time * (1 – p) + p * page fault service time

where p is the probability of a page fault

- Effective access time is directly proportional to page fault rate.
- Suppose memory access time = 100 ns, Page fault service time = 25 ms then
  Effective access time = Memory access time * (1 – p) + p * page fault service time
  = 100 (1 – p) + p * 25,000,000
  = 100 + 24,999,900 p

5.9.3 Advantages of Demand Paging

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

5.9.4 Disadvantages of Demand Paging

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. Due to the lack of an explicit constraints on a jobs address space size.

5.9.5 Comparison of Demand Paging with Segmentation

Sr. No.	Segmentation	Demand paging
1.	Segments may of different size.	Pages are of same size.
2.	Segment can be shared.	Pages can not be shared.

3.	It allows dynamic growth of segments.	Page size is fixed.
4.	Segment map table indicates the address of each segment in memory.	Page map table keeps track of pages in memory.
5.	Segments are allocated to the program while compilation.	Pages are loaded in memory on demand.
6.	Provides virtual memory.	Also provides virtual memory.

5.10 Page Replacement Algorithms

GTU : Summer-13, Winter-14

- When a page fault occurs, page replacement algorithms are used for loading the page in memory and no free page frame exist in memory.
- Page fault occurs if a running process references a nonresident page.
- The goal of a replacement strategy is to minimize the fault rate. To evaluate a replacement algorithm, following parameters are used :
  1. The size of a page 2. A set of reference strings 3. The number of page frames.
- Given these inputs, we can determine the number of page faults and hence the page-fault rate. A reference string is a list of pages in order of their reference by the processor.
- When we find that a page frame reference is in main memory then we have a page hit and when page fault occurs we say it is page miss. A poor choice of page replacement algorithm may result in lot of page misses.
- Page replacement algorithm deals with how the kernel decides which page reclaim. Operating systems selects the local or global page replacement policy. Local replacement policy allocates a certain number of pages to each process. If a process needs new page, it must replace one of its own pages. If the OS uses a global policy, it can take a page from any process, using global selection criteria.
- UNIX OS uses global page replacement policy.

5.10.1 First-In-First-Out

- FIFO page replacement algorithm removes the pages that have been in memory the longest. Here system keeps track of the order in which pages enter primary memory.
- When page must be replaced, the oldest page is selected. System maintain a FIFO queue of pages. Pages are inserted at the tail of the queue.
- FIFO algorithms interfere with the locality of the process.
- ider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9
- rame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	4	4	4	4	4	4	6	6	6	6	9	9	9	9	9	4	4
Page frame 2		2	2	2	2	5	5	5	5	5	7	7	7	7	7	7	2	2	2	2
Page frame 3			3	3	3	3	3	3	2	2	2	8	8	8	8	8	8	5	5	9
Page fault	PF	PF	PF	PF		PF			PF	PF	PF	PF		PF			PF	PF	PF	PF

Total number of page fault = 14

Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	1	5	5	5	5	5	5	5	5	9	9	9	9	9	9	9
Page frame 2		2	2	2	2	2	2	2	2	6	6	6	6	6	6	6	2	2	2	2
Page frame 3			3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	5	5	5
Page frame 4				4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	4	4
Page fault	PF	PF	PF	PF		PF			PF	PF	PF		PF			PF	PF	PF		

Total number of page fault = 12

5.10.1.1 Belady's Anomaly

- Using FIFO algorithm for some reference strings actually generate more p faults when more page frames are allotted. This truly unexpected result was demonstrated by Belady in 1970 and is known as Belady's anomaly.
- FIFO does not satisfy the inclusion property and is not a stack algorithm.
- A page replacement algorithm that satisfies the inclusion property is free Belady anomaly.
- Consider the reference string : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- Page frame size = 3

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Page frame 2	-	2	2	2	1	1	1	1	1	3	3	3
Page frame 3	-	-	3	3	3	2	2	2	2	2	4	4
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	

Total number of page fault = 9

Page frame size = 4

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	1	1	1	5	5	5	5	4	
Page frame 2	-	2	2	2	2	2	2	1	1	1	1	4
Page frame 3	-	-	3	3	3	3	3	3	2	2	2	5
Page frame 4	-	-	-	4	4	4	4	4	4	3	3	2
Page fault	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF

Total number of page fault = 10

- Paging algorithm has worse performance when the amount of primary memory allocated to the process is increased. The problem creates because the set of pages loaded with a memory allocation of 3 is not same as the with a memory allocation of 4.
- There are a set of demand paging algorithms whereby the set of pages loaded with an allocation of m frames is always a subset of the set of pages loaded with an allocation of m + 1 frame. This property is called the inclusion property.
- FIFO does not satisfy the inclusion property and is not a stack algorithm. LRU can be a stack algorithm.
- FIFO algorithm uses only the modified and status bits when swapping is used.

Advantage of FIFO :

1. FIFO is very simple and easy to implement.

Disadvantages of FIFO :

1. It suffers from Belady's anomaly.
2. It is not very effective.
3. System needs to keep track of each frame.

5.10.2 LRU Page Replacement Algorithm

- LRU stands for least recently used. This replacement policy chooses to replace the page which has not been referenced for the longest time. This policy assumes that recent past will approximate the immediate future. The operating system keeps track of when each page was referenced by recording the time of reference or by maintaining a stack of references.

- System must maintain a time stamp for every page that indicates the time of the last access. When deciding which page to remove, the operating system must scan all the pages of memory for finding the oldest time stamp. Instead of keeping an extra time stamp bits, LRU is implement by maintaining pages in an ordered list.
- System can manage LRU with a list called the LRU stack or the paging stack. In the LRU stack, the first entry describes the page referenced least recently, the last entry describes to the last page referenced. If a page is referenced, move it to the end of the list.
- LRU algorithm does not interfere with the locality of the process.
- Consider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9
- Page frame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	
Page frame 1	1	1	1	4	4	4	3	3	3	6	6	6	6	9	9	9	2	2	2	
Page frame 2	-	2	2	2	2	2	2	4	4	4	7	7	7	7	7	7	7	5	5	
Page frame 3	-	-	3	3	3	5	5	5	2	2	2	8	8	8	8	8	8	8	4	
Page fault	PF	PF	PF	PF		PF	PF	PF	PF	PF	PF	PF		PF			PF	PF	PF	

Total number of page fault = 16

- Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	
Page frame 1	1	1	1	1	1	5	5	5	5	6	6	6	6	6	6	6	2	2	2	
Page frame 2	-	2	2	2	2	2	2	2	2	2	2	2	2	9	9	9	9	5	5	
Page frame 3	-	-	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	4	
Page frame 4	-	-	-	4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	
Page fault	PF	PF	PF	PF		PF			PF	PF	PF		PF			PF	PF	PF	PF	

Total number of page fault = 13

- LRU is a stack algorithm. Increase in memory size will never cause an increase the number of page interrupts.

Advantages :

1. It is quite good algorithm.
2. It is amenable to full statistical analysis.

3. Never suffers from Belady's anomaly.

Disadvantages :

1. Problem is how to implement.
2. It requires hardware assistance.

5.10.3 LRU Approximation Algorithms

- LRU is a desirable algorithm to use, but it is expensive to implement directly. It is often approximated. By using one reference bit, user can do a second-chance or clock replacement algorithm.
- Page frames are arranged in circular list. Initially each reference bit is set to 0. It is set to 1 any time the page is referenced. Algorithm uses pointer to points to the next candidate for a page replacement.
- When a page replacement is needed, the frame pointed to is considered as a candidate. If its reference bit is 0, it is selected. If its bit is 1, the bit is set to 0, and the pointer is incremented to examine the next frame. This continues until a frame is found with a reference bit of 0. This may require that all frames be examined bringing us around to the one we started with.

Second chance replacement

- In some books, the second chance replacement policy is called the clock replacement policy. In the second chance page replacement policy, the candidate pages for removal are consider in a round robin matter and a page that has been accessed between consecutive considerations will not be replaced.
- The page replaced is the one that, considered in a round robin matter. It has not been accessed since its last consideration.
- Working :
  1. Add a "second chance" bit to each memory frame.
  2. Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
  3. A new page read into a memory frame has the second chance bit set to ZERO (0)
  4. When you need to find a page for removal, look in a round robin manner in the memory frames :
     i. If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
     ii. If the second chance bit is ZERO, replace the page in that memory frame

String	0	4	1	4	2	4	3	4	2	4	0	4	1	4	2	4	3	4
Frame 1	0	0	0	0	0	0	0	0	2	0	2	0	2	0	2	1	2	1	2	0	2	0	2	0	1	0	1	0	1	0	1	0	3	0	3
Frame 2	-		4	0	4	0	4	1	4	1	4	0	4	1	4	1	4	1	4	0	4	1	4	1	4	1	4	0	4	1	4	1	4	1	4
Frame 3	-		-		1	0	1	0	1	0	1	0	3	0	3	0	3	0	3	0	0	0	0	0	0	0	0	0	2	0	2	0	2	0	2
PF	PF		PF		PF			PF			PF			PF							PF			PF			PF			PF				

Total number of page fault = 9

5.10.4 Optimal Page Replacement

- The data which will not be accessed permanently or for a longest time should be replaced in optimal page replacement algorithm.
- Optimal page replacement algorithm gives less page fault as compared to FIFO and LRU. This algorithm never suffer from Belady's anomaly.
- Use of this page-replacement algorithm guarantees the lowest possible page-fault rate for a fixed number of pages
- The key distinction between FIFO and optimal is that FIFO uses the time when a page was brought into memory whereas optimal uses the time when a page is to be used.
- It is difficult to implement because of future reference.
- Consider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9
- Page frame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	4	4	4	4	4	6	7	7	7	7	7	7	2	2	4	4
Page frame 2		2	2	2	2	5	5	5	5	5	5	8	8	8	8	8	8	5	5	5
Page frame 3			3	3	3	3	3	3	2	2	2	2	2	9	9	9	9	9	9	9
Page fault	PF	PF	PF	PF		PF			PF	PF	PF	PF		PF			PF	PF	PF	

Total number of page fault = 13

- Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	1	5	5	5	5	5	5	5	5	9	9	9	9	9	9	9

Page frame 2	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	5	5	5	
Page frame 3	-	-	1	3	1	3	3	3	3	6	7	7	7	7	7	7	7	7	7	
Page frame 4	-	-	-	4	4	4	4	4	4	4	4	8	8	8	8	8	8	4	4	
Page fault	PF	PF	PF	PF		PF				PF	PF	PF		PF				PF	PF	

Total number of page fault = 11

Advantages :

1. It has lowest page fault rate.
2. Optimal never suffer from Belady's anomaly.
3. Used for comparison studies.

Disadvantages :

1. Difficult to implement.
2. It requires future reference string.

5.10.5 Difference between FIFO, LRU and Optimal

Sr. No.	FIFO	LRU	Optimal
1.	Number of page fault is more than LRU and optimal.	Number of page fault is more than optimal and less than FIFO.	Number of page fault is less both.
2.	Suffer from Belady's anomaly.	Does not suffer from Belady's anomaly.	Does not suffer from Belady's anomaly.
3.	Used in OS.	Used in OS.	Not used in OS.
4.	Future predication is not required.	Future predication is not required.	Future predication is required.
5.	Simple to implement.	Considered to be good.	Difficult to implement.

Example 5.10.1 *Consider the string 0, 1, 2, 3, 0, 1, 2, 3, 0 ,1 2, 3, 4, 5, 6, 7. With frame size 3 and 4. Calculate page fault.*

Solution : 1) FIFO

Page frame size = 3

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	3	3	3	2	2	2	1	1	1	4	4	4	7
1		1	1	1	0	0	0	3	3	3	2	2	2	5	5	5

2			2	2	2	1	1	1	0	0	0	3	3	3	6	6
Page fault	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

* Page fault

This example incurs 16 page faults in FIFO algorithm,

FIFO algorithm with four page frames

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1		1	1	1	1	1	1	1	1	1	1	1	1	5	5	5
2			2	2	2	2	2	2	2	2	2	2	2	2	6	6
3				3	3	3	3	3	3	3	3	3	3	3	3	7
Page fault	*	*	*	*									*	*	*	*

Number of page fault is 8.

2) LRU

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	3	3	3	2	2	2	1	1	1	4	4	4	7
1		1	1	1	0	0	0	3	3	3	2	2	2	5	5	5
2			2	2	2	1	1	1	0	0	0	3	3	3	6	6
Page fault	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Number of page fault = 16.

Consider the same reference string with 4 page frames

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1		1	1	1	1	1	1	1	1	1	1	1	1	5	5	5
2			2	2	2	2	2	2	2	2	2	2	2	2	6	6
3				3	3	3	3	3	3	3	3	3	3	3	3	7
Page fault	*	*	*	*									*	*	*	*

Number of page fault = 8.

3) Optimal

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	7
0	0	0	0	0	0	0	0	0	0	1	1	1	4	4	7
1		1	1	1	1	1	2	2	2	2	2	2	2	5	5
2			2	3	3	3	3	3	3	3	3	3	3	3	6
Page fault	*	*	*	*			*			*			*	*	*

Number of page fault = 10.

With page frame = 4 for same reference string.

Frame	0	1	2	3	0	1	2	3	0	1	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1		1	1	1	1	1	1	1	1	1	1	1	5	5	5
2			2	2	2	2	2	2	2	2	2	2	2	6	6
3				3	3	3	3	3	3	3	3	3	3	3	7
Page fault	*	*	*	*								*	*	*	*

Number of page fault = 8.

Example 5.10.2 *Consider the following page-reference string :*

2 3 2 1 5 2 4 5 3 2 5 2

How many page faults occur for the following replacement algorithms, assuming three frames. i) FIFO ii) LRU iii) Optimal.

Solution : i) FIFO

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	5	5	5	5	3	3	3	3
1		3	3	3	3	2	2	2	2	2	5	5
2				1	1	1	4	4	4	4	4	2
Page fault	*	*		*	*	*	*		*		*	*

Number of page fault = 9.

ii) LRU

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	2	2	3	3	3	3
1		3	3	3	5	5	5	5	5	5	5	5
2				1	1	1	4	4	4	2	2	2
Page fault	*	*		*	*		*		*	*		

Number of page fault = 7.

iii) Optimal

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	4	4	4	2	2	2
1		3	3	3	3	3	3	3	3	3	3	3
2				1	5	5	5	5	5	5	5	5
	*	*		*	*		*			*		

Number of page fault = 6.

Example 5.10.3 *Consider the following page reference. Indicate page faults and calculate total number of page faults for optimal and LRU. The total number of available frames are 4.*

1, 2, 3, 2, 5, 6, 3, 4, 6, 3, 7, 3, 1, 5, 3, 6, 3, 4, 2, 4, 3, 4, 5, 1.

Solution : i) Optimal (frames = 4)

Reference string	0	1	2	3	Page fault
1	1	–	–	–	*
2	1	2	–	–	*
3	1	2	3	–	*
2	1	2	3	–	
5	1	2	3	5	*
6	1	6	3	5	*
3	1	6	3	5	
4	1	6	3	4	*
6	1	6	3	4	

3	1	6	3	4	
7	1	6	3	7	*
3	1	6	3	7	
1	1	6	3	7	
5	1	6	3	5	*
3	1	6	3	5	
6	1	6	3	5	
3	1	6	3	5	
4	1	4	3	5	*
2	2	4	3	5	*
4	2	4	3	5	
3	2	4	3	5	
4	2	4	3	5	
5	2	4	3	5	
1	1	4	3	5	*

Number of page fault = 11.

ii) LRU

Reference string	0	1	2	3	Page fault
1	1	–	–	–	*
2	1	2	–	–	*
3	1	2	3	–	*
2	1	2	3	–	
5	1	2	3	5	*
6	6	2	3	5	*
3	6	2	3	5	
4	6	4	3	5	*
6	6	4	3	5	
3	6	4	3	5	
7	6	4	3	7	*

3	6	4	3	7	
1	6	1	3	7	*
5	5	1	3	7	*
3	5	1	3	7	
6	5	1	3	6	*
3	5	1	3	6	
4	5	4	3	6	*
2	2	4	3	6	*
4	2	4	3	6	
3	2	4	3	6	
4	2	4	3	6	
5	2	4	3	5	*
1	1	4	3	5	*

Number of page fault = 14.

Example 5.10.4 *Consider the following page reference string :*

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

How many page faults would occur for the following replacement algorithms, assuming 1, 2, 3, 4, 5, 6 or 7 frames ? Initially all frames are emtpy :

i) LRU ii) FIFO iii) Optimal. **RGTU : Dec.-02**

Solution :

i) LRU

No. of page frame	1	2	3	4	5	6	7
No. of page fault	20	18	15	10	9	6	7

ii) FIFO

No. of page frame	1	2	3	4	5	6	7
Page fault	20	18	16	14	10	10	7

iii) Optimal

No. of page frame	1	2	3	4	5	6	7
Page fault	20	15	11	8	7	7	7

Example 5.10.5 *Consider the following reference string :*

A B C D A B E A B C D E

How many page faults would occur for the following replacement algorithms with 3 and 4 frames ?

i) FIFO replacement ii) Optimal replacement iii) LRU replacement Whether this reference string suffers from Belady's anomaly ? Explain. **RGTU : June-03**

Solution :

i) FIFO : (Page frame = 3)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	D	D	D	E	E	E	E	E	E
2	-	B	B	B	A	A	A	A	A	C	C	C
3	-	-	C	C	C	B	B	B	B	B	D	D
Page Fault	*	*	*	*	*	*	*			*	*	

Total number of page fault = 9

(Page frame = 4)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	E	E	E	E	D	D
2	-	B	B	B	B	B	B	A	A	A	A	E
3	-	-	C	C	C	C	C	C	B	B	B	B
4	-	-	-	D	D	D	D	D	D	C	C	C
Page Fault	*	*	*	*			*	*	*	*	*	*

Total number of page fault = 10

FIFO algorithm for this reference string suffers from Belady's anomaly.

ii) Optimal (Page frame = 3)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	A	A	A	C	C	C
2	-	B	B	B	B	B	B	B	B	B	D	D
3	-	-	C	D	D	D	E	E	E	E	E	E
Page Fault	*	*	*	*			*			*	*	

l number of page fault = 7

(Page frame = 4)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	A	A	A	A	D	D
2	-	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	C	C	C	C	C	C
4	-	-	-	D	D	D	E	E	E	E	E	E
Page fault	*	*	*	*			*				*	

Total no. of page fault = 6

iii) LRU (Page frame = 3)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	D	D	D	E	E	E	C	C	C
2	-	B	B	B	A	A	A	A	A	A	D	D
3	-	-	C	C	B	B	B	B	B	B	B	E
Page fault	*	*	*	*	*	*	*			*	*	*

Total number of page fault = 10

(Page frame = 4)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	A	A	A	A	A	E
2	-	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	E	E	E	E	D	D
4	-	-	D	D	D	D	D	D	D	C	C	C
Page fault	*	*	*	*			*			*	*	*

Total number of page fault = 8

Example 5.10.6 *If the reference string is 1, 2, 3, 4, 5, 2, 5, 3, 2, 5, 4, 1, 5 and the maximum number of pages which can be stored at a time in memory is 3 then calculate the number of page fault when LRU (least recently used) page replacement algorithm is used.* **RGTU : June-04**

Solution :

LRU

	1	2	3	4	5	2	5	3	2	5	4	1	5
1	1	1	1	4	4	4	4	3	3	3	4	4	4
2	-	2	2	2	5	5	5	5	5	5	5	5	5
3	-	-	3	3	3	2	2	2	2	2	2	1	1
Page Fault	*	*	*	*	*	*		*			*	*	

Total number of page fault = 9.

Example 5.10.7 *Consider the following page reference string :*

A B C D B A E F B A B C E G C B A B

How many page faults will occur for LRV, FIFD and optimal page replacement algorithms, assuming four available frames ? **RGTU : Dec.-04**

Solution :

i) LRU (page frame = 4)

	A	B	C	D	B	A	E	F	B	A	B	C	E	G	C	B	A	B
1	A	A	A	A	A	A	A	A	A	A	A	A	A	G	G	G	G	G
2	-	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	E	E	E	E	E	C	C	C	C	C	C	C
4	-	-	-	D	D	D	D	F	F	F	F	F	E	E	E	E	A	A
Page Fault	*	*	*	*			*	*				*	*	*			*	

Total number of page fault = 10

ii) FIFO (page frame = 4)

	A	B	C	D	B	A	E	F	B	A	B	C	E	G	C	B	A	B
1	A	A	A	A	A	A	E	E	E	E	E	C	C	C	C	C	C	C
2	-	B	B	B	B	B	B	F	F	F	F	F	E	E	E	E	E	E
3	-	-	C	C	C	C	C	C	B	B	B	B	B	G	G	B	B	B
4	-	-	-	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A
Page Fault	*	*	*	*			*	*	*	*		*	*	*		*		

Total number of page fault = 12

iii) OPTIMAL (page frame = 4)

	A	B	C	D	B	A	E	F	B	A	B	C	E	G	C	B	A	B
1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	-	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
4	-	-	-	D	D	D	E	F	F	F	F	F	E	G	G	G	G	G
Page Fault	*	*	*	*			*	*					*	*				

Total number of page fault = 8

Example 5.10.8 *Consider the following reference string for pages :*

0, 1, 1, 2, 3, 4, 3, 3, 1, 0, 2

Apply page replacement policies-LRU and optimal on this reference string assuming three and four frames respectively. **RGTU : Dec.-06**

Solution :

i) LRU (Page frame = 3)

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	3	3	3	3	3	3	2
2	-	1	1	1	1	4	4	4	4	0	0
3	-	-	-	2	2	2	2	2	1	1	1
Page Fault	*	*		*	*	*			*	*	*

Total number of page fault = 8

Page frame = 4

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	0	4	4	4	4	4	2
2	-	1	1	1	1	1	1	1	1	1	1
3	-	-	-	2	2	2	2	2	2	0	0
4	-	-	-	-	3	3	3	3	3	3	3
Page Fault	*	*		*	*	*				*	*

ii) Optimal (Page frame = 3)

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	0	4	4	4	4	0	0
2	-	1	1	1	1	1	1	1	1	1	2
3	-	-	-	2	3	3	3	3	3	3	3
Page fault	*	*		*	*	*				*	*

Total number of page fault = 7

Page frame = 4

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	0	0	0	0	0	0	0
2	-	1	1	1	1	1	1	1	1	1	1
3	-	-	-	2	2	4	4	4	4	4	2
4	-	-	-	-	3	3	3	3	3	3	3
Page Fault	*	*		*	*	*					*

Total number of page fault = 6

Example 5.10.9 *Assume a reference string of (7012030423). With page frame size 03, calculate number of page faults with the following algorithms :*

i) LRU ii) FIFO. **RGTU : June-08**

Solution :

i) LRU (Page frame = 3)

	7	0	1	2	0	3	0	4	2	3
1	7	7	7	2	2	2	2	4	4	4
2	-	0	0	0	0	0	0	0	0	3
3	-	-	1	1	1	3	3	3	2	2
Fault	*	*	*	*		*		*	*	*

Total number of page fault = 8

ii) FIFO

	7	0	1	2	0	3	0	4	2	3
1	7	7	7	2	2	2	2	4	4	4
2	-	0	0	0	0	3	3	3	2	2
3	-	-	1	1	1	1	0	0	0	3
Fault	*	*	*	*		*	*	*	*	*

Total number of page fault = 9

Example 5.10.10 *Consider the following page reference string :*

1, 2, 3, 1, 4, 5, 6, 2, 1, 3, 2, 7, 6, 3, 4, 1, 2, 6

How many page faults would occur for the LRU replacement algorithm assuming six frames ? All frames are initially empty. **RGTU : June-10**

Solution :

LRU

	1	2	3	1	4	5	6	2	1	3	2	7	6	3	4	1	2	6
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	-	-	-	-	4	4	4	4	4	4	4	7	7	7	7	7	7	7
5	-	-	-	-	-	5	5	5	5	5	5	5	5	5	4	4	4	4
6	-	-	-	-	-	-	6	6	6	6	6	6	6	6	6	6	6	6
Page fault	*	*	*		*	*	*					*			*			

Total number of page fault = 8.

5.10.6 Second Chance Algorithm

- Second chance algorithm is also called clock algorithm. It maintains reference bit per page.

- Goal : Remove a page that has not been referenced recently.
- Second chance algorithm is actually a FIFO replacement algorithm with a small modification that causes it to approximate LRU. It uses reference bit. When a page is selected according to a FIFO order, user checks its reference bit.
- It uses a circular buffer with as many slots as there are frames in memory. Each buffer slot contains a record (page number; reference bit), for a page currently in memory.
- If reference bit is set then page was referenced and this page is not removed otherwise page is removed. When a page is replaced, a pointer is set to point to the next frame in buffer.
- Second chance is looking for an old page that has not been referenced in the most recent clock interval. If all the pages have been referenced, second chance degenerates into pure FIFO.
- It requires hardware support for reference bit.
- A reference bit for each frame is set to 1 in following conditions :
  1. Page is first time loads into the frame memory.
  2. When page frame is referenced.
- When it is time to replace a page, the first frame encountered with the reference bit set to 0 is replaced. During the search for replacement, each reference bit set to 1 is changed to 0.

Example :

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
0	0	0	0	0	2	2	2	2	2	2	3	3	3	3	4	4	4
1	-	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1
2	-	-	3	3	3	3	5	5	5	5	5	5	2	2	2	2	0
3	-	-	-	6	6	6	6	6	6	0	0	0	0	5	5	5	5
Page Fault	PF	PF	PF	PF	PF	PF	PF			PF	Pf	PF	PF	PF	PF		PF

Total number of page fault = 14

5.10.7 Not Recently Used

- Each page maintains two bit : Reference bit and dirty bit. Bits are set when page is referenced and/or modified. These bits are updated on every memory reference and must be set by the hardware.
- Dirty bit is set when page is written. Dirty bit is sometimes called as Modified Bit.
- Pages are classified into four classes :

0	Page is not referenced and not dirty
1	Page is not referenced but dirty
2	Page is referenced but not dirty
3	Page is referenced but dirty

Choose a page from class 1.

If none, choose a page from class 2.

If none, choose a page from class 3.

If none, choose a page from class 4.

- NRU randomly chooses a page from the lowest class to remove. Reference bit is cleared periodically. The algorithm removes the page from the lowest non-empty class.
- Variants of this scheme are used in many operating systems, including UNIX and Macintosh operating system. Most variations use a scan pointer and go through the page frames one by one, in some order, looking for a page that has not been used recently.

Advantages :

1. Simple to implement
2. Easy to understand

Disadvantages :

1. Performance adequate
2. Overhead of maintaining two bits

University Questions

1. *Write short note on : NRU page replacement algorithm.* **GTU : Summer-13, Marks 4**
2. *For the page reference string :*
   7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 1, 7, 0, 1
   Calculate the page faults applying i) Optimal ii) LRU and iii) FIFO page replacement algorithms for a memory with three frames. **GTU : Winter-13, Marks 7**

5.11 Thrashing

- Thrashing occurs when a process does not have "enough" frames allocated to store the pages it uses repeatedly, the page fault rate will be very high.
- A process is thrashing if it is spending more time for paging in/out than executing. Since thrashing leads to low CPU utilization, the operating system's medium or long term scheduler may step in, detect this and increase the degree of multiprogramming.
- Local replacement algorithms can limit the effects of thrashing. If the degree of multiprogramming is increased over a limit, processor utilization falls down considerably because of thrashing. Fig. 5.11.1 shows thrashing.

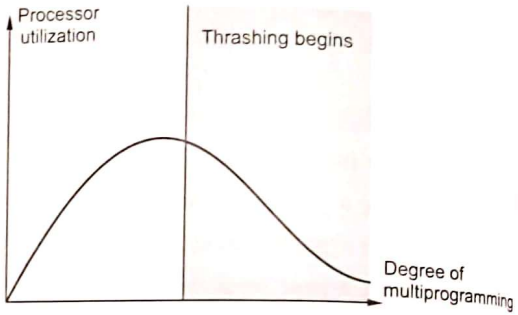


Fig. 5.11.1 Thrashing

- The selection of a replacement policy to implement virtual memory plays an important part in the elimination of the potential for thrashing. A policy based on the local mode will tend to limit the effect of thrashing. Replacement policy based on the global mode is more likely to cause thrashing. Since all pages of memory are available to all transactions, a memory-intensive transaction may occupy a large portion of memory, making other transactions susceptible to page faults and resulting in a system that thrashes.
- Thrashing is solved by using working set model and page fault frequency.

5.11.1 Working Set Model

- Working sets model is proposed by Peter Denning to prevent thrashing.
- We define the working set of information W(t, τ) of a process at time t to be the collection of information referenced by the process during the process time interval (t–τ, t). Fig. 5.11.2 shows working set model.

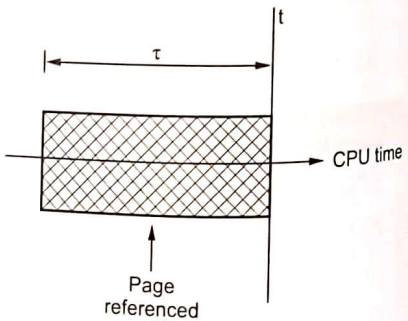


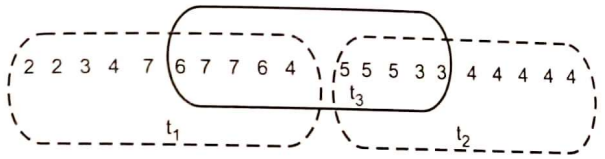
Fig. 5.11.2 Working set model

- Here τ is the working set parameter. The elements of W(t, τ) are pages but they can be anything else used by a process.
- If the sum of all working sets of all runnable threads exceeds the size of memory, then stop running some of the threads for a while. Divide processes into two groups : active and inactive.
- When a process is active its entire working set must always be in memory : never execute a thread whose working set is not resident. When a process becomes inactive, its working set can migrate to disk.
- Threads from inactive processes are never scheduled for execution. The collection of active processes is called the balance set. The system must have a mechanism for gradually moving processes into and out of the balance set. As working sets change, the balance set must be adjusted.

How to compute working sets ?

Δ = Working set windows size.

Example : Assume windows size is 10 and the reference string given below, on which the window is shown at different time instants.



Working set for process

$WS(t_1) = \{2, 3, 4, 6, 7\}$ $WS(t_2) = \{5, 3, 4\}$

$WS(t_3) = \{6, 7, 4, 5, 3\}$

Page Fault Frequency

- Page fault frequency is used to preventing thrashing. Page fault rate is controlled by using this method. In the per-process replacement policy, each process is allocated a fixed number of physical page frames. Then monitor the rate at which page faults are occurring for each process.

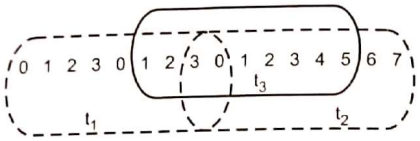
1. If the rate gets too high for a process, assume that its memory is overcommitted; increase the size of its memory pool.
2. If the rate gets too low for a process, assume that its memory pool can be reduced in size.
3. If the sum of all memory pools doesn't fit in memory, deactivate some processes.

Example 5.11.1 *Given the following reference string :*

0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7

Determine $WS(t_i)$ *with* $\Delta = 9$.

Solution :



$WS(t_1) = \{ 0, 1, 2, 3 \}$

$WS(t_2) = \{ 3, 0, 1, 2, 4, 5, 6, 7 \}$

$WS(t_3) = \{ 1, 2, 3, 0, 4, 5 \}$

Locality of reference

- Locality of reference is also known as the principle of locality. Many applications continually reference large amounts of data.
- Locality of reference is observes that an application does not access all of its data at once with equal probability. Instead, it accesses only a small portion of it at any given time.
- There are two basic types of reference locality.

  1. Temporal locality 2. Spatial locality

  1. Temporal locality : Temporal locality is locality over time. If at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.

  2. Spatial locality : If a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.

□□□

6 I/O Management

Syllabus

Principles of I/O hardware : I/O devices, Device controllers, Direct memory access, Principles of I/O software : Goals of interrupt handlers, Device drivers , Device independent I/O software, Secondary - Storage structure : Disk structure, Disk scheduling algorithm.

Contents

6.1 Principles of Input/Output Hardware . . . . . **Summer-13, 14,** . . . . . Marks 7

6.2 Principles of I/O Software . . . . . **Winter-13,** . . . . . Marks 4

6.3 I/O Software Layers

6.4 Disks

6.5 RAID . . . . . **Summer-13,14,** . . . . . Marks 7

6.6 Disk Performance Parameters

6.7 Disk ARM Scheduling Algorithm . . . . . **Summer-13,** . . . . . Marks 7

6.1 Principles of Input/Output Hardware

GTU : Summer-13,14

6.1.1 I/O Devices

- I/O devices are of two types :
  1. Block devices
  2. Character devices
- **Block device** stores information in fixed size blocks. Operating system assigns address to each block. A Block device is one with which the driver communicates by sending entire blocks of data. Examples for block devices : hard disks, USB cameras, Disk-On-Key.
- A block device is a device (e.g., a disk) that can host a file system. In UNIX systems, a block device can only handle I/O operations that transfer one or more whole blocks, which are usually 512 bytes in length.
- Every block device driver must provide an interface to the buffer cache as well as the normal file operations interface. Each block device is identified by major and minor numbers.
- Block devices are characterized by the capability to read and write blocks of data to and from random locations on an addressable medium. Examples of block devices include hard drives and USB Flash drives.
- Block and character devices are represented for user space applications as files than can be manipulated using the traditional file API.
- A **character device** is one with which the driver communicates by sending and receiving single characters. Examples for character devices : serial ports, parallel ports and sound cards.
- The only difference between a character device and a regular file is that, user can always move back and forth in the regular file, whereas most character devices are just data channels, which user can only access sequentially. User can perform mmap and lseek operation on the character device.
- Character devices can be thought of as serial streams of sequential data. Examples of character devices include serial ports and keyboards.
- Linux supports loadable device drivers, it is relatively easy to demonstrate a simple device driver skeleton. A device driver is a special kind of binary module. Unlike a stand-alone binary executable application, a device driver cannot simply be executed from a command prompt.
- Character device drivers are referenced through special files in the file system. This file is stored in the */dev* folder. The command "*ls -l*" is used to check the

folder. The character "c" in the file listing indicates the file is a char device. It also contains device major and minor numbers.

6.1.2 Device Controllers

- The logic circuit that contains these registers is called the **device controller,** and the software that communicates with the controller is called a **device driver.** Fig. 6.1.1 shows arrangement of device controller.

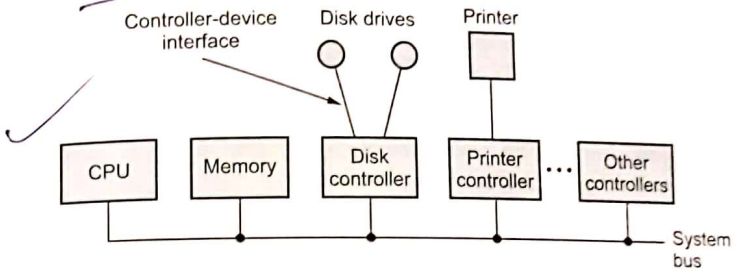


Fig. 6.1.1 Arrangement of device controller

- Input-output unit used in the computer are the combination of the electronic chip and mechanical part. It is possible to separate two parts as a component and design separately. The electronic components are called as **device controller.**
- Electronic part is printed circuit board and PCI bus. The mechanical part is the device itself. Many standards are used for connecting board and cables. All components manufactures follows the standards.
- The interface between the controller and the device is often a very low level interface. The controllers function is to convert the serial bit stream into a block of bytes and perform any error correction.
- Interface between device and controller is a standard interface and it is de facto standard. These standards are defined by ANSI, ISO and IEEE.
- The controller performs the conversion of bit stream into a block of bytes and error correction. Monitor controller also works as a bit serial device at low level. It reads the character to be displayed from the memory and generate signal. This generated signal writes to modulated CRT beam.
- Operating system initializes the controller with some parameters. The parameters are number of characters, pixel per line and number of lines per screen.

6.1.3 Memory Mapped I/O

- Instead of having special methods for accessing the values to be read or written, just get them from memory or put them into memory. The device is connected directly to certain main memory locations.
- When using memory-mapped I/O, the same address space is shared by memory and I/O devices. Some addresses represent memory cells, while others represent registers in I/O devices. No separate I/O instructions are needed in a CPU that uses memory-mapped I/O. Instead, we can perform I/O operations using any instruction that can reference memory.
- I/O devices and memory share the same address space. Each I/O device is assigned a unique set of addresses. When the processor places a particular address on the address lines, the device recognizing this address responds to the commands on the control lines.
- Memory addresses xffff0000 and above is used for I/O devices. The register used in device controller. These registers are given a memory address. Memory is attached to the bus. When the bus sees certain addresses, it knows they are not memory addresses, but are addresses for accessing I/O devices.
- The processor requests either a read or a write operation, and the requested data is transferred over the data lines. Any machine instruction that can access memory can be used to transfer data to/from input-output devices.
- Fig. 6.1.2 shows memory mapped I/O.
- With separate device addresses, each component in a device has a two component address. One address is device address and other is address of command, status and data registers with a device. Devices are associated with logical primary memory addresses rather than having a specialized device address.

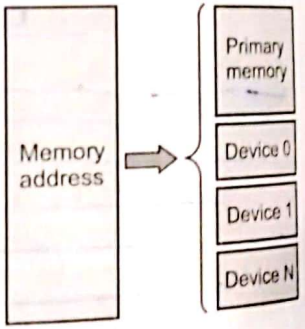


Fig. 6.1.2 Memory mapped I/O

1. Memory mapped I/O reduces the number of instruction types in the processor.
2. No special protection mechanism is needed to keep user processes from performing I/O.
3. Every instruction that can reference memory can also reference control register.

- Fig. 6.1.3 shows single and dual bus architecture. In single bus architecture, all memory modules and all I/O device must examine all memory references to see

which one to respond to. This is straightforward method. I/O buses tap into the processor-memory bus via bus adaptors.

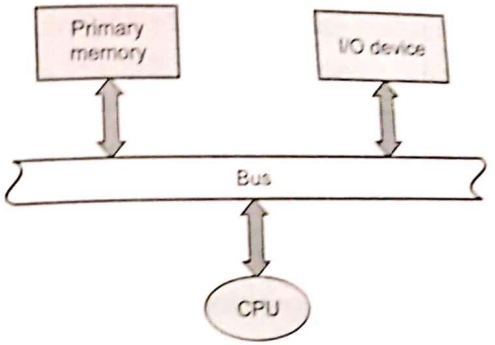


Fig. 6.1.3 (a) Single bus architecture

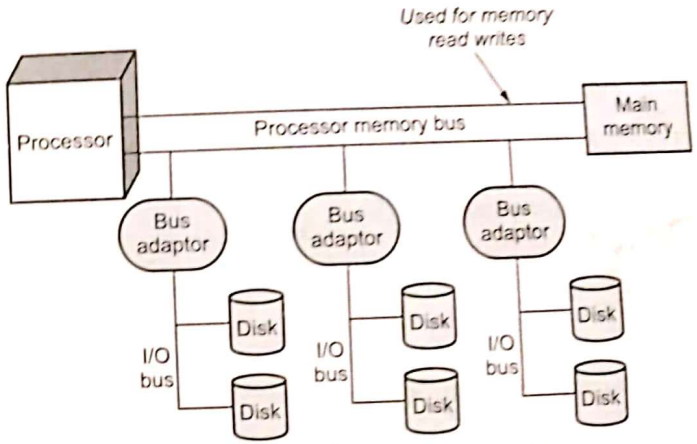


Fig. 6.1.3 (b) Dual bus architecture

- Modern computer uses dual bus architecture. In this architecture, I/O device have no way of responding to memory. Separate mechanism is required to make memory mapped I/O work on a system with multiple buses. Three methods is used for this.
  1. First send all memory references to the memory.
  2. Use of snooping device on the memory bus.
  3. Filter addresses in the PCI bridge chip.
- Memory mapped operations are useful because they are more efficient than the alternative and also save on file descriptors. Memory mapping is used for dynamic libraries. The basic idea of memory mapping is to map a byte range of a file to a range of process addresses.

Advantages :

1. It makes programming simpler.
2. It does not have special commands to access I/O devices.

6.1.4 Direct Memory Access

- Direct Memory Access (DMA) is an I/O technique that allows a control unit to directly access main memory. This means that once reading or writing has begun the remainder of the data can be transferred to and from memory without processor (CPU) intervention.
- Without DMA, the processor is responsible for the physical movement of data between main memory and a device.
- A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called Direct Memory Access (DMA).
- DMA can be used with either polling or interrupt software. Fig. 6.1.4 shows the typical DMA block diagram. DMA is particularly useful on devices like disks, where many bytes of information can be transferred in single I/O operations When used in conjunction with an interrupt, the CPU is notified only after the entire block of data has been transfered. For each byte or word transfered, it must provide the memory address and all the bus signals that control the data transfer.

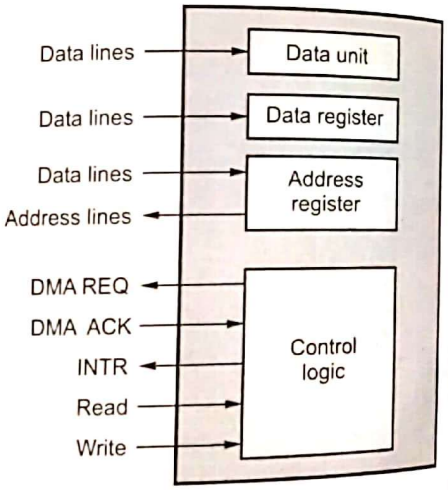


Fig. 6.1.4 DMA block diagram

- Following Fig. 6.1.5 shows the difference between traditional I/O and DMA.

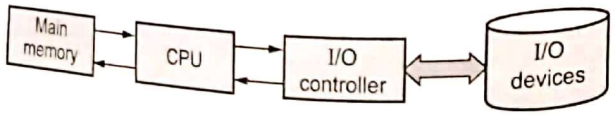


Fig. 6.1.5 (a)

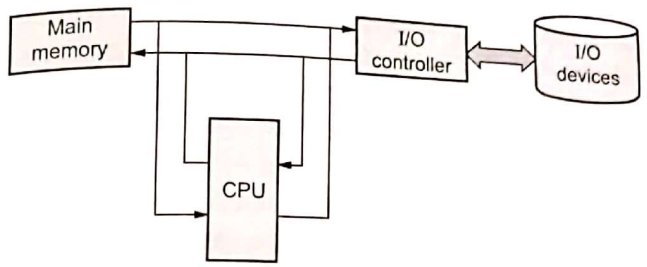


Fig. 6.1.5 (b)

- Operation of the DMA is shown in Fig. 6.1.6. The following sequence of steps takes place for a DMA transfer :

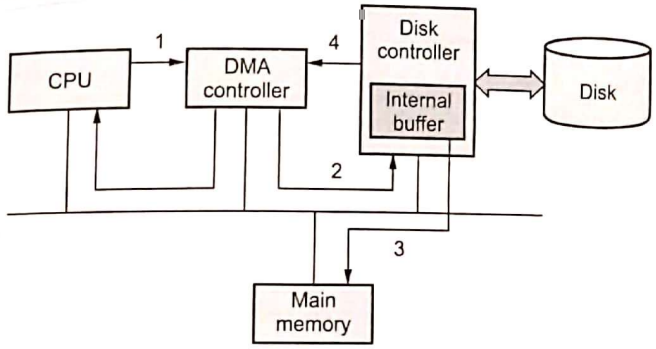


Fig. 6.1.6 DMA operation

1. DMA controller is programmed by CPU and its registers are set to perform required operations. It issues a command to a DMA controller for reading data from the secondary storage disk and stores it in the internal buffer. It verifies the checksum If disk controller buffer contains valid data then DMA operation starts.
2. Data transfer is start by disk controller by issuing read request over the bus. The DMA controller then puts the memory address on the bus.
3. Disk controller fetches the next word from the internal buffer, it knows where to write it. Writing to memory is another bus cycle.
4. Disk controller sends an acknowledgment to DMA controller when writes completes.
5. DMA controller increments the memory address to use and decrements the byte count. The step 2 to step 4 is repeated until byte count reach to zero.

- The DMA controller interrupts the CPU to know that the transfer is now complete.
- The DMA Controller has several options available for the transfer of data. Few options are given below :

1. **Cycle steal :** A read or write signal is generated by the DMA controller and the I/O device either generates or latches the data. The DMA controller effectively steals cycles from the processor in order to transfer the byte, so single byte transfer is also known as cycle stealing.

2. **Burst transfer :** To achieve block transfers, some DMA controller incorporate an automatic sequencing of the value presented on the address bus. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMA controller will release the bus. When the DMA controller operates in burst mode, the CPU is halted for the duration of the data transfer.

3. **Fly by mode :** The DMA controller tells the device controller to transfer the data directly to main memory. It requires extra bus cycle.

- A device driver using DMA has to talk to hardware connected to the interface bus, which uses physical addresses, whereas program code uses virtual addresses.
- Use of internal buffer is because of following reasons :
  1. Checksum : Disk controller can verify the checksum before starting a data transfer. Error is generated if checksum is incorrect. Then there is no data transfer.
  2. Once data transfer is start, at constant rate data is arriving even if disk controller is not ready. To synchronize between the disk controller and main memory, internal buffer is required.
- In DMA transfers, The CPU initiates the transfer, then it does other operations while the transfer is in progress, and it receives an interrupt from the DMA controller once the operation is finished. This is especially useful in real-time computing applications.
- DMA can lead to cache coherency problems. If a CPU is equipped with a cache and an external memory, then the data the DMA controller has access to may not be updated with the correct data stored in the cache.

DMA Configuration :

1. Single bus, detached DMA

- All the module use same system bus.
- This configuration is inefficient but inexpensive.
- It uses programmed I/O to exchange data between memory and an I/O module through the DMA module.

2. I/O bus

- I/O bus provide easily expandable configurati

- It reduces number of I/O interfaces in the DMA module.
- Exchange of data between the DMA and I/O module place off the system bus.

Advantages of DMA :

1. high transfer rates.
2. Fewer CPU cycles for each transfer.

Disadvantages of DMA :

1. Expensive system.
2. Synchronization mechanisms must be provided in order to avoid accessing non-updated information from RAM.

University Questions

1. *Draw the block diagram for DMA. Explain the steps for DMA data transfer.* **GTU : Summer-13, Marks 7**
2. *Write short note : Direct Memory Access (DMA)* **GTU : Summer-14, Marks 7**
3. *Write short note : Device Controllers.* **GTU : Summer-14, Marks 7**

6.2 Principles of I/O Software

GTU : Winter-13

6.2.1 Goals of I/O Software

Device independence

- Device independence is the key concept in the design of I/O software. We want to have most of the OS to be unaware of the characteristics of the specific devices attached to the system.
- This objective has been accomplished quite well for files stored on various devices. Most of the OS, including the file system code, and most applications can read or write a file without knowing if the file is stored on a floppy disk, an internal SATA hard disk, an external USB SCSI disk, an external USB Flash RAM, a tape, or CD-ROM for reading.
- This principle also applies for user programs reading or writing streams. A program reading from ``standard input'', which is normally the user's keyboard can be told to instead read from a disk file with no change to the application program. Similarly, ``standard output'' can be redirected to a disk file. However, the low-level OS code dealing with disks is rather different from that dealing keyboards and terminals.

- One can say that device independence permits programs to be implemented as if they will read and write generic or abstract devices, with the actual devices specified at run time. Although writing to a disk has differences from writing to a terminal, UNIX "cp" command, DOS "copy" command, and many programs we compose need not be aware of these differences.
- However, there are devices that really are special. The graphics interface to a monitor does not resemble the ``stream of bytes".

Error handling

- There are several aspects to error handling including : Detection, correction and reporting.
  1. Detection should be done as close to where the error occurred as possible before more damage is done. Moreover, the error may be obvious at the low level, but harder to discover and classify if the erroneous data is passed to higher level software.
  2. Correction is sometimes easy, for example ECC memory does this automatically. Other easy cases include successful retries for failed Ethernet transmissions. In this example, while logging is appropriate, it is quite reasonable for no action to be taken.
  3. Error reporting tends to be bad. The trouble is that the error occurs at a low level but by the time it is reported the context is lost.

Creating the illusion of synchronous I/O

- I/O must be asynchronous for good performance. That is the operating system cannot simply wait for an I/O to complete. Instead, it proceeds with other activities and responds to the interrupt that is generated when the I/O has finished.
- Users mostly want no part of this. Consider the following code sequence :

  Read X

  Y = X+1

  Print Y

should print a value one greater than that read. But if the assignment is performed before the read completes, the wrong value can easily be printed.

Buffering

- Buffering is often needed to hold data for examination prior to sending it to its desired destination.

- Since this involves copying the data, which can be expensive, modern systems try to avoid as much buffering as possible. This is especially noticeable in network transmissions, where the data could conceivably be copied many times.
  1. From user space to kernel space as part of the write system call.
  2. From kernel space to a kernel I/O buffer.
  3. From the I/O buffer to a buffer on the network adaptor.
  4. From the adapter on the source to the adapter on the destination.
  5. From the destination adapter to an I/O buffer.
  6. From the I/O buffer to kernel space.
  7. From kernel space to user space as part of the read system call.

Sharable vs. Dedicated Devices

- For devices like printers and CD-ROM drives, only one user at a time is permitted. These are called serially reusable devices, which we studied in the deadlocks chapter. Devices such as disks and Ethernet ports can, on the contrary, be shared by concurrent processes without any deadlock risk.

6.2.2 Programmed I/O

- I/O can be performed in three different ways :
  1. Programmed I/O
  2. Interrupt driven I/O
  3. I/O using DMA
- With **programmed I/O** the main processor moves the data between main memory and the device. This is the most straightforward method for performing I/O.
- In the simplest implementation, the processor, when it seeks to use a device, loops continually querying the device status, until the device reports that it is free. This is called **polling** or **busy waiting.**
- In polling
  1. The CPU sends a request to an I/O device.
  2. The I/O device processes the request and sets a device-ready signal when it is ready.
  3. The CPU reads in this signal via another I/O address and checks the value.
  4. If the signal is set, it performs the data transfer. If not, it loops back.
- If we poll infrequently, there can be a significant delay from when the previous I/O is complete to when the OS detects the device availability.

- If we poll frequently and the device is slow, polling is clearly wasteful.
- The extreme case is where the process does nothing between polls. For a slow device this can take the CPU out of service for a significant period.
- CPU is responsible for data transfer. It transfers data between primary memory and device controller. Following are the steps for polling :

1. Application process issues a read operation.
2. Device driver check the status of device. If device is busy, the driver waits for it to become idle.
3. Driver stores an input command into the controller command register. It starts the device.
4. Driver continuously read the status register while waiting for the device to complete its operations.
5. Driver copies the content of the controller data register into the user process space.

Advantages of program-controlled I/O :

1. All control is directly under the control of the program, so changes can be readily implemented.
2. The order in which devices are serviced is determined by the program. The priority of a device can be varied under program control.
3. It is relatively easy to add or delete devices.

Disadvantage of program-controlled I/O :

1. Time may be spent testing the status inputs of the I/O devices, when the devices do not need servicing.
2. If a particular device has its data available for only a short time, the data may be missed because the input was not tested at the appropriate time.

University Question

1. *Explain goals of I/O software.* **GTU : Winter-13, Marks 4**

6.3 I/O Software Layers

- Layers of abstraction as usual prove to be effective. Most systems are believed to use the following layers :

1. User-level I/O routines.
2. Device-independent (kernel-level) I/O software.

3. Device drivers.
4. Interrupt handlers.

6.3.1 Interrupt Handlers

- When interrupt I/O is used, the CPU is free to ignore the I/O module until the interrupt signal a received. The only added overhead is processing a single interrupt when the I/O operation completes.
- In this method the program issues an I/O command and than continues to execute until it is interrupted by the I/O hardware to signal the end of I/O operation. Here the program enters a wait loop in which it repeatedly checks the device status. During this process the processor is not performing any useful computation.
- There are many situations where tasks can be performed while waiting for an I/O device to be ready, to allow this the I/O device should alert the processor when it becomes ready. It can be done by sending a hardware signal called an **interrupt**.
- The routine executed in response to an interrupt request is called Interrupt Service Routine (ISR). The processor first completes execution of instruction then it loads the program counter with the address of 1st instruction of ISR.
- In a multiprogramming system the wasted CPU time could be used by another process; because the CPU is used by other processes in addition to the one waiting for the I/O operation completion, in multiprogramming system may result a sporadic detection of I/O completion; this may be remedied by use of interrupts.
- The reason for incorporating the interrupts into computer hardware is to eliminate the need for a device driver to constantly poll the CSR.

Steps for performing an input operation

- The application process requests a read operation.
- The device driver queries the CSR to find out if the device is idle; if busy, then it waits until the device becomes idle.
- The driver stores an input command into the controller's command register, thus starting the device.
- When this part of the device driver completes its work, it saves information regarding the operation it began in the device status table; this table contains an entry for each device in system; the information written into this table contains the return address of the original call and any special parameters for the I/O operation; the CPU, after is doing this, can be used by other program, so the

device manager invokes the scheduler part of the process manager. It then terminates.

- The device completes the operation and interrupts the CPU, therefore causing an interrupt handler to run.
- The interrupt handler determines which device caused the interrupt; it then branches to the device handler for that device.
- Device driver retrieves the pending I/O status information from the device status table. The device driver copies the content of the controller's data register(s) into the user process's space.
- The device handler returns the control to the application process.

6.3.2 Device Drivers

- A device driver is a program that controls a particular type of device that is attached to user computer. There are device drivers for printers, displays, CD-ROM readers, hard disk and so on.
- Applications do not directly interact with I/O devices. But it will interact with the help of operating system. Some device driver provides access to user space applications and the devices are managed directly by the applications.
- Fig. 6.3.1 shows device drivers partitions.

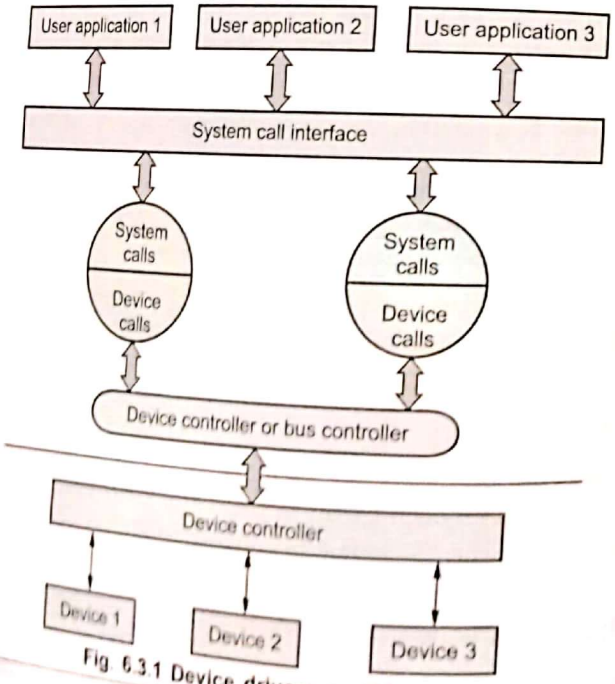
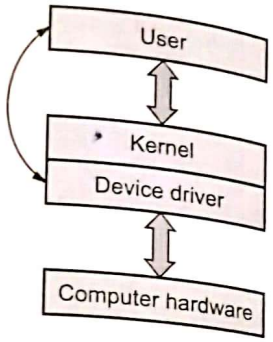


Fig. 6.3.1 Device drivers partitions

- General examples of device controllers include hard disk controllers, display controllers, and audio controllers that in turn manage devices connected to them. More technical examples would be an IDE controller, PCI controller, USB controller, SPI controller.
- A device driver has two parts :
  1. Device-specific
  2. OS-specific.
- The device-specific portion of a device driver remains the same across all operating systems, and is more about understanding and decoding the device data sheets than software programming. A data sheet for a device is a document with technical details of the device, including its operation, performance, programming, etc.
- The OS-specific portion is the one that is tightly coupled with the OS mechanisms of user interfaces, and thus differentiates a Linux device driver from a Windows device driver.
- General classification of device drivers is as follows :
  1. Serial port drivers
  2. Network device drivers
  3. Serial ATA
  4. Flash
  5. Bus drivers
  6. Interrupt controller.
- **Serial port drivers :** Universal Asynchronous Receiver Transmitter (UART) interface is used for connecting host. RS232 serial cable is used in between host and serial port drivers. It provides support for baud rate, flow control etc.
- **Bus drivers :** Bus drivers provide hardware-specific interfaces for the corresponding hardware protocols, and are the bottom-most horizontal software layers of an operating system.
- **Network device driver :** Network device driver interface is Ethernet interface and wifi interface. This driver connects to the TCP/IP protocol stack within the operating system. It offers services like socket API.
- **Serial ATA :** Secondary storage disks are connected to this interface. This interface defines an Advanced Host Controller Interface (AHCI). The driver provides block services to a file system driver.
- **Flash :** Flash is one type of storage for block driver. Blocks are erase, write and program by using special program. Driver uses wear leveling for using all blocks

of flash. In wear leveling, it is often desirable to move the frequently changing data (so-called hot data) into the same blocks, while storing the mostly static data together in other blocks.

- **Interrupt controller :** It is integral part of the operating system and it could be considered as device driver.



- A device driver has three sides of interface :
  1. Rest of the kernel
  2. Hardware
  3. User.

Functions of Device Drivers

Accept abstract read and write requests from the device-independent layer above.

Initialize the device.

Manage power requirements and log events.

Check input parameters if they are valid.

Translate valid input from abstract to concrete terms.

Control the device by issuing a sequence of commands. The driver determines what commands will be issued.

Manages privilege : A driver allows an unprivileged application to interface with a device.

Abstracts the hardware : Device driver should abstract the underlying implementations of the hardware.

Provide security : Provide security to prevent an unauthorized process from accessing device.

Enables multiplexed access : Device driver provides mechanism for multiple applications to use a device concurrently.

Character Device Driver

Device drivers are of two types : *character devices* and *block devices*.

Character devices can be thought of as serial streams of sequential data. Examples character devices include serial ports and keyboards.

Block devices are characterized by the capability to read and write blocks of data and from random locations on an addressable medium. Examples of block devices include hard drives and USB Flash drives.

- Block and character devices are represented for user space applications as files than can be manipulated using the traditional file API.
- Linux supports loadable device drivers, it is relatively easy to demonstrate a simple device driver skeleton. A device driver is a special kind of binary module. Unlike a stand-alone binary executable application, a device driver cannot simply be executed from a command prompt.
- Character device drivers are referenced through special files in the file system. This file is stored in the /dev folder. The command "**ls-l**" is used to check the folder. The character "c" in the file listing indicates the file is a char device. It also contains device major and minor numbers.

Device driver initialization

- In order that the device driver is correctly initialized when the operating system is booted, the xxx_init ( ) routine must be executed. To ensure this happens, add the following line to the end of the chr_drv_init() function in the /usr/src/linux/driver/char/mem.c file :

```
mem_start = xxx_init(mem_start);
```

and resave the file back to disk.

Installing the driver in the Kernel

- Fig. 6.3.2 shows position of device driver in OS.
- A character device driver has to be archived into the /usr/src/linux/drivers/char/char.a library. The following steps are required to link the driver to the kernel :
  1. Put a copy of the source file in the /usr/src/linux/drivers/char directory.
  2. Edit Makefile in the same directory so it will compile the source for the driver--add xxx_drv.o to the OBJS list, which causes the make utility to automatically compile xxx_drv.c and add the object code to the char.a library archive.
  3. The last step is the recompilation of the kernel.

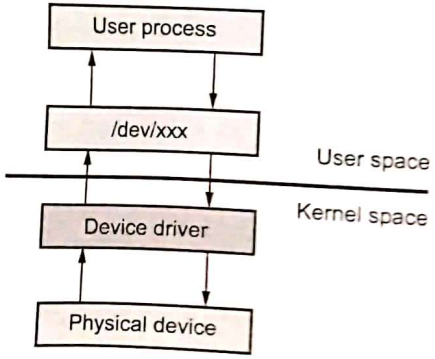


Fig. 6.3.2 Position of device driver in OS

Kernel interface of a device driver

- In order to talk to the kernel, the driver registers with subsystems to respond to events. Such an event might be the opening of a file, a page fault, the plugging in of a new USB device, etc.

User interface of a device driver

- Since Linux follows the UNIX model, and in UNIX everything is a file, users talk with device drivers through device files. Device files are a mechanism, supplied by the kernel, precisely for this direct User-Driver interface.
- Modern systems often have "loadable device drivers", which are loaded dynamically when needed. This way if a user buys a new device, no changes to the operating system are needed. When the device is installed it will be detected during the boot process and the corresponding driver is loaded.

6.3.3 Device Independent I/O Software

- The device-independent code cantains most of the I/O functionality, but not most of the code since there are very many drivers. All drivers of the same class do essentially the same thing in slightly different ways due to slightly different controllers.
- Function of device independent I/O software
  1. It provides buffering.
  2. Support for error reporting.
  3. Provides uniform interfacing for device drivers.
  4. Perform allocation and releasing dedicated devices.
  5. Software provides a device indepenentdent block size.

Uniform interfacing for device drivers

- As stated above the bulk of the OS code is made of device drivers and thus it is important that the task of driver writing not be made more difficult than needed. As a result each class of devices (e.g. the class of all disks) has a defined driver interface to which all drivers for that class of device conform. The device independent I/O portion processes user requests and calls the drivers.
- **Major device numbers** are used by the Linux system to map I/O requests to the driver code, thereby deciding which device driver to execute, when a user reads from or writes to the special file.
- The **minor device numbers** are entirely under the control of the driver writer, and usually refer to sub-devices of the device.
- The inode contains both major and minor device numbers.

- These sub-devices may be separate units attached to a controller. Thus, a disk device driver may, for example, communicate with a hardware controller (the device) which has several disk drives (sub-devices) attached.

6.3.3.1 Buffering

- Buffering is the technique by which the device manager can keep slower I/O devices busy during times when a process is not requiring I/O operations. Input/output buffering is a mechanism that improves the throughput of input and output operations. It is implemented directly in hardware and the corresponding drivers.
- *Input buffering* is the technique of having the input device read information into the primary memory before the process requests it.
- *Output buffering* is the technique of saving information in memory and then writing it to the device while the process continues execution.

Types of I/O buffering schemes

1. Single buffering
2. Double buffering
3. Circular buffering
4. No buffering.

1. Single buffering

Operating system assigns a buffer in the system portion of main memory. Fig. 6.3.3 shows single buffering.

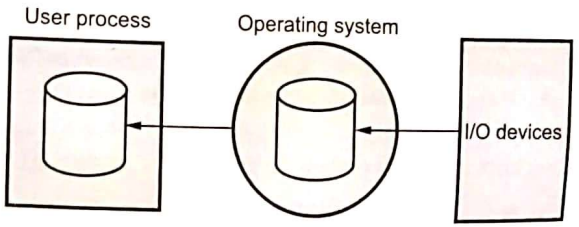


Fig. 6.3.3

Block oriented device

1. Input transfers are made to the system buffer.
2. After transferring, the process moves the block into user space and request for another block.
3. User process can be processing one block of data while the next block is being read in.

4. OS is able to swap the process out.
5. OS must keep track of the assignment of system buffers to user processes.

2. Double buffering

- There are two buffers in the system.
- One buffer is for the driver or controller to store data while waiting for it to be retrieved by higher level of the hierarchy.
- Other buffer is to store data from the lower level module.
- Double buffering is also called buffer swapping.
- Fig. 6.3.4 shows the hardware and software double buffering for bytes.

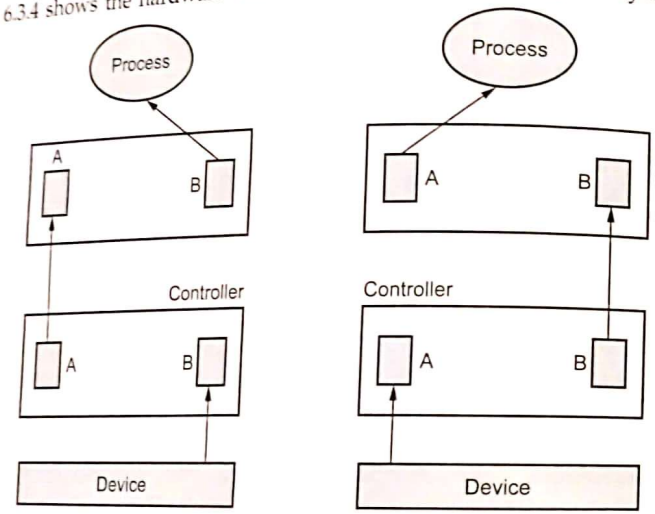


Fig. 6.3.4 Double buffering in the driver

- Double buffering improvement comes at the cost of increased complexity.
- Double buffering may be inadequate if the process performs rapid bursts of I/O.

Circular buffer

- When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer.
- In this, the producer cannot pass the consumer because it would overwrite buffers before they had been consumed.
- The producer can only fill up to buffer j–1 while data in buffer j is waiting to be consumed.

- Fig. 6.3.5 shows circular buffering.

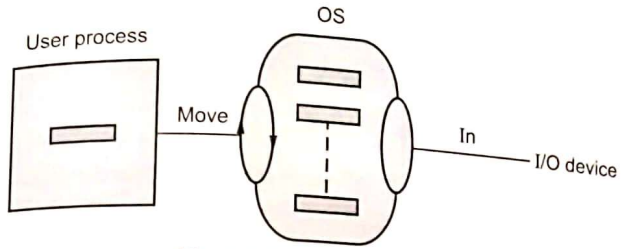


Fig. 6.3.5 Circular buffering

No buffering : There is a buffer between user process, operating system and I/O device.

6.4 Disks

- A storage device is any device used in a computer to store information. It will retain this information when the computer is switched off.
- Type of storage device used varies based on the type of data and the rate at which it is created and used.
- Examples of storage devices are hard disk, CD-ROM, magnetic tape and removable media etc.
- The computer has many types of data storage devices. Some of them can be classified as the removable data storage devices and the others as the non removable data storage devices.

1. **Hard disk :** The most common form of internal storage device used in a computer is a hard disk. A hard disk has a circular, magnetized surface on which the data is stored. A hard disk spins at a speed of between 60 and 120 revolutions per second. The data stored on the hard disk is read or written by a head that floats just above the disk (less than 0.1 mm) on a cushion of air. The surface of a hard disk is divided up into sectors and tracks. Data is stored in the 'blocks' created by the sectors and tracks. Moving data into a 'block' is called random access.
2. **Compact disk :** Compact disks can be used to store much more data than a floppy disk. Most compact disks can hold 650 megabytes of data. There are three main types of compact disk : CD-ROM, CD-R, CD-RW.
3. **Magnetic tape :** This is a cheap method of storing large amounts of data (typically 26 gigabytes) and is often used as a 'backing store' for large and mainframe computers.

4. **Removable media :** Zip drives are similar to floppy drives and use special (and rather expensive) floppy disks that can hold between 100 megabytes and 2 gigabytes of data.

6.4.1 Characteristics of Storage Device

1. When power to the device is shut off, data stored on the medium remains.
2. Storage devices are cheaper than memory.
3. They are capable of storing more data.
4. Rotational speeds have increased over time.
5. Storage devices are used for backup.

6.4.2 Hard Disk

- To store computer data, hard disks, flash memory, magnetic tape and optical media is used.
- Alternate storage for hard disk is Solid State Disks (SSD). These flash memory based devices offer a different set of tradeoffs from a standard disk. At the same time, capacity of the hard disk also increases.
- Hybrid category is introduced for storage media. It is hard disks with large flash memory buffers. Disk sizes are specified in gigabytes.
- Performances of hard disk and SSD technology is given below :

Characteristics	Hard disk	SSD
Size	Terabytes	Gigabytes
Random access time	8 ms	0.25 ms
Sequential read	100 MB/s	250 MB/s
Random read	2 MB/s	250 MB/s
Cost	$0.10 /GB	$3 /GB
Reliability	Moderate	Unknown
Limited writes	No	Yes

- Hard disk contains several rotating platters coated with magnetic film. They are read and written by tiny skating heads that are mounted on a metal arm that swings back and forth to position them. Heads float close to the surface of the platters but do not actually touch.
- A hard disk is really a set of stacked "disks," each of which, like phonograph records, has data recorded electromagnetically in concentric circles or "tracks" on

the disk. A "head" writes or reads the information on the tracks. Two heads, one on each side of a disk, read or write the data as the disk spins. Each read or write operation requires that data be located, which is an operation called a "seek."

- A hard disk/drive unit comes with a set rotation speed varying from 4500 to 7200 rpm. Disk access time is measured in milliseconds. Although the physical location can be identified with cylinder, track and sector locations, these are actually mapped to a Logical Block Address (LBA) that works with the larger address range on today's hard disks.
- Mean Time Between Failures (MTBF) is the predicted elapsed time between inherent failures of a system during operation. MTBF can be calculated as the arithmetic mean (average) time between failures of a system.
- Each disk drive is managed by a controller. Each type of controller can support a fixed number of drives. SCSI controllers support up to seven disks per controller or up to 15 disks per controller.
- Each disk is assigned a drive address. This address is set by a switch, a dial or jumpers on the disk or by the physical location of the disk. Some SCSI devices, such as RAID's, have an additional identifying number called a logical unit number. It is used to address disks within the device.
- Fig. 6.4.1 shows physical disk.

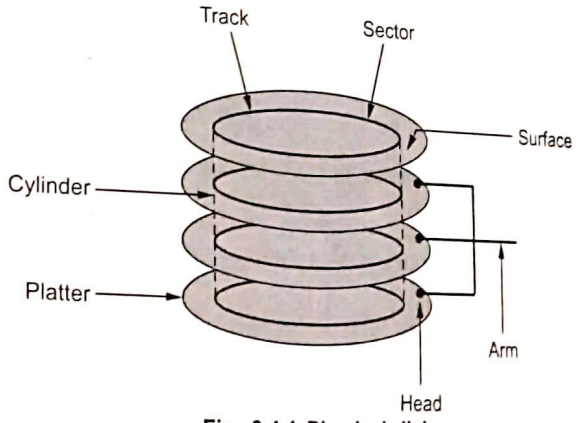


Fig. 6.4.1 Physical disk

- A disk consists of circular plates called **platters**. Each platter has an upper and lower oxide-coated surface. Recording heads, at least one per surface, are mounted on arms that can be moved to various radial distances from the center of the platters. The heads float very close to the surfaces of the platters, never actually touching them, and read and record data as the platters spin around.
- A ring on one surface is called a **track.** Each track is divided into disk blocks. Sometimes called sectors, these physical blocks on a disk are different from file system blocks.
- Formatting a disk divides the disk into tracks and disk blocks that can be addressed by the disk controller, writes timing marks, and identifies bad areas on the disk.

- SCSI disk drives are shipped preformatted. They do not require formatting at any time. Bad block handling is performed automatically by SCSI disks. Bad blocks are areas of a disk that cannot reliably store data.

Platter

- Platter is a circular, metal disk that is mounted inside a hard disk drive. Several platters are mounted on a fixed spindle motor to create more data storage surfaces in a smaller area.
- The platter has a core made up of aluminum or glass substrate, covered with a thin layer of Ferric oxide or cobalt alloy. On both sides of the substrate material, a thin coating is deposited by a special manufacturing technique. This, thin coating where actual data is stored is the media layer.
- Attributes of platter
  1. It is a rigid, round disk this is coated with magnetically sensitive material.
  2. Data is stored in binary code.
  3. It is encoded by polarizing magnetic areas on the disk surface.
  4. Data can be written to and read from both surfaces of a platter.
  5. A platter's storage capacity varies across drives.

Tracks

- Each platter is broken into thousands of tightly packed concentric circles, known as tracks. These tracks resemble the structure of annual rings of a tree.
- All the information stored on the hard disk is recorded in tracks. Starting from zero at the outer side of the platter, the number of tracks goes on increasing to the inner side.
- Each track can hold a large amount of data counting to thousands of bytes.

Sectors

- Each track is further broken down into smaller units called sectors. As sector is the basic unit of data storage on a hard disk. Disk contains concentric **tracks**. Tracks are divided into **sectors**. A sector is the smallest addressable unit in a disk.
- Fig. 6.4.2 shows surface of disk showing tracks and sectors.
- A single track typically can have thousands of sectors and each sector can hold more than 512 bytes of data. A few additional bytes are required for control structures and error detection and correction.

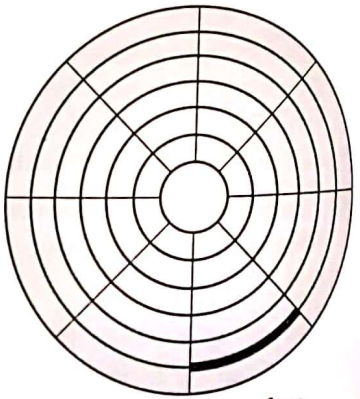


Fig. 6.4.2 Disk surface

Clusters : Sectors are often grouped together to form clusters.

Read / Write Heads

- The heads are an interface between the magnetic media where the data is stored and electronic components in the hard disk. The heads convert the information, which is in the form of bits to magnetic pulses when it is to be stored on the platter and reverses the process while reading.
- The heads are the most sophisticated part of the hard disk. Each platter has two read/write heads, one mounted on the top and the other one at the bottom. These heads are mounted on head sliders, which are suspended at the ends of head arms.
- The head arms are all fused into a singular structure called actuator, which is responsible for their movement. Drives rotate at 60 to 200 times per second.
- **Transfer rate** is rate at which data flow between drive and computer.
- **Positioning time (random-access time)** is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head.
- **Head crash** results from disk head making contact with the disk surface.
- Each platter (disc-shaped) is coated with magnetic material on both surfaces. All platter surfaces has *arm* extended from fixed position. Tip of the arm contains read/write *head* for reading or writing data.
- The arm moves the heads from the spindle edge to the edge of the disc.
- When a program reads a byte from the disk, the operating system locates the surface, track and sector containing that byte, and reads the entire sector into a special area in main memory called **buffer**.
- The bottleneck of a disk access is moving the read/write arm.
- A **cylinder** is the set of tracks at a given radius of a disk pack. A cylinder is the set of tracks that can be accessed without moving the disk arm. All the information on a cylinder can be accessed without moving the read/write arm.
- Fig. 6.4.3 shows moving-head disk mechanism.
- The arm assembly is moved in or out to position a head on a

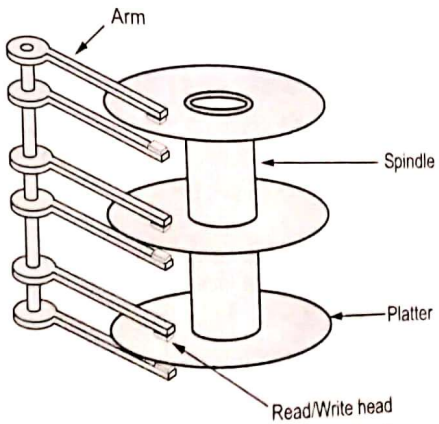


Fig. 6.4.3 Moving-head disk mechanism

desired track. Tracks under heads make a cylinder. Only one head reads/writes at any one time. Block size is a multiple of sector size.

- Disks can be removable. Drive attached to computer via I/O bus. Buses vary, including EIDE, ATA, SATA, USB, Fibre channel, SCSI etc.
- Host controller in computer uses bus to talk to disk controller built into drive or storage array.
- Disk controllers typically embedded in the disk drive, which acts as an interface between the CPU and the disk hardware. The controller has an internal cache that it uses to buffer data for read/write requests.

Zone bit recording

- Also known as multiple zone recording or zone-CAV recording. Disk divided into zones.
- Cylinders in different zones have a different number of sectors. Number of sectors in a particular zone is constant. Data is buffered so the data rate to the I/O interface is constant. Fig. 6.4.4 shows zone bit recording.

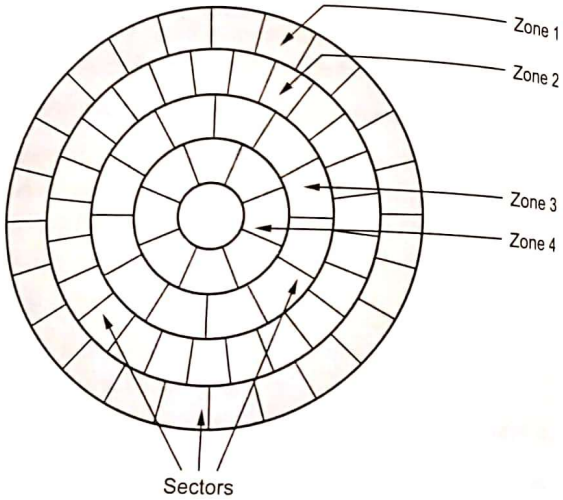


Fig. 6.4.4 Zone bit recording

- Zoned-bit recording uses the disk more efficiently. It groups tracks into zones that are based upon their distance from the center of the disk. Each zone is assigned an appropriate number of sectors per track. This means that a zone near the center of the platter has fewer sectors per track than a zone on the outer edge.

Solid state disks

- SSD is a storage device that is based on semiconductors rather than rotating magnetic platters. Most SSDs are based on NAND Flash chips because they are fast, highly reliable, widely available and are non-volatile, meaning they save data even without a power source.

- SSDs use the same Serial ATA (SATA) or IDE interface as hard drives, making them functionally identical.
- Solid-state storage technology typically provides faster system performance than traditional magnetic media drives. In addition, there are no moving parts in SSDs and therefore the risk of mechanical failure is near zero. Solid-state drives also provide improved overall system responsiveness while consuming much less power than a traditional hard disk drive.
- A flash-based SSD typically uses a small amount of DRAM as a cache, similar to the cache in hard disk drives. A directory of block placement and wear leveling data is also kept in the cache while the drive is operating.
- Each page of flash memory in an SSD can be rewritten only a limited number of times. To limit the wear on any given page, the SSD firmware maintains a mapping table and distributes writes across all the drive' pages. This remapping is invisible to operating system.
- Flash memory pages must be erased before they can be rewritten. Erasing is separate operation that is slower than writing. Rebuilding a buffer of erased pages is harder than it might seem because filesystems typically do not mark or erase data blocks they are no longer using.
- Standard size of the disk block is 512 bytes, but that size is too small for filesystems to deal with efficiently. Filesystem manages the disk in terms of clusters of 1 kilobytes to 8 kilobytes in size. The translation layer maps filesystem clusters into ranges of disk blocks for reads and writes.
- SSD only can read or write data in 4 kilobytes pages, file system cluster boundaries and SSD page boundaries should coincide.

6.5 RAID

GTU : Summer-13,14

- RAID means Redundant Array of Independent Disks. It consists of multiple disks. RAID provides data protection against Hard disk failures.
- RAID implementation improves the I/O performance of storage systems by storing data across multiple hard disks. Data is organized in multiple ways because of many hard disks. Data is stored on several disks instead of a single disk
- RAID capacity depends upon the type of RAID array user build and the number, size of disk drives.
- RAID consists of 7 layers. RAID is implemented in two ways : software RAID and Hardware RAID.

6.5.1 Software RAID

- Software RAID runs on the server processor (CPU). It is implemented at the operating system level. Dedicated hardware is not required to manage the Software RAID.
- It consumes server memory and CPU, so performance of server is decreases. Software RAID is simpler to implement than hardware RAID. It does not support all levels of RAID.
- The upgradation of operating system means upgradation of software RAID so it may create problem for data processing.
- Software RAID also works with cheaper IDE disks as well as SCSI disks.

6.5.2 Hardware RAID

- Hardware RAID is implemented in two ways :
  1. Specialized hardware controller on the server.
  2. Specialized hardware controller on the array.
- Specialized hardware controller is required to install on server machine or on the array. It is not dependent on operating system. It does not occupy any server memory.
- Host CPU can execute applications while the array adapter's processor simultaneously executes array functions.
- In hardware RAID on the server, RAID controller interacts with the hard disks using the PCI bus. Some of mother board manufacture provides the facility of hardware RAID. It reduces the cost but not flexible.
- Array based hardware RAID uses external controller. External controller acts as the interface between the host and the hard disks.

6.5.3 RAID Array Components

- RAID array consists of multiple hard disk with required hardware and software. Hardware and software is required to implement the RAID array. It also consists of physical array and logical array.
- Physical array contains the fixed number of hard disk, power supply and supporting hardware.
- Logical array contains subset of disks within a RAID array. It is also called RAID set or RAID group. Logical array are comprised of logical volumes.
- Operating system identifies the logical volume. Actually logical volumes mean physical hard disk. Physical hard disk is managed by RAID controller. RAID level

depends on the number of hard disk used in a logical array. Fig. 6.5.1 shows the RAID array components.

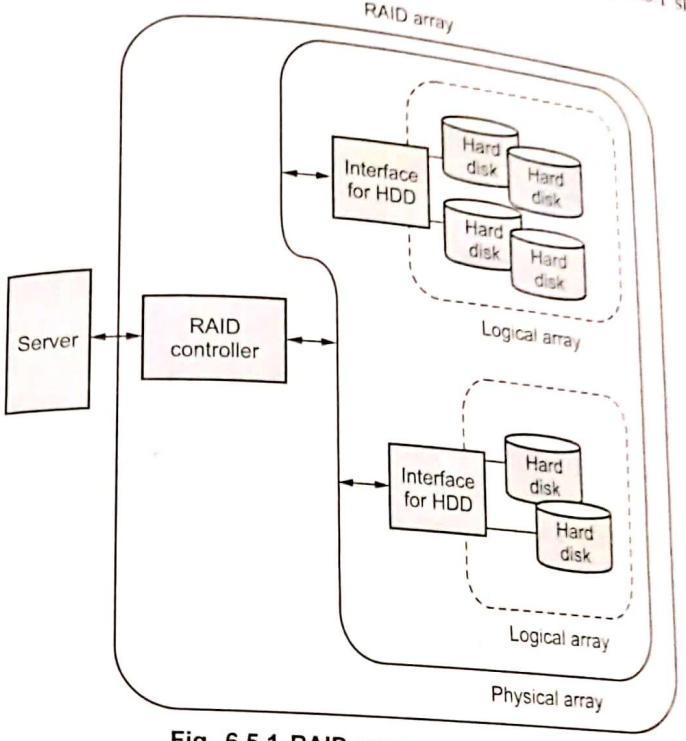


Fig. 6.5.1 RAID array components

6.5.4 RAID Levels

1. RAID Level 0

- In RAID level 0, data is split into blocks that are written across all the hard disk in the array. It utilize full storage capacity of the hard disk. It creates one large virtual disk from a number of smaller disk.
- Storage is grouped into logical unit called strips. Each hard disk in a RAID 0 stripe must be of the same size, since I/O requests are interleaved to read or write to multiple disks in parallel.
- RAID 0 is not suitable for storing sensitive data. If one hard disk fails, all data is lost. It is less reliable than single hard disk systems.
- Distribution of data over multiple disks is called striping.
- Suppose system wants to write data of 256 K to one disk, RAID 0 can simultaneously write 64 K (64 K × 4 = 256 K) to each of the four disk in the

array. RAID level 0 writes consecutive strips over the hard disk in round robin method.

- If user gives the command to read the data block consisting of four consecutive strips starting at a strip boundary. The RAID controller will break this command up into four separate commands for each hard disk. These four commands works in parallel.
- RAID level 0 is suitable for larger requests. It is suitable for non-critical data and data which changes rarely.

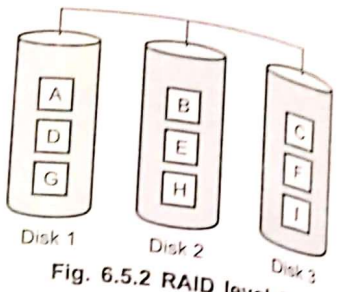


Fig. 6.5.2 RAID level 0

2. RAID Level 1

- RAID level 0 and level 1 works with strips of sector.
- RAID level 1 is also known as mirroring. It writes all data two times, i.e. once to the main hard disk and once to a mirror set hard disk. Because of mirroring, it improves the fault tolerance. Each drive in a mirror contains an identical copy of the original disk data.
- Fig. 6.5.3 shows RAID level 1. For read operation, it reads either copy of the data. To implement RAID level 1, it requires minimum two hard disks.
- RAID level 1 provides excellent fault tolerance facility. RAID controller uses the mirror drive for data recovery and continuous operations. Data recovery means installing new hard disk and copying data from the mirror hard disk.
- There is high storage overhead and it increases cost per unit capacity.
- RAID level 1 is suitable for applications that require high availability.

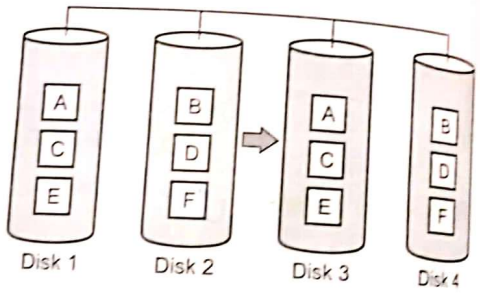


Fig. 6.5.3 RAID level 1

3. RAID Level 2

- RAID level 2 uses concept of parallel access technique. It works on the word(byte) level. So each strip stores one bit. It takes data striping to the extreme, writing only 1 bit per strip, instead of in arbitrary size blocks. For this reason, it requires a minimum of 8 surfaces to write data to the hard disk.

- RAID level is also known as memory style error correcting code organization. It generates hamming code for data protection and stores it on the extra hard disk. Hamming code uses parity bits to check for errors in data transmission from disks and correct it if possible. Check disk is used to store a hamming code of each data stripe. This code can correct as well as detect data errors and permits data recovery without complete duplication of data. Fig. 6.5.4 shows RAID level 2.

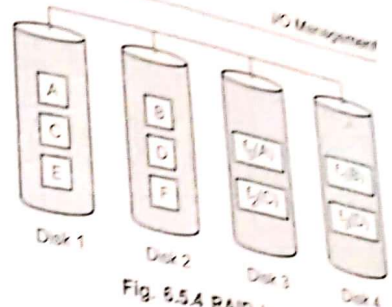


Fig. 6.5.4 RAID level 2

- In RAID level 2, strips are very small, so when a block is read, all disks are accessed in parallel. Hamming code generation is time consuming, therefore RAID level 2 is too slow for most commercial applications.
- RAID level 2 is also sometimes referred to as RAID level 0+1. It combines the benefits of both striping and RAID level 1.
- On a single write operation, all data disks and parity disks must be accessed and read operation, all hard disks are simultaneously accessed. At the present time there are no manufacturers of RAID level 2 arrays.

4. RAID Level 3

- RAID level 3 is also called bit interleaved parity organization. It uses single parity bit for error correction and for error detection. It always reads and writes complete stripes of data across all hard disk, as the hard disk operate in parallel.
- RAID level 3 uses only one extra hard disk drive to store simple parity bits. It uses XOR error correcting codes. Parity is calculated using following formula:

$$\text{Parity} = b_0 \text{ XOR } b_1 \text{ XOR } b_2 \text{ XOR } b_3 \text{ XOR } b_4 \text{ XOR } b_5 \text{ XOR } b_6 \text{ XOR } b_7$$

- Fig. 6.5.5 shows RAID level 3. If a hard disk drive fails, the data on the failed drive can be re-created by using the above equation, and substituting the parity bit in for the failed bit.
- RAID level 3 reduces the storage overheads. Read and write operation of a byte are spread over multiple disks with N-way stripping of data. Data transfer rate increases.
- RAID level 3 is used in video streaming. Here large sequential data access is required.

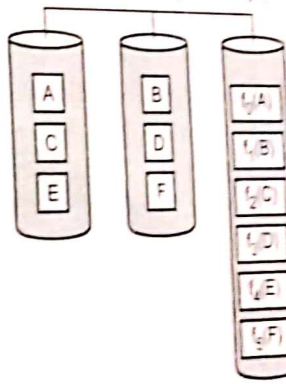


Fig. 6.5.5 RAID level 3

5. RAID Level 4

- RAID level 4 is like RAID level 2 and it would offer poor performance if implemented. It is also called block interleaved parity organization. It uses block level stripping. Fig. 6.5.6 shows RAID level 4.

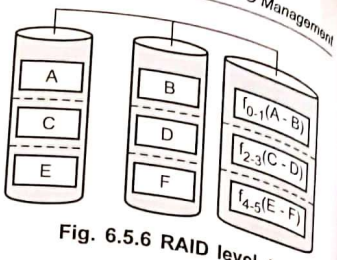


Fig. 6.5.6 RAID level 4

- RAID level 4 maintains separate hard disk for parity bit. Parity bit is stored on the block. Parity block improve the fault tolerance.
- Let us consider following example : Assume 5 hard disk with 4 data drive and 1 parity drive. Suppose we want to write strip 3, then following thing happens :
  1. Read the data currently occupying strip 3, and the parity strip.
  2. XOR the old data with the new data to get the new parity.
  3. Write the new data to strip 3.
  4. Update the parity strip.
- If there were write requests pending for strips 1 and 4, each would have to wait until the previous write completed since each need the updated state of the parity. Previous levels could do multiple writes concurrently as no strips data was dependent upon the data on any other strip.
- If any of the hard disk drive crash, the lost byte data can be recomputed from the parity drive by reading the entire set of drives.
- RAID level 4 provides good read throughput and less write throughput.
- There are no commercial implementations of RAID level 4 because of poor performance.

6. RAID level 5

- RAID level 5 also called block interleaved distributed parity. It remove the bottleneck of RAID level 4.
- RAID level 5 arrays are striped at the block level and uses XOR ECC parity. Parity is distributed across all hard disk. The distribution of parity strips across all hard disk avoids the potential input-output bottleneck of the single parity disk.

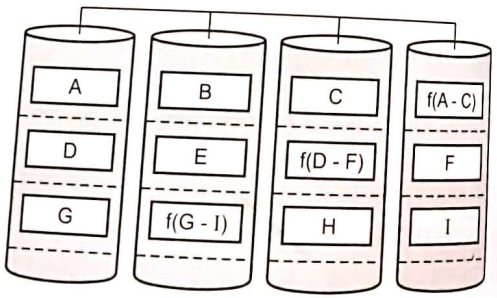


Fig. 6.5.7 RAID level 5

- Fig. 6.5.7 shows RAID level 5.

- RAID level 5 has been a commercial success because it offers the best protection for the least cost, and is more commonly used than any of the other RAID systems.
- RAID level 5 used for implementation of RDBMS.

7. RAID Level 6

- RAID level 6 extends RAID level 5 by distributing two parity blocks per stripe. It is also called P + Q redundancy schemes.
- RAID level 6 attempts to protect against multiple-disk failure by using two sets of error-correction strips for each row of drives. One strip holds the parity, and the other stores Reed-Solomon error-correcting codes.
- Hard disks are considered to be in a matrix formation and parity is generated for each row and column of the matrix. Multidimensional parity is calculated and distributed among the hard disks in the matrix.
- RAID level 6 offers good protection against the entire array becoming useless while a single failed disk is rebuilt, it is not often used commercially because it has 2 performance drawbacks.
- There is sizeable overhead involved in generating Reed-Solomon code. It requires twice as many read/write operations to update the error-correcting codes.
- RAID level 6 provides high data availability.

6.5.5 Comparison of RAID Levels

Parameters	RAID 0	RAID 1
Minimum hard disk required	2	2
Protection for data	None	Single drive failure
Description	Data striping without redundancy	Disk mirroring
Merits/strength	High performance	Very high performance ; very high data protection
Demerits	No data protection ; one drive fails then all data lost	High redundancy cost overhead.
Application	Data logging, Real-Time rendering,	Used in operating system.

University Questions

1. Short note on RAID. GTU : Summer-13, Marks 4
2. Write short note : RAID levels. GTU : Summer-14, Marks 7

6.6 Disk Performance Parameters

- The actual details of disk I/O operation depend on the :
  1. Computer system
  2. Operating system
  3. Nature of the I/O channel and disk controller hardware
- Currently, disks are at least four orders of magnitude slower than main memory. Fig. 6.6.1 shows a general timing diagram of disk I/O transfer.

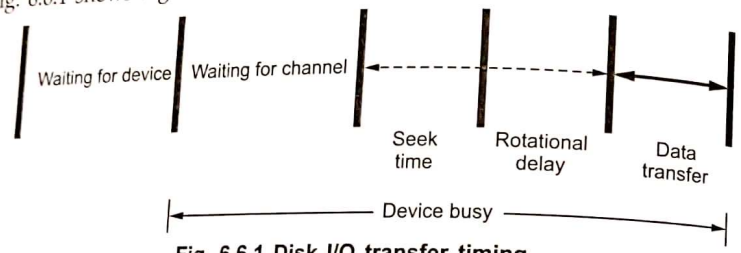


Fig. 6.6.1 Disk I/O transfer timing

- The disk is rotating at constant speed when it perform read or write operation. To read or write the head must be positioned at the desired track and at the beginning of the desired sector on that track. Head is moving from one track to other track for selecting proper track.
- Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system.
- Following parameters are used for disk performance :
  1. Seek time
  2. Rotational delay
  3. Access time

1. **Seek time :** The time it takes to position the head at the track is known as **seek time**.

   Seek time = Number of track traversed × Disk drive constant + Startup time

   Seek time do not apply to device with fixed read/write heads.

2. **Rotational delay :** The time it takes for the beginning of the sector to reach the head is known as rotational delay. It is also known as search time.

3. **Access time :** The sum of the seek time and the rotational delay equals the access time.

   Access time = Seek time + Rotational latency

- **Transfer time :**

$$\text{Transfer time} = \frac{\text{Number of bytes to be transferred}}{\text{Number of bytes on the track} \times \text{Rotation speed}}$$

- **Disk capacity :**

$$\text{Disk capacity} = \text{Number of cylinders} \times \text{Number of heads} \times \text{Number of sectors per track} \times \text{Number of bytes per track}$$

Example 6.6.1 *What is the maximum size of disk which contains following parameters ? Cylinder = 1024, heads = 16 and sectors per track is 63.*

Solution :

$$\text{Total size of disk} = \text{Number of cylinders} \times \text{Number of heads} \times \text{Number of sectors/track} \times \text{Number of bytes/sector.}$$

$$= 1024 \times 16 \times 63 \times 512 = 528\ 48$$

Example 6.6.2 *A disk has 19456 cylinders, 16 heads and 63 sectors/tracks. The disk spins at 5400 r.p.m. Seek time between adjacent tracks is 2 ms. Assuming the read/write head is already positioned at track 0. How long does it take to read the entire disk ?*

Solution : 1) Each track can be read in one revolution.

2) 11.11 ms is required to read one track.

3) For reading all 19456×16 tracks, approximately 3459 seconds is required.

4) Seek time = 19456 − 1 × 2 = 39 sec.

5) Total time is 3498 sec.

6.7 Disk ARM Scheduling Algorithm

GTU : Summer-13

- Disk scheduling algorithms are used to reduce the total seek time of any request. I/O request issues a system call to the operating system. If desired disk drive or controller is idle then the request is served immediately. If device is bust then the request for service will be placed in the queue of pending requests.
- When one request is completed, the operating system has to choose which pending request to service next.
- The processor is much faster than the disk, so it's highly likely that there will be multiple outstanding disk requests before the disk is ready to handle them.

Because disks have non-uniform access times, re-ordering disk requests can greatly reduce the latency of disk requests.

- Multiple requests to disk will arrive while one is being serviced. Disk scheduling increases the disk's bandwidth.
- To service a request, a disk system requires that the head be moved to the desired track, then a wait for latency and finally the transfer of data. In the following section we will examine several scheduling algorithms.

1. FIFO
2. Shortest Seek Time First (SSTF)
3. SCAN
4. Circular SCAN
5. LOOK
6. Circular LOOK

6.7.1 First In First Out

- FIFO is also called first come first served method. Requests are processed in queue order.
- FCFS has a fair policy in the sense that once a request has arrived, its place in the schedule is fixed irrespective of arrival of a higher priority request.
- The requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38 and 184. Starting track at 100 and total number of track is 200. FCFS process the entire request in sequential order.
- FCFS would begin at track 100, move 45 tracks to 55, and move 3 tracks to 58 and so on. It is shown in Fig. 6.7.1. The Y-axis corresponds to the tracks on the disk. The X-axis corresponds to time or the number of tracks traversed. Starting at track 100.

Next accessed track is as above.

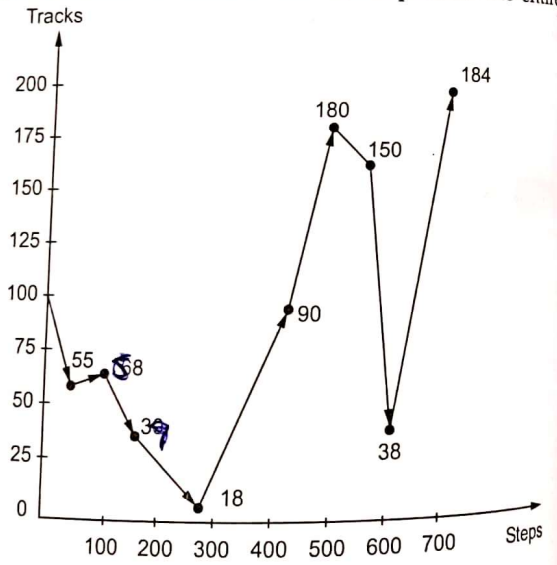


Fig. 6.7.1 FCFS

Next track accessed	Number of track traversed
55	45
58	03
39	19
18	21
90	72
160	70
150	10
38	112
184	146

$$\text{Average seek length} = \frac{45+3+19+21+72+70+10+112+146}{9} = \frac{498}{9} = 55.333$$

- FCFS is simple to implement. But it does not provide fast services. It can not take special action to minimize the overall seek time.

Advantages :

1. Simple and easy to implement.
2. Suitable for light loads.

Disadvantages :

1. FCFS does not provide fast services.
2. Do not maximize throughput.
3. May involve lots of unnecessary seek distance.

6.7.2 Shortest Seek Time First

- SSTF select the next request at the one requiring the minimum seek time from the current position.
- In Shortest-Seek-Time-First (SSTF) scheduling priority is given to those processes which need the shortest seek, even if these requests are not the first ones in the queue. It means that all requests nearer to the current head positions are serviced together before moving head to distant tracks.

- Select the disk I/O request that requires the least movement of the disk arm from its current position and always choose the minimum seek time.

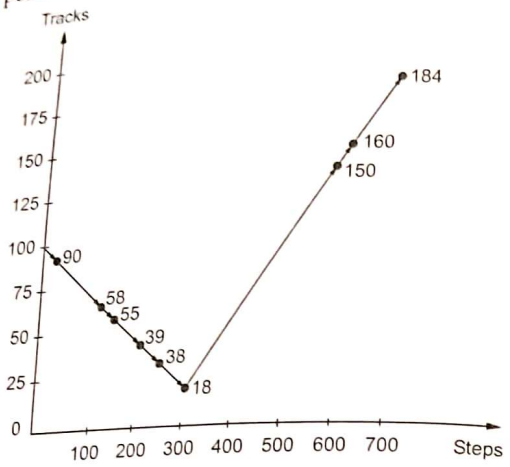


Fig. 6.7.2 SSTF

Next track accessed	Number of track traversed
90	10
58	32
55	03
39	16
38	01
18	20
150	132
160	10
184	24

Average seek length $= \frac{10+32+3+16+1+20+132+10+24}{9} = \frac{248}{9} = 27.55$

- The only tricky part is if there are two jobs with the same distance. In this case, some kind of tie-breaking needs to be employed to pick one. For instance, you could just use a random number to effectively flip a coin and pick one.
- SSTF does not ensure fairness and can lead to indefinite postponement because its seek pattern tends to be highly localized.
- Under heavy load, SSTF can prevent distant requests from ever being serviced. This phenomenon is known as starvation. SSTF scheduling is essentially a form of shortest job first scheduling. SSTF scheduling algorithm are not very popular because of two reasons.
  1. Starvation possibly exists. 2. It increases higher overheads.

Advantages :

1. Throughput is better than FCFS.
2. SSTF minimize the response time.
3. Less number of head movements.

Disadvantages :

1. One major issue is STARVATION.
2. Localize under heavy load.

6.7.3 SCAN

- SCAN is also called elevator algorithm.
- The next request scheduled is closest to current request but in one particular direction. All requests in other direction are put at the end of the list. SCAN services tracks in only one direction i.e. either increasing or decreasing track number.
- When SCAN reaches the edge of the disk (or track 0), it reverses direction.
- If any request comes on its way it will be serviced immediately, while request

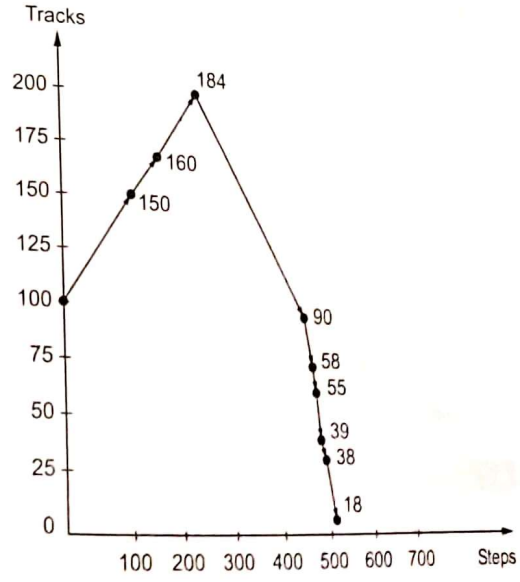


Fig. 6.7.3 SCAN

arriving just behind the head will have to wait until disk head moves to the end of the disk, reverses direction and returns before being serviced.

- SCAN is called elevator because an elevator continues in one direction servicing requests before reversing direction.
- Consider the example in which the requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38, and 184. Starting track at 100 and total number of track is 200. Head is moving towards increasing track number.

Current head position = 100

Head is moving towards increasing number of track.

Next track accessed	Number of track traversed
150	50
160	10
184	24
90	94
58	32
55	03
39	16
38	01
18	20

Average seek length $= \frac{50+10+24+94+32+3+16+01+20}{9} = \frac{250}{9} = 27.77$

Advantages :

1. SCAN eliminates starvation.
2. Throughput similar to SSTF.

Disadvantages :

1. Increase overhead.
2. Needs directional bit.

6.7.4 Circular SCAN

- C-SCAN also moves the head in one direction, but it offers fairer service with more uniform waiting times.

- It moves the head from one end of the disk to the other end, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- C-SCAN treats the cylinders as a circular list that wraps around from the last cylinder to the first one. It scans in cycles, always increasing or decreasing order.
- Fig. 6.7.4 shows C-SCAN. Consider the example in which the requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38 and 184. Starting track at 100 and total number of track is 200. Head is moving towards increasing track number.

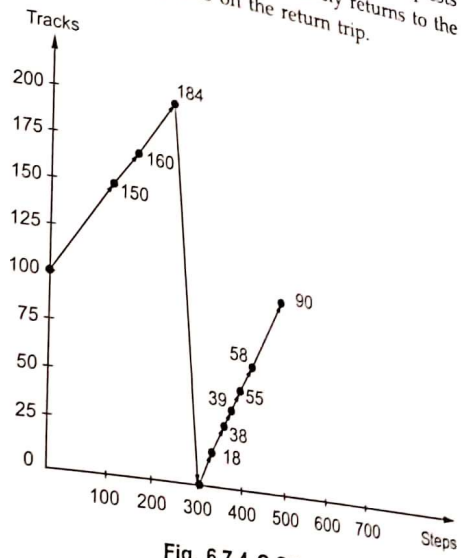


Fig. 6.7.4 C-SCAN

Next track accessed	Number of track traversed
150	50
160	10
184	24
0	184
18	18
38	20
39	1
55	16
58	3
90	32

Average seek length $= \frac{50+10+24+184+18+20+1+16+3+32}{10} = \frac{358}{10} = 35.8$

6.7.5 LOOK Scheduling

- Start the head moving in one direction. Satisfy the request for the closest track in that direction when there is no more request in the direction, the head is traveling, reverse direction and repeat. This algorithm is similar to SCAN, but unlike SCAN, the head does not unnecessarily travel to the innermost and outermost track on each circuit.
- Fig. 6.7.5 shows the look scheduling algorithm. For example, the disk request queue contains a set of references for blocks on tracks 76, 124, 17, 269, 201, 29, 137 and 12.

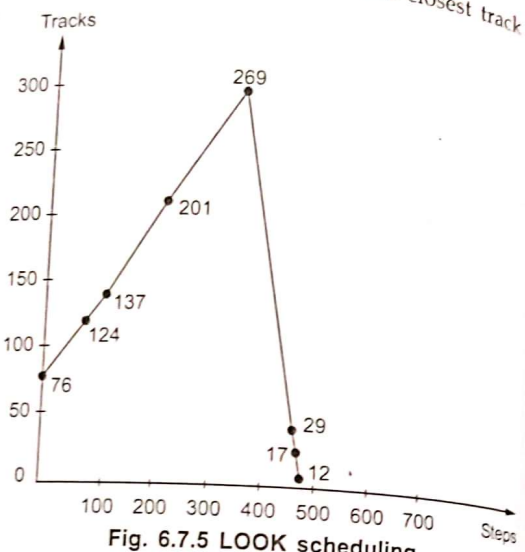


Fig. 6.7.5 LOOK scheduling

Current head position = 76

Next track accessed	Number of track traversed
124	48
137	13
201	64
269	68
29	240
17	12
12	5

Average seek length $= \frac{48+13+64+68+240+12+5}{10} = \frac{450}{7} = 64.28$

6.7.6 C-LOOK

- C-LOOK is same as C-SCAN except the head stops after servicing the last request in the preferred direction, then services the request to the cylinder nearest the opposite side of the disk.
- **Example :** Consider the example in which the requested tracks in the order received are : 23, 89, 132, 42, 187. Starting track at 100 and there are 200 cylinders numbered from 0 - 199.

Next track accessed	Number of track traversed
89	11
42	47
23	19
187	164
132	55

Average seek length $= \frac{11+47+19+164+55}{5} = \frac{296}{5} = 59.2$

Example 6.7.1 *Disk requests come in to the driver for cylinders 10, 22, 20, 2, 40, 6 and 38. A seek takes 6 msec per cylinder moved. How much seek time needed for FCFS, Closet cylinder next algorithm? Initially arm is at cylinder 20.*

GTU - Summer-13, Marks 7

Solution : **Initially arm is at cylinder 20.**

First-come first serve		Closet cylinder next algorithm	
Next Request Accessed	Number of head movement	Next Request Accessed	Number of head movement
10	20 – 10 = 10	20	20 – 20 = 0
22	22 – 10 = 12	22	22 – 20 = 2
20	22 – 20 = 2	10	22 – 10 = 12
2	20 – 2 = 18	6	10 – 6 = 4
40	40 – 2 = 38	2	6 – 2 = 4
6	40 – 6 = 36	38	38 – 2 = 36

38	38 – 6 = 32	40	40 – 38 = 2
Total number of head movement	148		60

Total seek time for FCFS = 148 × 6 = 888 msec

Total seek time for Closet cylinder next = 60 × 6 = 240 msec

University Question

1 *Disk requests come in to the disk for cylinders 10, 22, 20, 2, 40, 6 and 38. A seek take 6 msec per cylinder move. How much seek time is for closest cylinder next algorithm ? Initially arm is at cylinder 20.* **GTU : Summer-13, Marks 7**

❑❑❑

7 File Management

Syllabus

File concept, Access methods, File types, File operation, Directory structure, File system structure, Allocation methods (Contiguous, Linked, Indexed), Free-space management (Bit vector, Linked list, Grouping), Directory implementation (Linear list, Hash table), Efficiency and performance.

Contents

7.1 Basic File Concept ..... Winter-13, Summer-14, Marks 7
7.2 File Access Methods
7.3 File Directory and Disk Structure ..... Winter-13, Marks 7
7.4 File System Mounting
7.5 File Sharing ..... Winter-13, Marks 4
7.6 Record Blocking
7.7 File System Structure
7.8 File System Implementation
7.9 Log Structured File Systems
7.10 Virtual File Systems
7.11 Directory Implementation
7.12 Allocation Methods ..... Winter-13, Summer-14, Marks 7
7.13 Free Space Management
7.14 Efficiency and Performance

7.1 Basic File Concept

GTU : Winter-13, Summer-14

- Files are the building blocks of any operating system. Permanent storage of information and data is a file. File provides an easy means of information sharing. The operating system is not interested in the information stored in the files. Operating system map files with physical devices.
- An unstructured sequence of data means a file. It is also logical units of information created by processes. Information stored in the files must be persistent. File is any organized information and stored within the secondary storage. It is physical view of a file.
- A program which user prepare is a file. File represents programs and data. The output from complier may be an object code file or an executable file.
- Content or information/data of file is decided by the creator of the file. File contains various types of data like source program, object program, text and numeric data, images, graphs, sounds, payroll records, dead-stock records etc.

7.1.1 File Naming

- File store information on the disk. It is an abstract mechanism. File name is given, when process creates a file. File name continues to exist even after closing a file. It is accessed by other process.
- File name rule is changed according the operating system. All operating system allowed upto 8 string as a file name. Special characters and numbers are allowed to use as a file name. Many file systems support names as long as 255 characters.
- File names contains two parts. It is separated by a period. For example : **sorting.c**
- In the file name, after period is called file extension. File extension indicates something about a file. UNIX operating system support more than one file extension. It is up to the user, for extending the file extension.
- MS-DOS support only one file extension. File extension after period contains upto three characters. Following is the list of file extension.

Sr. No.	File extension	Remarks
1.	.c	C source program file
2.	.exe	Executable file
3.	.bak	Backup file
4.	.hlp	Help file

5.	.pdf	Portable document format file
6.	.txt	General text file
7.	.zip	Compressed file
8.	.gif	Graphics interchange format image file
9.	.html	WWW html file
10.	.mp3	Music encoded in MPEG layer 3 audio file format
11.	.ps	Postscript file

7.1.2 Types of File

- File type represents the internal structure of the file. According to the structure, file is divided into certain types : text, source, executable and object file.
  1. **Text file :** It is sequence of character which organized into lines. Text file is a regular file.
  2. **Source file :** It contains sequence of subroutines, procedure and functions.
  3. **Executable file :** It contains a series of code sections. This file is input to the loader and loader loads this file into memory and then execute.
  4. **Object file :** Object file is input to the linker. An Object file is a binary file that the compiler creates that converts the source to object code.

7.1.2.1 File Types

- File is divided into following types :
  1. Ordinary file
  2. Directory file
  3. FIFO file
  4. Character device file
  5. Block device file

Ordinary file

- Also called regular file. It contains only data as a stream of characters. An ordinary file cannot contain another file, or directory.
- It is either text file or a binary file.
- Examples of ordinary files include simple text files, application data files, files containing high-level source code, executable text files, and binary image files.
- **Text file :** It contains only printable characters.

- **Binary file :** It contains printable and unprintable characters that cover the entire ASCII range.
- Regular files may be created, browsed through, and modified by various means such as text editors.

Directory file

- A directory file is like a folder that contains other files, including subdirectory files.
- Directory files act as a container for other files, of any category. Directory files don't contain data in the user sense of data; they merely contain references to the files contained within them.
- Directory is created in UNIX by the *mkdir* command.
- The UNIX directory is considered empty if it contains no other files except the "." and ".." .
- Directory is removed by using *rmdir* command. Content of directory is displayed by ls command.

Device file

- Special files are also known as device files. In UNIX all physical devices are accessed via device files; they are what programs use to communicate with hardware.
- Files hold information on location, type, and access mode for a specific device.
- There are two types of device files; **character** and **block**.
- **Block device** files are used to access block device I/O. Block devices do buffered I/O, meaning that the data is collected in a buffer until a full block can be transferred.
- **Character device** files are associated with character or raw device access. They are used for un-buffered data transfers to and from a device. Rather than transferring data in blocks the data is transferred character by character.
- One transfer can consist of multiple characters.
- Some devices, such as disk partitions, may be accessed in block or character mode. Because each device file corresponds to a single access mode, physical devices that have more than one access mode will have more than one device file.
- Device files are found in the */dev* directory. Each device is assigned a major and minor device number.
- The major device number identifies the type of device, i.e. all SCSI devices would have the same number as would all the keyboards.

- The minor device number identifies a specific device, i.e. the keyboard attached to this workstation.
- Device files are created using the *mknod* command.
- Example of character device files is printers, modems and consoles.

FIFO file

- FIFO file is special pipe device file which provides temporary buffers for two or more processes to communicate by writing data to and reading data from the buffer.
- A FIFO is created by the *mknod* system call and may be opened and accessed by any process that knows the name and has the permissions.
- The buffer associated with a FIFO file is allocated when the first process opens the FIFO file for read or write. The buffer is discarded when all processes which are connected to the FIFO close their reference to the FIFO file.
- FIFO file may be removed like any regular file. FIFO files can be removed in UNIX via the *rm* command.

7.1.3 File Attributes

- One of the characteristics of file is a set of file attributes that give the operating system more information about the file and how it is intended to be used. For human users convenience bane is given to the file. File attributes are varies from system to system. File attributes are as follows :
  1. Name
  2. Identifier
  3. Type
  4. Location/place
  5. Size
  6. Protection
  7. Date and time
- **File name** is in human readable. User can perform any operation on file using its name.
- When file is created by user, operating system assigns unique identification number to each file. OS uses this **identifier** for its operation.
- Type information is required for systems that support different types of files.
- **Location/place** information is pointer to a device and actual location of files on the device. The information needed to locate the file on disk is kept in memory.

- **Size** is measured in bits or bytes. It gives idea about current size of the file.
- **Protection** : This information is stored on the per-process table so the operating system can allow or deny subsequent requests. Attributes like protection, password, creator and owner provides protection to a file.
- **Date and time** : This information is related to creation and modification of files. Protection, security and monitoring purposes this data is used by operating system.

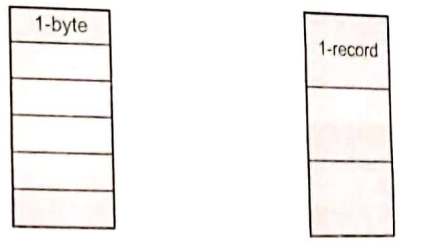
7.1.4 Operation on File

- File operations are simply those things that user can perform on a file. For example, user can create a file, save a file, open a file, read a file and modify a file. There are many different types of file operations supported by operating system.
- Operating system can provides system call for performing various operations on the file. Six basic file operations are creating, write, read, delete, repositioning and truncating.
- Following are the some list of file operation :
  1. Create 2. Write 3. Close
  4. Read 5. Delete 6. Repositioning
- All these operations require that the directory structure be first searched for the target file.
- **File creation :** Any time user can create a file. File is also created without data. The steps for creating a file.
  a. Space : File system must provide a space for the file.
  b. New file entry is made in the directory.
- **Write :** To perform write operation on the file, file name and data is required. It updates a file by adding new data and changes some content of the file. For writing into a file, operating system searches the directory for particular file. Write pointer is kept at the location where the writing starts. Write pointer position is at middle of the file or end of the file. Write pointer is updated when writing to file is completed.
- **Close :** Process is not used a file after closing. Process can not perform any operation on file. Operating system remove it entry from open file table.
- **Read :** A process reads all or some portion of the data in a file. Read system call is used for reading a file. For reading a file, name and position of the memory is specified. Read pointer is kept into the file for next read operation. Read pointer is updated after completion of the read system call. Information about read pointer is stored in *per process current file position pointer.*

- **Delete :** Operating system remove the file from file structure. For deleting a file, directory is searched with particular name and file is deleted. Deleting means making space free from memory and disk. Operating system also removes the directory entry. This free space is used by another file.
- **Repositioning :** The directory is searched for the proper entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seeks.
- Other file operations are open, append, seek, get and set attributes and rename.
- **Open :** before performing any operation on file, it is necessary to open file in required mode.
- **Append :** Append is related to write operation of file. Append only add data to the end of the file.
- **Get and set attributes :** Get attribute is used before performing operation on file. When user request a file for any operation, get attributes is executed by the process. Depending upon the attributes, user can allow or deny the operation. User can set the attributes and it changed after creation of file.
- **Rename :** User can change the name of existing file.

7.1.5 File Structure

- File structure is represented by field, record, file and database. It is also represented by byte sequence, record sequence and tree. Fig. 7.1.1 shows the three different kinds of files. Unix and windows both use unstructured byte sequence



(a) Byte sequence (b) Record sequence

Fig. 7.1.1

- Byte sequence is unstructured file. Operating system is nothing to do with content of file. All content is in byte format. Windows and UNIX operating

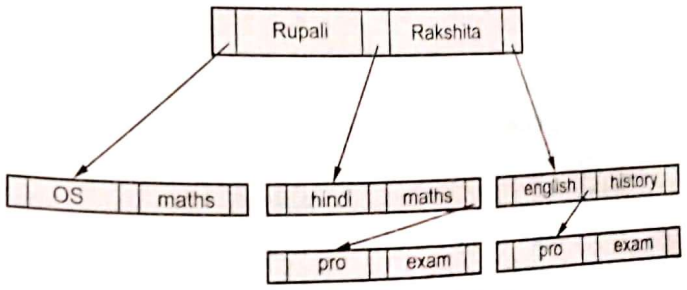


Fig. 7.1.1 (c) Tree structure

system uses this byte sequence method. It provides more flexibility to the user. User can put anything in the file and assign name whatever they want to the file.

- In record sequence model, file contains a sequence of fixed length record. Read and write operation on this type of file will perform one record at a time. It uses an idea of 80 column punched card. Most of the operating system based their file system on file consisting of 80 character records.
- Record sequence model also support 132 character records. Line printer uses 132 characters for printing.
- Last method is tree structure. File consists of tree of records. Each record contains key field in the fixed position. Key field is used to sort the tree record.
- Field, record, file and database is used in these file structure. Field is a basic element of data. Persons last name and date example of the field. It contains single value. Length and data type is characteristics of field.
- Collection of related field is records. A student record contains name, address, mobile number, university exam number, college name etc. Records may be fixed length or variable length. A record also contains sub-records. Sub-records may be one or more than one.
- File is collection of similar records. From user view, file is single entity and reference by name.
- Database is a collection of related data. All records have some relation in database. Engineering college database contains information about college, student, staff, university, syllabus, timetable, fee structure, various departments etc. One or more field is used in database.

Review Questions

1. *Explain the following in brief : File attributes and File operations.* **GTU : Winter-13, Marks 7**
2. *Explain in detail about file attributes and file operations.* **GTU : Summer-14, Marks 7**

7.2 File Access Methods

- Information and data is kept in files. Files are stored on the secondary storage device. When this information is to be used, it has to be accessed and brought into primary main memory. Information in files could be accessed in many ways. It is usually dependent on an application. File organization checks how efficiently the input-output storage medium used.

- File access method defines the way processes read and write files. Different access methods are available. Some operating systems provide only one access method and other systems provide many access methods.
  1. Sequential access
  2. Direct /Random access
  3. Indexed sequential access
- Access method means accesses to files that use a particular file organization are implemented by an input output control system.

7.2.1 Sequential Access Method

- Sequential access method is simple method. The information in a file is accessed sequentially one record after another.
- In this method, a process could read all the records in a file in order, starting at the beginning. It can not skip any records and can not read out of order. Fig. 7.2.1 shows sequential access method.
- Batch application uses sequential files. A byte stream file uses sequential access method.

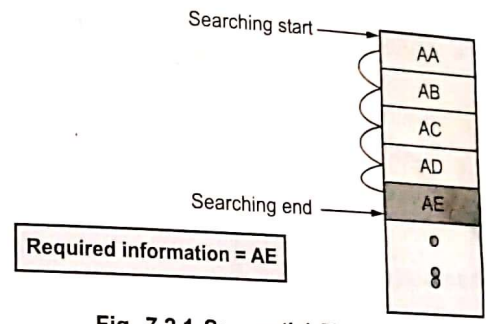


Fig. 7.2.1 Sequential file access

- **Example :** Compiler and editor usually access files in this method. Transaction file is also example of sequential access method.
- Sequential file organization only method easily stored on tape and hard disk. Sequential access is best suited where most of the records in a file are to be processed.

Disadvantages :

- It provides poor performances.
- More efficient search technique is required.

7.2.2 Direct Access Method

- It is also called random access method. This access allows a user to position the read/write mark before reading or writing. Fig. 7.2.2 shows direct access method.

- Direct access file method provides, accessing the records directly. Direct access method is based on the hard disk that is a direct access device. It allows random access of any file block.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing. This feature is used by editors. An editors need to randomly access the contents of the file.

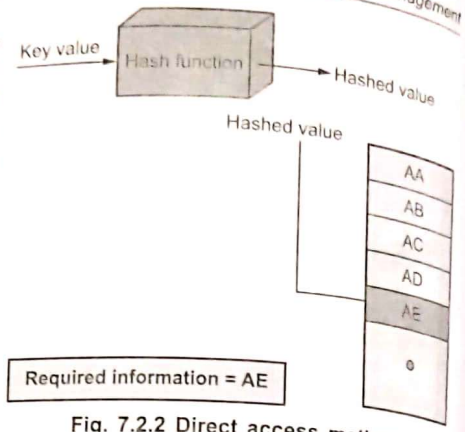


Fig. 7.2.2 Direct access method

- There is no restriction on the order of reading or writing for a direct access file. Operating system support is not required for direct access file.
- At the time of file creation, access method is defined. According to defined access method, file is accessed. Sequential access of a direct access file is possible but direct access of a sequential file is not.

Disadvantages :

1. Poor utilization of input-output device.
2. Consumes CPU time for record address calculation.

7.2.3 Indexed Sequential Access

- It is modified method of direct access method. This method is combination of direct and sequential access method.

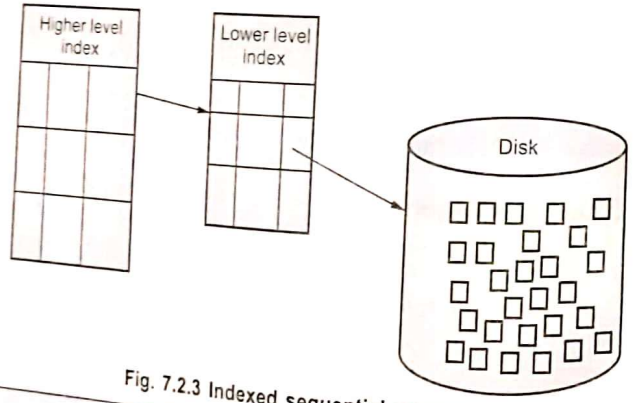


Fig. 7.2.3 Indexed sequential access

- In simple indexed sequential method, simple single level of indexing is used. Sequential file is used as index.
- Fig. 7.2.3 shows indexed sequential file. Indexed sequential access contains key field and a pointer to the main file.
- Indexed sequential files are store records in the order that they are written to the disk. Records may be retrieved in sequential order or in random order using an index to represent the record number in the file.
- If file size is large, larger index is required and it contains large number of entries. CPU takes time to search in to the index. Higher level index is used to reduce the search time. An entry in the higher level index points to a section of the index. This higher index contains the records. This index is searched to find the section of the disk that may contains a required record.
- File system maintains an overflow file. It adds new records to an overflow file. The record in the main file that immediately precedes the new record in logical sequence is updated to contain pointer to the new record in the overflow file.
- The overflow file is merged with the main file during a batch update. The multiple indexes for the same key field can be set up to increase efficiency.
- The lowest level of index file is treated as a sequential file and a higher level index file is created for that file. Higher index file would contain pointers to lower index files, which would point to the actual data items.

7.3 File Directory and Disk Structure

GTU : Winter-13

- User stores file in file system using directory. File system stores files of many users. Files are stored on the secondary storage device like hard disk, optical disk and memory disk. These device support random access method.
- Directory structure organizes and provides information about all files in the system. Every partition has a file system which consists of files and directory.
- Hard disks are split into one or more partitions. It is called as minidisks or volumes. User can assign names to volumes. Each disk contains minimum one partition. Operating systems support upto 24 partitions. Partitions can be larger than a disk i.e combing two hard disks. These partitions are called "virtual disks". Partition maintains information about files in "Device Directory".
- Files are stored in the directory. Directory is created by user. User is free to create as many as directory and assign suitable names. File system contains many directories. Directory maintains information about the file. File information includes type, size, date modified and name.

- In some operating system, directory itself is considered as a file. A set of logically associated files is called directory. Directory structure of the file system connects a directory to other directories in the system.
- Directory is a data structure that lists files and subdirectories in a collection. Directory structure contains a list of entries one for each file.
- Directory does not store any user data. Single dot (.) indicates a current directory and double dot (..) indicates parent directory. The root is identified by the directory '/'. The root file system has several subdirectories.
- Windows file system uses a letter followed by colon for specifying root directory. For example, root directory in windows is C: or D: or E:
- UNIX and Linux based file system uses slash (/) for specifying root directory.
- A directory can contain any number of files. A directory can also contain subdirectories. Because a directory is nothing more than a special kind of file, directories also have names.

Operation on directory :

1. **File searching :** Directory structure is searched for particular file entry. File uses symbolic names and similar names may indicate a relationship between files.
2. **Create a file :** User creates new files and added to the directory.
3. **Delete a file :** User remove the file after completing its work on files.
4. **Rename a file :** User change the file name if file content is change.
5. **Listing of directory :** Listing of files in the directory for some use. MS-DOS and windows uses "dir" command and Linux/UNIX uses "ls" command for this purposes.

7.3.1 Single Level Directory

- Single level directory is a simple structure and there are no sub-directories. Fig. 7.3.1 shows three files with single level directory. It is also called flat directory structure.
- Single level directory structure is suitable only for single user. If user increases, it creates the problem with assigning the name to files.

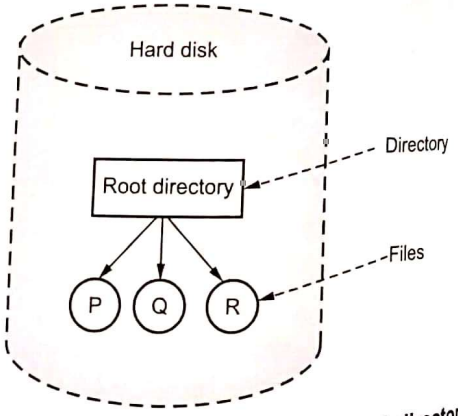


Fig. 7.3.1 Three files with single level directory

- In single level directory structure, no two files can have the same name.
- It is simple to implement and searching of file is faster. This type of directory system is used in cameras and phones.
- Because of limitation of file names, this structure is rarely implemented.
- File name size varies according to the operating system. MS-DOS operating system allows only 11 character file names and UNIX OS allows upto 255 characters.

Disadvantages of single level directory :

1. Not suitable for a large number of files
2. Limitation of unique file name.
3. For large number of file, recalling file name is difficult.

7.3.2 Hierarchical Directory System

- Here files are grouped together to form a sub directory. Each directory may contain several subdirectories. In hierarchical file system, a root directory points to other directories and files.
- Hierarchical directory system contains various forms of directory structures. They are as follows :
  1. Two-level directory structure
  2. Tree structured directories
  3. Acyclic graph directories

7.3.2.1 Two-level Directory Structure

- Two-level directory structure contains master file directory at the top level then user file directory at the second level. Actual user files are at the third level.
- File system maintains a master block for each user. Master block has one entry for each user. User directory address is stored in the master block. Each user has a private directory. Different user can assign same names to their files as the directories for each user are now separate. Fig. 7.3.2 shows two-level directory structure.

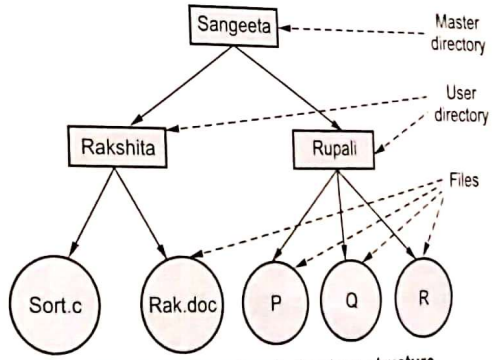


Fig. 7.3.2 Two-level directory structure

- Here user can work only in own directory. Other user need not worry about deleting or modifying files by other users.
- When user wants to create a file, system search that particular file name in the directory. If same file name is not found, then it creates a file otherwise it gives error messages.
- In this structure, different users may have files with same name. It solves the name conflict problem. When user want to open a file, filename is searched for in users directory.
- A two level directory can be a tree or an inverted tree of height 2.
- There are still problems with two-level directory structure. It effectively isolates one user from another. This is some time advantage or some time disadvantages. For completely independent user is an advantages and disadvantages when cooperation between two users is required.

7.3.2 Tree Structured Directory

- Fig. 7.3.3 shows the tree structured directory. In this structure, directory itself is a file. A directory and sub directory contains a set of files. Internal format is same for all directories.

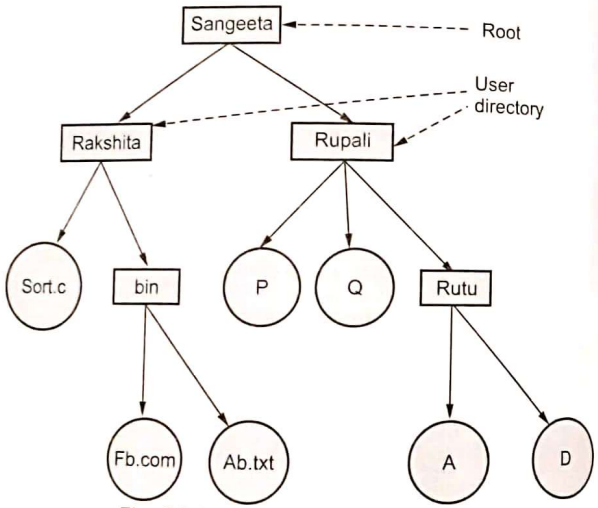


Fig. 7.3.3 Tree structured directory

- Commonly used directory structure is tree structure. Tree has a root directory. All files in disk have a unique path name. A path name describes the path that the operating system must take to get from some starting point in the file system to the destination object. Each process has its own working directory. Path names are of two types : **relative and absolute.**
- Relative path name : Its path starts from current working directory (may start with a "." or "..").
- Absolute path name : Its path starts from root directory (starts from "/").

- Tree structured directory differentiate file and subdirectory by using one bit representation. Zero (0) represents file and one (1) represents subdirectory.
- When process makes reference to a file for opening, file system search this file in the current directory. If needed file is not found in the current directory, then user either change the directory or give the full path of the file. System calls are used for performing any operation on the files or directory.
- Empty directory and its entry are easily deleted by operating system. If directory contains files and subdirectory, i.e. non-empty directory, some operating systems not delete the directory. To delete a directory, user must first delete all the files and subdirectory in that directory.

Merits

1. User can create his/her own directory.
2. User can access other user files by specifying path name.
3. Separate subdirectory for files associated with different topics
4. Searching id fast and efficient.

Demerits

1. Sub directory can not share between the user
2. Path is longer than two level directory structure

7.3.3 Acyclic Graph Directory

- Acyclic graph directory solve the problem of tree structure directory. It allows sharing the directory in between two users. At a time more than one places, shared directory or file will exist in the file system. Fig. 7.3.4 shows acyclic graph directory. Links can be used to construct acyclic graph structure. Acyclic graph is graph with no cycles.

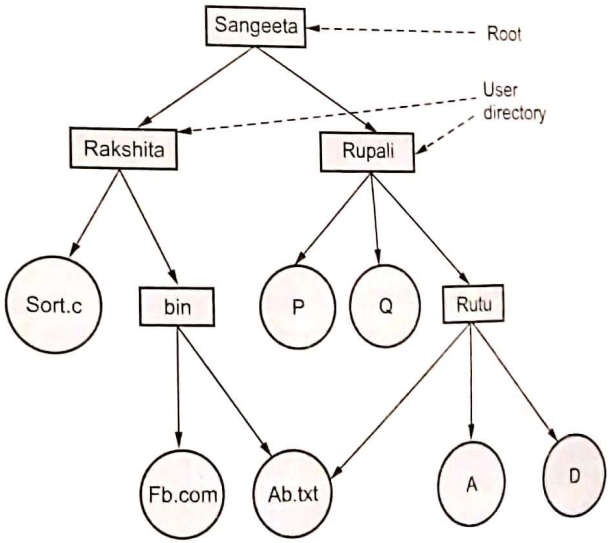


Fig. 7.3.4 Acyclic graph directory

- It is very interesting to note that a shared directory or file is not the same as two copies of the file. When there are two copies of files, each user can view the copy rather than the original, but if one user changes the file content, the changes will not appear in the other's copy.
- Only one original file exists for shared copy. Any changes made by one user are immediately visible to the other user. When user create file in shared directory, it automatically appear in all the shared subdirectories.

Implementation of shared files and directory :

1. Using link :

- Link is used for creating shared files and directory. A link is a pointer to another file or subdirectory. If reference to a file is made, the directory is first searched. A link is a logical relationship between an inode and a file that relates the name of a file to its physical location.
- UNIX operating system uses acyclic graph directory structure. Reference count is maintained by UNIX for files and directory. UNIX operating system provides two types of links : **hard link and symbolic link**
- A **hard link** contains multiple directory entries that both refer to the same file. Hard links are only used with ordinary files in the same file system.
- A symbolic link is also called soft link. It contains a special file which maintains the information about file where to find the linked file. Symbolic links may be used to link directories and files in other filesystems.
- Soft links were designed for two specific situations: Links to directories, which must be symbolic, and links to files in other file systems.
- Windows operating system only supports symbolic links.
- Hard links require a reference count, or link count for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.

2. Duplicating information :

- Duplicate all information about shared files in both sharing directories. Original file will remain as it is without modifying any content by shared user.
- Consistency is not supported by copied file and shared file. Shared files may have more than one absolute path.

Advantages :

1. Simple for file traverse.

2. Structure is more flexible than a simple tree structure.
3. It allows directories and files to be shared between other users.

Disadvantages :

1. Deletion of file is difficult.
2. Problem with duplicate directory entries is maintaining consistency if the file is modified.

Review Question

1. *What are the uses of device driver and controller in OS ? Explain.* **GTU : Winter-13, Marks 7**

7.4 File System Mounting

- Files contained in a file system can be accessed only when the file system is mounted. Each device consists of its own local file system. Mounting is the operation that administrator/user perform to cause the file system on a physical storage device to appear as part of the file system.
- Mount system call is used on UNIX/Linux OS for mounting file system and unmount system call for disconnecting the file system. Mounting file system is used when there are more than one file system is used. Required file system is mounted in the host file system machine as a root directory.
- Some operating system support the mounting the file system at any point.
- File system stores directories and files. Each file system contains exactly one directory at the very top level, called the root directory for that file system. This root directory can contain other directories and files. One file system is designated the root file system or /. Every other file system is mounted under the root file system.

Procedure for mounting file system :

- To mount the file system, device name and location within the file structure is required. It is provides by operating system. A mount point is typically an empty directory where the mounted file system will be attached.
- The operating system verifies the file system. It reads the file system by using device driver and directory for proper format and valid file system.
- Fig. 7.4.1 shows the mounting a file system on secondary storage device. File system manage mounted directories by using mount table. User can create soft links to files in mounted file system. Mount tables keep track of the actual physical location of the files.

- When a file system is mounted, the client can reference files by the local path name. There is no reference to remote host location, although files remain physically located at the remote site.

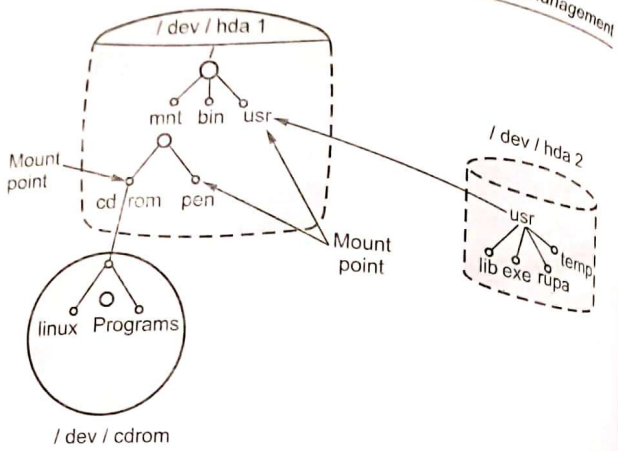


Fig. 7.4.1 Mounting file system

- Busy file system is can not unmount by using unmount command. A file system is considered busy in following situation :
  1. File system is shared in between the users.
  2. A program has a file open in that file system.
  3. A user is accessing a file/directory in the file system.
- **Example :** Consider a file structure of the pen drive. Following is Fig. 7.4.2 of file system before mounting the pen drive.

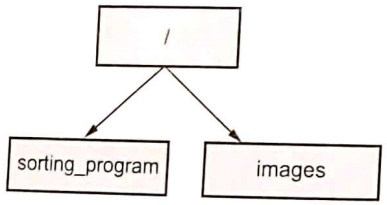


Fig. 7.4.2 File system before mounting

- The file system structure in the pen drive is as follows :

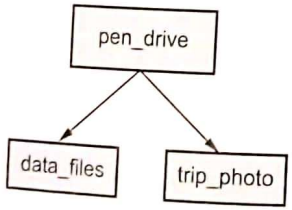


Fig. 7.4.3 File system structure in pen drive

- Now, the pen drive is mounted to the directory sorting_program. Here, the pen drive becomes the part of the file structure. To access a file in trip_photo, the path can be /sorting_program/ trip_photo. Fig. 7.4.4 shows file system after pen drive file system.

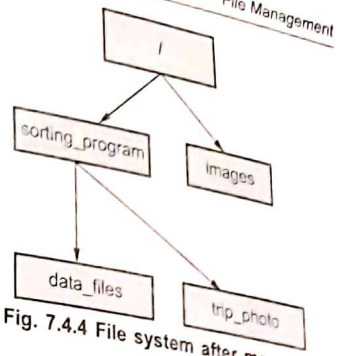


Fig. 7.4.4 File system after mounting

Advantages of mounting :

1. It is a user-oriented naming scheme
2. It makes sense to be able to mount at any level.

Disadvantages of mounting :

1. The hardware is less obvious since devices can be located anywhere in the tree structure

7.5 File Sharing

GTU : Winter-13

- Multiprogramming and multitasking operating system support the multiple user. These users perform various operations on the files. Files are shared between numbers of users. File protection, naming and sharing is issue for file sharing.
- Directory structure may be allows files to be shared by users. Sharing must be done through a **protection** scheme.
- In single user system, operating system maintains attributes for all files and directory. According to the attributes, operating system allows to operate the files for users. But on the multi-user system, more attributes are needed. An attributes like owner, group and other user are required. These are the access permission for files and directory.
- Access permissions are of three types : Read, Write, and eXecute.
  1. An "r" indicates read permissions for a file or directory.
  2. A "w" indicates write permissions.
  3. An "x" indicates execution permissions if it is a file and search permissions if it is a directory.
  4. Owner is the file owner who created a file. Owner having all rights (read, write and execute).
  5. The groups of other user have some limited access permission. Mostly read and execute permission.
  6. Last categories of user have only executed permission.

- **Example :** In UNIX operating system, file owner can perform all operation on a file. Group member can execute one subset of those operations and other than the group user can execute another subset of operations.
- When a user requests an operation on a file, the user ID can be compared with the owner attribute to determine if the requesting user is the owner of the file. Likewise, the group IDs can be compared.

Remote file systems

- Computer network is used to support remote file system. User can access remote file system. File Transfer Protocol (FTP) and Secure Shell protocol (SSH). Other supporting protocol like distributed file system and world wide web is also used for remote file system.
- SSH is a protocol for secure remote access to a machine over untrusted networks.
- A client-server model has three components :
  1. **Service :** A service is a software entity that runs on one or more machines. It provides an abstraction of a set of well-defined operations in response to applications' requests.
  2. **Server :** A server is an instance of a particular service running on a single machine.
  3. **Client :** A client is a software entity that exploits services provided by servers.
- Client application program running on the local machine requests a service from another application program, server running on the remote machine. Commonly server provides service to any client, not a particular client.
- Generally, a client application program that requests a service should run only when it is needed. A server program providing service should run all the time, as it does not know when its services will be needed.
- **Client-server** model allows clients to mount remote file systems from servers. The server is connected with multiple clients and it serve to all clients. Client and user-on-client identification is insecure or complicated.
- A client opens the communication channel using IP address of the remote host and the port address of the specific server program running on the host i.e. active open. Request-response may be repeated several times, the process is finite. The client closes the communication channel with an active close.
- A server program opens its door for incoming requests from clients but never initiates a service unless explicitly requested i.e. passive open. A server program is infinite and runs unless a problem occurs.

- Two or more clients can run at the same time on a machine is called concurrency in client. One client must start, run and terminate before another client may start. It is called iterative.
- Network File System (NFS) is standard for UNIX client-server file sharing protocol and Common Internet File System (CIFS) is standard Windows operating system protocol. The NFS client maintains a cache of file and directory attributes. The default settings will not ensure that files created or modified on one system will be visible on another system within a minute of file creation or modification. Network file system is common distributed file sharing system.
- File data modifications might not be visible on any NFS client system other than the one where the modifications are being made until an NFS commit is executed. Most NFS clients will issue a commit as part of closing a file. If multiple systems are reading files that might be modified, file system locking should be used.
- NFS communication protocols lets processes running in different environments share a file system. NFS implements a file system model that is almost identical to a UNIX system.
- **Distributed Information Systems** such as LDAP, DNS, NIS, and Active Directory implement unified access to information needed for remote computing. It is also known as distributed naming services.
- A name in a distributed system is a string of bits or characters used to refer to an entity. To resolve names a naming system is needed. Names are used to share resources, uniquely identify entities and refer to locations.
- The naming and locating facilities jointly form a naming system that provides the users with an abstraction of an object that hides the details of how and where an object is actually located in the network.
- It provides a further level of abstraction when dealing with object replicas.
- Email and FTP is used for sending information to remote users. But these two protocols will not support true file sharing. Domain Name System (DNS) is used to map the host name to network address translation for the internet. DNS replace the Email and FTP. The DNS is a hierarchical distributed naming system.
- On windows operating system, user name and password is used to create login on the server. Using this authentication method, server will allow and deny access to a requested file system. CIFS is used in conjunction with login name and password.
- Industry is moving towards the new technology. So new protocol called Lightweight Directory Access Protocol (LDAP) is used for secure distributed

values or items. Each value or item is stored in a field of a specific data type. Records may be of either fixed or variable lengths.

- Blocks are of fixed length. Disks are divided into concentric circular tracks on each disk surface. Track capacities vary typically from 4 to 50 kbytes. A track is divided into blocks. Whole blocks are transferred between disk and main memory for processing.
- Reading or writing a disk block is time consuming because of the seek time s and rotational delay. Double buffering can be used to speed up the transfer of contiguous disk blocks.
- Blocking is refers to storing a number of records in one block on the disk. The blocking factor refers to the number of records per block. There may be empty space in a block if an integral number of records do not fit in one block.
- Spanned records are referring to records that exceed the size of one or more blocks and hence span a number of blocks.
- The operating system may choose to use a larger block size than the sector size of the physical disk. Each block consists of consecutive sectors. A larger block size increases the transfer efficiency. It can be convenient to have block size match the machine's page size.
- Some systems allow transferring of many sectors between interrupts.
- Hard disk and tape drive consists of one or more records. These devices recognize blocks of data. Interblock gap is a blank space and it separates one block from another. The length of an IBG on tape is 0.3 to 0.6 inches depending on the device, and the length of an IBG on disk varies by device and by track location.
- Functions of IBG :
  1. To define the start and end of each block of data
  2. To provide space for the tape when the drive stops and restarts for each read or write of a block.
- Record blocking are of three types :
  1. Fixed blocking.
  2. Variable length spanned blocking.
  3. Variable length unspanned blocking.

Fixed blocking

- Fixed blocking uses fixed length records. Fig. 7.6.1 shows fixed blocking.
- After allocating data, it creates unused space at the end of each block. It is called internal fragmentation.

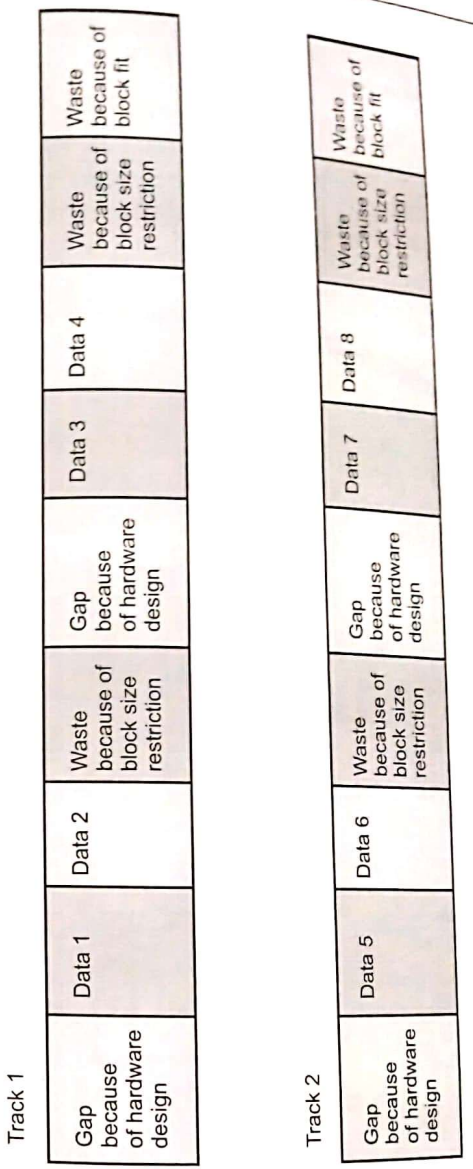


Fig. 7.6.1 Fixed blocking

- If block capacity is 100 records and the number of records in block is 80 then the wasted space is 100 – 80 = 20. This is called internal fragmentation.
- It is used in sequential files with fixed record records.

Variable length spanned blocking

- Fig. 7.6.2 shows variable length spanned blocking.

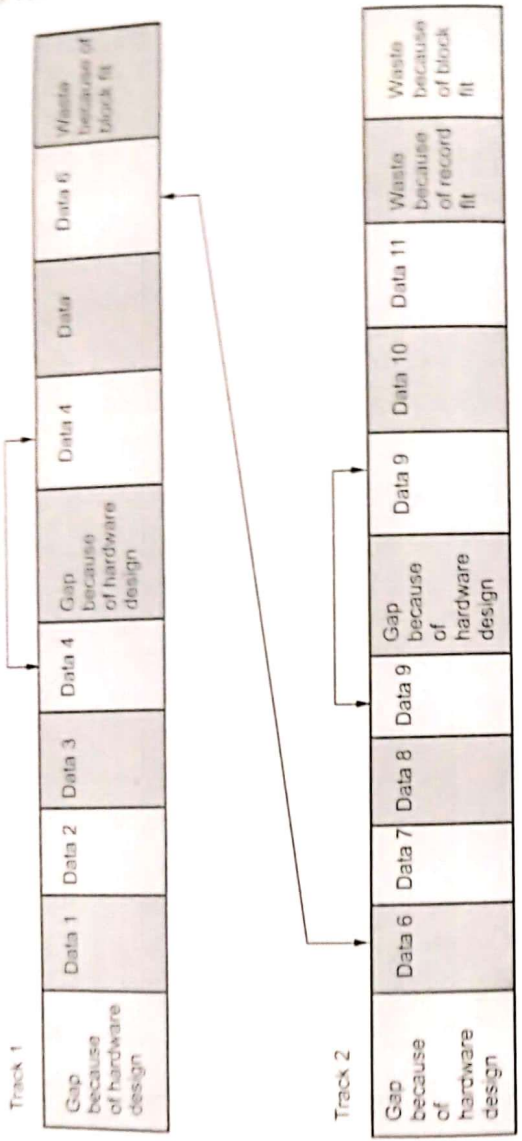


Fig. 7.6.2 Variable length spanned blocking

- In this blocking method, record sizes are not same. It uses variable length records. These records are packed into blocks without wastage of space.

- Some record may consume two blocks. In this situation, pointer is passed from one block to another block.
- This method is suitable for storage. There is no limit on the size of records.
- Variable length spanned blocking method is difficult to implement.

Variable length unspanned blocking

- Fig. 7.6.3 shows variable length unspanned blocking. It also uses variable length records. But records are not spanned between the blocks. It creates larger waste space as compared to other method.

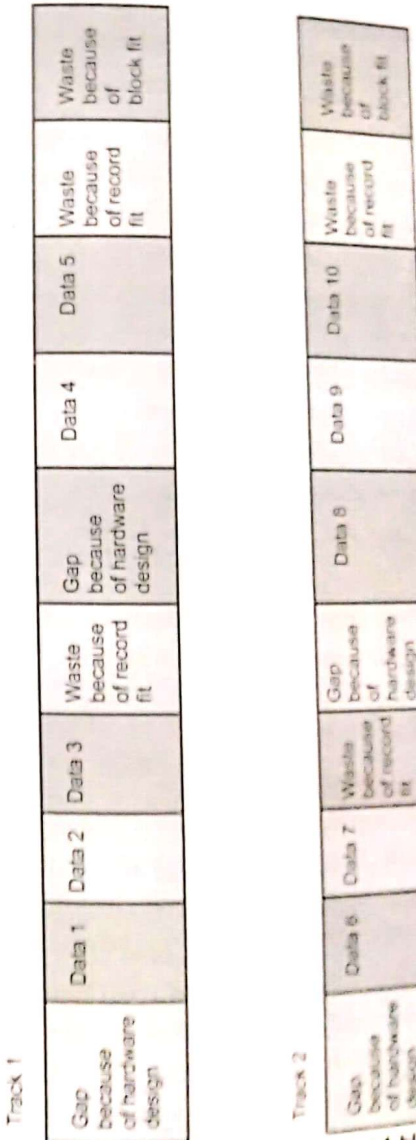


Fig. 7.6.3 Variable length unspanned blocking

- If next record is larger that the spanned size, then it creates unused space.

7.7 File System Structure

- File system is mounted and maintain by the secondary storage which provides by disk. Characteristics of disk are as follows :
  1. Same place is used for reading, writing and modification.
  2. User can access disk directly for any block of information.
- Block is used for data transfer. Each block contains one or more sectors. The I/O transfer between memory and disk are performed on block. Default sector size is 512 bytes.
- Concept of file system is used for the efficient and convenient access to the disk. File system allows data to be stored, located and retrieved easily. The file system is generally composed of many different levels. Fig. 7.7.1 shows layers of file system. There are six layers. Each layers give support to neighbour layers. It also provides function to above layer and below layer. Each layer uses the features of neighbour layer while creating new features for use by the higher levels.

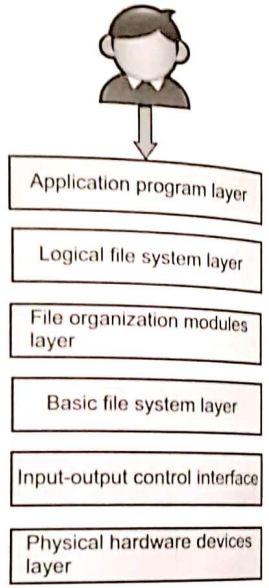


Fig. 7.7.1 Layers of file system

- File system creates two main design problems :
  1. Creation of algorithm and data structure for mapping of logical file system to physical secondary storage devices.
  2. File system view for user.
- **Input-output control interface :** It consists of device driver and interrupt handler. Both are used for data transfer between main memory and disk system. Device driver used as a translator. Device driver input is high level commands and output is low level hardware specific instructions.
- **Basic file system layer :** It generates the command for device drivers. Particular device driver read data and writes physical blocks on the disk. Basic file system also manages the memory buffer and caches that hold various file system. Cache memory is used to hold meta-data information of file system.

- **File organization modules layer :** This layer maintain information about files and their logical blocks and physical blocks. It translate the logical block address to physical block address by considering physical location of files and file allocation method. File logical numbers are do not match with physical block containing data, so translation is required. Free space manager is included in file organization modules.
- **Logical file system layer :** It manages metadata information. Actual data is not part of metadata information. But file system structure is part of metadata. Directory structure is used to organized file module. Operating system maintains file control block for each file. UNIX uses inode instead of file control block. File control block stored information about file, location, ownership and access permissions.
- **Application program layer :** User/programmer creates an application programs.
- **Physical hardware devices layer :** It contains actual hardware device link disk, memory etc.
- Following is the list of different operating systems file system :

Operating system	File System formats
Windows XP, NT and 2000	FAT, FAT32, NTFS
UNIX	UNIX File System (UFS)
Standard LINUX	Extended file system

- Google uses its own Google file system.

7.8 File System Implementation

- File system is implemented on disk and memory. How to implement the file system, it varies according to operating system and file system. But all the operating system follow some general rules. If the file system is implemented on the disk, it contains the following information :

1. **Boot block control :** Boot block is maintained per volume. The boot block contains the initial bootstrap program used to load the operating system. Operating system requires some information at the time of booting. If the disk is divided into number of partitions, the operating system is stored in the first partition of the disk. If the operating system is not installed on the disk, then this block can be empty. In UNIX file system, this is called the book block and in NTFS, it is the partition boot sector.

2. **Volume control block :** It consists of volume or partitions detail information. The information like block size, number of blocks in the partition, free block count, free block pointer, free FCB count and FCB pointers. In UNIX operating system, each partition is a standalone file system. Superblock is name in UNIX file system. A super block describes the state of the file system : The total size of the partition, the block size, pointers to a list of free blocks, the inode number of the root directory, magic number, etc. In network file system, it is stored in the master file table.
3. **Directory structure :** It is used to organize the files. Directory structure is maintained per file system.
4. **Per-file PCB :** It contains information about files such as file size, file ownership, file permission and location of data blocks. In NTFS, master file table stored this information. Master table file uses a relational database structure.

- In-memory information is used for caching and files system management. Caching improve the performance.

1. **In-memory mount table :** It contains information about each mounted volume.
2. **In-memory directory structure cache :** It contains recently accessed directories information.
3. **System wide open table :** Open files FCB information is stored.
4. **Per-process open file table :** It maintains the pointer to the appropriate entry in the system wide open file tables.

- UNIX operating system treats a directory exactly as a file. Windows NT implements separate system calls for files and directories. It treats directory as entities separate from files. Fig. 7.8.1 shows a file control block.

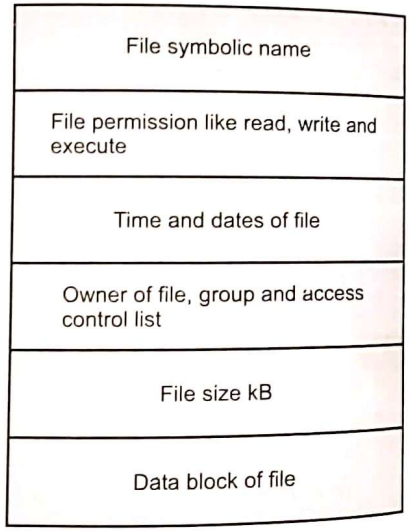


Fig. 7.8.1 File control block

- FCB specify the information that the system needs to manage a file. Sometimes it is called file attributes. These are highly system dependent structures. For creating new file, an application program calls the logical file system.

7.9 Log Structured File Systems

- Rosenblum and Ousterhout introduced new type of file system i.e. log structured file systems. To improve write performance by buffering a sequence of file system changes to disk sequentially in a single disk write operation. Logs written include all file system information, including file data, file inode, directory data, directory inode.
- When data is writing to disk, LFS stores all updates in in-memory segment. Data is written to disk when segment is full. Unused part of disk is used for writing the disk. LFS never overwrites existing data but it search free space and write to that position.
- New data is written sequentially in the log files free space. Writing to disk sequentially is not enough. Because in between first write and second write, disk may rotate from its position. To solve this problem, LFS uses write buffering.
- Before writing to the disk, LFS keeps track of updates in memory; when it has received a sufficient number of updates, it writes them to disk all at once, thus ensuring efficient use of the disk. When writing to disk, LFS buffers updates in an in-memory segment and then writes the segment all at once to the disk. As long as the segment is large enough, these writes will be efficient.

7.10 Virtual File Systems

- Different operating system uses different types of file systems. Different types file system are NTFS, FAT16, FAT32, ReiserFS etc. Windows operating system uses NTFS but it also supports FAT16 and FAT32.
- Implementing multiple file system in single operating system is by write directory and file routines. UNIX operating system uses virtual file system for integrating multiple file system into an orderly structure. Position of virtual file system is in between user process and file system. Fig. 7.10.1 shows virtual file system.

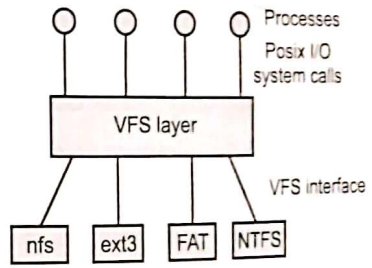


Fig. 7.10.1 Virtual file system

- UNIX operating system uses object oriented techniques to simplify, organize implementation method. This simplified and modular method allows very dissimilar file system type to be implemented within the same structure.
- Data structure and procedures are used to isolate the basic system call functionality from the implementation details. All system calls relating to files are directed to the VFS for initial processing. User process issue these system calls. System call includes read, write, open and lseek. In UNIX operating system, upper interface is called POSIX interface.

- Middle layer is VFS interface. VFS interface consists of multiple system calls. It issue these calls to different file system to get work done. Normally file system under VFS is implemented on a local disk but this is not always true.
- Virtual file system identifies the local file system and remote file system. LINUX operating system defined four objects for virtual file system.
  1. Inode object : For individual files
  2. File object : For open file
  3. Superblock object : For entire file system
  4. Dentry object : For individual directory entry
- Set of operation is performed by these objects. Every object of one of these types contains a pointer to a function table.

7.11 Directory Implementation

- Directory is implemented by using linear list and hash table.

Linear list

- Linear list is simple method of directory implementation. It uses file names with pointers to data block. It is time consuming because of searching overheads.
- To create a new file, file name is searched in the directory to avoid the file name duplication. Then it adds new entry at the end of the directory. To delete a file, again file name is searched in the directory. Space is released after deleting a file. Directory entry is also removed. Unused space is marked as free entry. Last entry of directory is copied into the free space list.
- For deleting a file, some OS uses linked list. Linked list decreases time required.
- Searching the directory entry is the major disadvantage. It is slow process. Because of this reason, many operating system uses software cache memory. It stores the most recently used directory information.

Hash table

- Hash table is one more data structure used for directory implementation. Hash table takes a value computed from the file name and returns a pointer to the file name in the liner list.
- Hash table reduces the searching time. It uses fixed size block.

7.12 Allocation Methods

GTU : Winter-13, Summer-14

- The primary issue of file implementing file storage is how to keep track of file blocks. File consists of a collection of blocks.

File allocation

- Following issue are considered for file allocation :
  1. After creation of new file, whether the required maximum space for the file is allocated at once or not ?
  2. Portion : Space is allocated to a file as one or more contiguous units.
  3. Data structure or table used to keep track of the portion which is assigned to a file.
- Three major methods of allocating disk space are in wide use
  1. Contiguous
  2. Linked
  3. Indexed

7.12.1 Contiguous Allocation

- When user creates a file, system allocates a set of contiguous blocks on disk. This is a pre-allocation method that uses portion of variable size. Contiguous allocation is simple method to implement. It only search free list of correct number of consecutive blocks and mark them as used block.

File allocation table

File name	Start block number	Block length
File 1	1	3
File 2	18	2
File 4	6	6
File 5	20	1
File 6	25	3

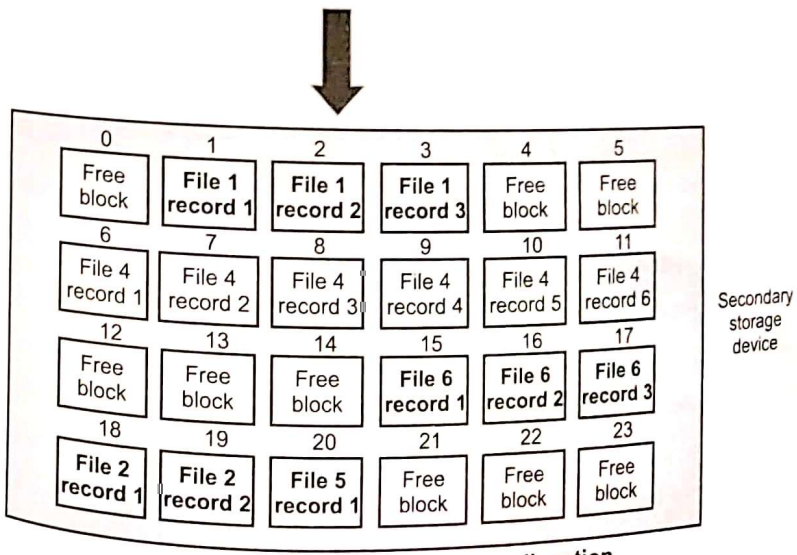


Fig. 7.12.1 Contiguous allocation

- Disk address is a linear ordering on the disk. Because of linear ordering, accessing block b + 1 after block b normally requires no head movement. Here head movement is only one track. Fig. 7.12.1 shows contiguous allocation.
- Contiguous allocation of a file is defined by the disk address and the length of the first block.
- If the file size is "n" blocks and starting location is "L", then it occupies blocks L, L+1, L+2, L+3, ..., L+(n–1). The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- Sequential and random access can be supported by contiguous allocation.
- It is easy to retrieve single block. To improve the I/O performance of sequential processing, multiple blocks can be brought in one at a time. It supports sequential access very well because files entire data is stored in adjacent block. This method also supports random access.
- Contiguous allocation also suffers from **external fragmentation.** Small free disk spaces are created after allocation free block and deleting files. External fragmentation means there will require free space available but that is not contiguous. To solve the problem of external fragmentation, compaction method is used.
- One more problem is that how to calculate the space needed for a file. It is difficult to estimate the required space.
- This method is good for storing data on CD-ROM.

Characteristic of contiguous file allocation :

1. It supports variable size portions.
2. Pre-allocation is required.
3. It requires only single entry for a file.
4. Allocation frequency is only once.

Advantages :

1. It supports variable size portion.
2. Easy to retrieve single block.
3. Accessing a file is easy.
4. It provides good performance.

Disadvantages :

1. It suffers from external fragmentation.
2. Pre-allocation is required.

7.12.2 Linked Allocation

- Linked allocation is also called chained allocation. Operating system keeps an ordered list of free blocks. File descriptor stores pointers to the first block and each block stores pointer to the nest block.
- Fig. 7.12.2 shows linked allocation. The disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. No space is lost to disk fragmentation.
- Creation of new file is easy. For new file, simply create new entry in the directory. Reading a file is straightforward. User simple read blocks by following pointers from block to block. There is no external fragmentation with linked allocation.
- To write to file, system finds a free block, and this new block is written to and linked to the end of the file.

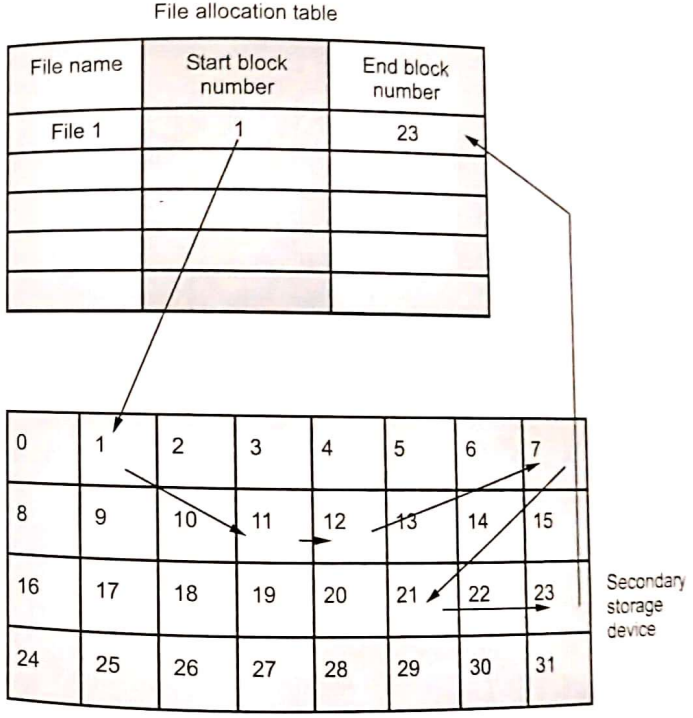
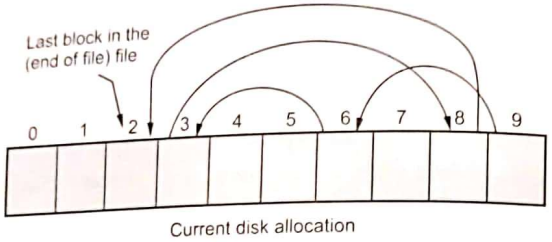


Fig. 7.12.2 Linked allocation

- While creating a new file, it is not necessary to declare the size of the file. A file can contiguous to grow as long as free blocks are available.
- Compaction can be used so that blocks of one file are located continuously on the disk. It optimizes disk access.

- File allocation table is an extension of the linked allocation method. Instead of putting the pointers in the file, keep a table of the pointers around. This pointer table can be quickly searched to find ant random block in the file.
- Fig. 7.12.3 shows the file allocation table. All blocks on the disk must be included in the table. This method is used by windows operating system.



Current disk allocation

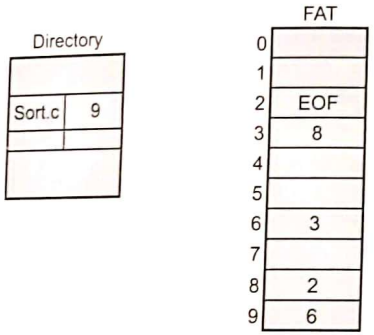


Fig. 7.12.3 FAT

Characteristics

1. It supports fixed size portions.
2. Pre-allocation is possible.
3. File allocation table is one entry for a file.
4. Allocation frequency is low to high.

Advantages :

1. There is no external fragmentation.
2. It is never necessary to compact disk space.
3. Pre-allocation is not required.

Disadvantages :

1. Files are accessed only sequentially.
2. Space required for pointers.
3. Reliability is not good.
4. Can not support direct access.

7.12.3 Indexed Allocation

- Indexed allocation method solves the problem of both contiguous and linked allocation. It uses concept of index block. Index block stores the entire pointer in one location. But the index block will occupy some space and thus could be considered as an overhead of the method.
- OS keeps a list of free blocks. It allocates an array to hold pointers to all the blocks used by the file. It allocates blocks only on demand. Fig. 7.12.4 shows indexed allocation.

File allocation table

File name	Index block number
File 1	10
File 4	24

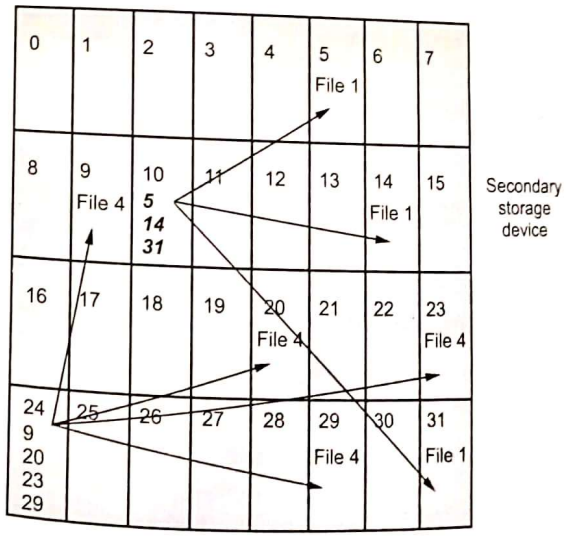


Fig. 7.12.4 Indexed allocation

- In indexed allocation, each file maintains its own index block. It contains an array of disk sector of addresses. For example : The nth entry in the index block points to the nth sector of the file. The directory contains the address of the index block of a file. To read the nth sector of the file, the pointer in the nth index block entry is read to find the desired sector.
- It supports direct access and without suffering from external fragmentation. Any free block anywhere on the disk may satisfy a request for more space.
- Indexed allocation does suffer from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Advantages :

1. It supports sequential and direct access.
2. No external fragmentation.
3. Faster than other two methods.
4. It supports fixed and variable size blocks.

Disadvantages :

1. Indexed allocation does suffer wasted space.
2. Pointer overhead is generally greater.

Review Questions

1. *Explain the following in brief : Contiguous and linked list allocation for implementing file system.* **GTU : Winter-13, Marks 7**
2. *Describe various file organization techniques.* **GTU : Summer-14, Marks 7**

7.13 Free Space Management

- Space that is allocated to files must be managed and space which is not allocated to any file must be managed. To keep track of free disk space, the system maintains a free space list. File allocation table and disk allocation both are required to manage free space properly.
- When a file is deleted, its disk space is added to the free-space list.
- Following are the techniques used for free space management :
  1. Bit tables or bit vector
  2. Chained free portions or linked list
  3. Indexing or grouping
  4. Free block list or counting

1. Bit tables or bit vector

- Each block on the disk is represented by bit. It uses vector chain for blocks. Free block is represented by o and used block represented by 1.
- For example, consider a disk where blocks 3, 4, 5, 7, 9, 14, 15, 19, 30, 31, 32, 35, 36, 37, 38 are free blocks and rest of the blocks are allocated. The free space is shown below :

  11100010101111001101111111111000110000
- The memory required for a block bitmap = Disk size / 8 × (block size of file system)
- Bit map requires extra space. For example :

  Block size = 2^{12} bytes

  Disk size = 2^{30} bytes

$$\text{Block number (n)} = \frac{\text{Block size}}{\text{Disk size}} = \frac{230 \text{ bytes}}{212 \text{ bytes}} = 2^{18} \text{ bytes}$$

- When bit table is stored into the memory, then exhaustive search of the table can slow file system performance. Most of the file system use auxiliary data structure for bit table. File system also maintain summary table. Summary table contains sub-range, number of free blocks and maximum sized contiguous number of free blocks.
- Summary table is used to store information about contiguous free blocks. Whenever file system needs a number of contiguous blocks, it can scan the summary table to find an appropriate sub-range and then search that sub-range.
- This method is used in Apple Macintosh.

Advantage :

a. Easy to find a free blocks

b. It is as small as possible.

Disadvantage :

It may not be feasible to keep the bitmap in memory for large disks.

2. Link list

- In this method, all free space disk blocks are linked, keeping a pointer to the first free block. All file allocation methods used link list free space techniques.
- There is small space overhead because there is no need for a disk allocation table. In the above example, free blocks are 3, 4, 5, 7, 9, 14, 15, 19, 30, 31, 32, 35, 36, 37, 38. Here free block pointer is on block number 3. Block 3 contains pointer to block

4, block 4 contains pointer to block 5 and block 5 contains pointer to block 7 and so on.

- This method is not efficient because to reach free block 9, we have to traverse block 3, 4, 5 and 7 then we will reach to block 9.
- Disk will become quite fragmented after some use. Many portions will be a single block long.

2. Grouping

- First free block contains the addresses of n free blocks. The first n-1 of these blocks is actually free. The last block also contains the address of another n free block.
- Consider the free blocks : 3, 4, 5, 7, 9, 14, 15, 19, 30, 31, 32, 35, 36, 37, 38 Then

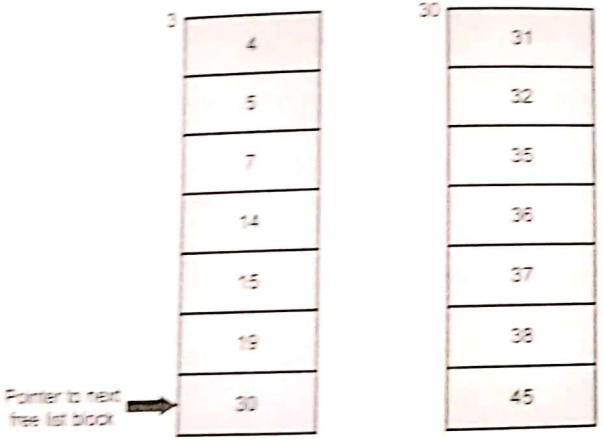


Fig. 7.13.1

- When all the blocks in the group have been allocated, then use the block that held the pointer.
- Because of grouping method, address of a large number of free blocks can be found quickly.

4. Counting

- It keeps the address of the first free block and the number n of free contiguous blocks that follow the first block. Each entry in the free space list then consists of a disk address and a count.

7.14 Efficiency and Performance

- To read a file may require many head seek movements and consequently take a much longer time than if its blocks were written one after another on the disk.
- Directories are searched as lists so that the average time to find a directory entry initially increases in direct proportion to the total number of entries. The blocks that a directory uses to store its entries are referenced from its inode. Searching for a directory entry therefore becomes slower when indirect blocks have to be accessed.
- For large numbers of files the only way to efficiently scale to support large numbers of files is to dynamically allocate index space for files.
- Many traditional file systems require a checking program to check file system consistency after a crash. On large, active file systems this type of checking can take a prohibitively long time to complete. Solving this problem must not degrade I/O performance.

7.14.1 Efficiency

- Disk allocation and directory algorithm are the main factor for efficient use of secondary storage disk. System efficiency and performance is depending upon the disk accessing method.
- UNIX operating system pre-allocates inode so it occupies space even before any files are created. It also distributes inode across the disk and tries to store data files near their inode, to reduce the distance of disk seeks between the inode and the data.
- Depending on the file size, some operating systems use variable size clusters The large data that is stored in a directory so the more often the directory blocks have to be re-written. In year 1990, Sun implemented a new algorithm called clustering that allows for more extent-like behavior by gathering up to 56 kB of data in memory to allocate disk space contiguously. Sun also extended the maximum file size to 1 TB in Solaris 2.6.
- Kernel table sizes used to be fixed and could only be changed by rebuilding the kernels. Modern tables are dynamically allocated, but that requires more complicated algorithms for accessing them.

7.14.2 Performance

- Performance should not degrade as the size of the file system, an individual file or the total number of files stored grows.

- File system performance is often a major component of overall system performance and is heavily dependent on the nature of the application generating the load. To achieve optimal performance, the underlying file system configuration must be balanced to match the application characteristics. Fig. 7.14.1 shows file system caching via the page cache.

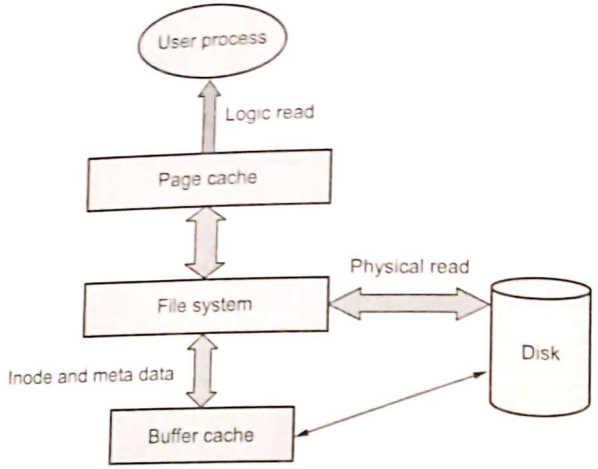


Fig. 7.14.1 File system caching via the page cache

- Traditional UNIX implements file system caching in the I/O subsystem by keeping copies of recently read or written blocks in a block cache. This block cache sits just above the disks and caches data corresponding to physical disk sectors.
- The file system is still involved in the page cache lookup, but the amount of work the file system needs to do is dramatically simplified. The page cache is implemented in the virtual memory system. In fact, the virtual memory system is architected around the page cache principle, and each page of the physical memory is identified the same way, by file and offset. Pages associated with files point to regular files, while pages of memory associated with the private memory space of processes point to the swap device.

❑❑❑

8 Security and Protection

Syllabus

Security environment, Design principles of security, User authentication, Protection mechanism, Protection domain, Access control list.

Contents

8.1 Security	Winter-13	
8.2 User Authentication	Summer-13,	Marks 2
8.3 Protection Mechanisms	Summer-13, Winter-13,	Marks 4

8.1 Security

GTU : Winter 13

- Computer security addresses the issue of preventing unauthorized access to resources and information maintained by computers. Computer system must provide mechanism to manage security threats.
- Basic methods used to protect software and hardware is passwords, backups, maintenance of written security policies and user training.
- **Computer security** deals with the prevention and detection of unauthorized actions by users of a computer system.
- How to protect the valuable assets? It is necessary to keep in safe place like a bank to protect the valuable assets. But bank is not a safe place now a day. There are so many example where bank robbery in our country.
- Bank robbery is the crime of stealing from a bank during opening hours. Protecting assets was difficult and not always effective.
- Now a day, protection is easier because many factors working against the potential criminal. Very sophisticated alarm and camera systems silently protect secure places like banks.
- Traditionally information security provided by physical i.e. rugged filing cabinets with locks and administrative mechanisms i.e. personnel screening procedures during hiring process.
- Asset protection systems are designed to recover stolen cash and high value assets, apprehend criminals and deter crime. The system has the capacity to track, protect and manage critical assets in real-time.
- The techniques of criminal investigation have become so effective that a person can be identified by genetic material, voice, retinal pattern, fingerprints etc.
- Use of networks and communications links requires measures to protect data during transmission.
- **Data security** is the science and study of methods of protecting data from unauthorized disclosure and modification.
- Data and information security is about enabling collaboration while managing risk with an approach that balances availability versus the confidentiality of data.
- **Computer security** : Generic name for the collection of tools designed to protect data and to hackers.
- **Network security** : Measures to protect data during their transmission.
- **Internet security** : Measures to protect data during their transmission over a collection of interconnected networks.

Protecting valuables

- Following are certain aspects for the need of security :
  1. Increasing threat of attacks.
  2. Fast growth of computer networking for information sharing.
  3. Availability of number of tools and resources on internet.
  4. Lack of specialized resources that may be allotted for securing system.

8.2 Security Goals

- Security goals are as follows :
  1. Confidentially
  2. Integrity
  3. Availability.

1. Confidentiality

- Confidentiality refers to limiting information access and disclosure to authorized users and preventing access by or disclosure to unauthorized ones.
- Sensitive information should be kept secret from individuals who are not authorized to see the information.
- Underpinning the goal of confidentiality are authentication methods like user-IDs and passwords that uniquely identify a data system's users, and supporting control methods that limit each identified user's access to the data system's resources.
- Confidentiality is not only applied to storage of data but also applies to the transmission of information.
- Confidentiality means that people cannot read sensitive information, either while it is on a computer or while it is traveling across a network.

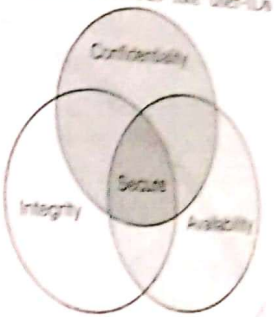


Fig. 8.1.1 Relationship between confidentiality integrity and availability

- Fig. 8.1.1 Relationship between Confidentiality Integrity and Availability

2. Integrity

- Integrity refers to the trustworthiness of information resources.
- Integrity should not be altered without detection.
- It includes the concept of "data integrity" namely, that data have not been changed inappropriately, whether by accident or deliberately malign activity.

- It also includes "origin" or "source integrity" that is, that the data actually came from the person or entity you think it did, rather than an imposter.
- Integrity ensures that information is not changed or altered in transit. Under certain attack models, an adversary may not have to power to impersonate an authenticated party or understand a confidential communication, but may have the ability to change the information being transmitted.
- On a more restrictive view, however, integrity of an information system includes only preservation without corruption of whatever was transmitted or entered into the system, right or wrong.

3. Availability

- Availability refers, to the availability of information resources. An information system that is not available when you need it is at least as bad as none at all.
- Availability means that people who are authorized to use information are not prevented from doing so. It may be much worse, depending on how reliant the organization has become on a functioning computer and communications infrastructure.
- Almost all modern organizations are highly dependent on functioning information systems. Many literally could not operate without them.
- Availability, like other aspects of security, may be affected by purely technical issues (e.g. a malfunctioning part of a computer or communications device), natural phenomena (e.g. wind or water), or human causes (accidental or deliberate).
- For example, an object or service is thought to be available if
  i) It is present in a usable form.
  ii) It has capacity enough to meet the services needs.
  iii) The service is completed an acceptable period of time.
- By combining these goals, we can construct the availability. The data item, service or system is available if
  i) There is a timely response to our request.
  ii) The service and system can be used easily.
  iii) Concurrency is controlled.
  iv) It follows the fault tolerance.
  v) Resources are allocated fairly.

4.2 Policy and Mechanism

- *Policy* is a particular approach that dictates the way a mechanism is used to achieve specific goals.
- **Mechanism** is the method by which the system implements the security policy.
- In most of the system, the policy changes over the time as the system's set of resources and user changes but mechanisms that implements access control do not require modification.

4.3 Security Terminology

1. Encryption : It is a process of encoding a plain text message into non-readable format.
2. Decryption : Process of transferring an encrypted message back into its normal form.
3. Plaintext : The original form of message.
4. Ciphertext : The encrypted form of plain text.
5. Cryptosystem : A system for encryption and decryption is called a cryptosystem.
6. Vulnerability : It refers to the security flaws in a system that allows an attack to be successful.
7. Passive attack attempts to learn or make use of information from the system but does not affect system resources.
8. Active attacks involve some modification of the data stream or the creation of a false stream.

4.4 Intruders

- An intruder is a person who attempts to gain unauthorized access to a system, to damage that system, or to disturb data on that system.
- Three classes of intruders are Masquerader, Misfeasor, and Clandestine user.
  1. **Masquerader :** An unauthorized user who penetrates a computer system's access control and gains access to user accounts.
  2. **Misfeasor :** A legitimate user who accesses resources he is not authorized to access. Who is authorized such access but misuses his privileges.
  3. **Clandestine user :** A user who seizes the supervisory control of the system and uses it to evade auditing and access control.

Intrusion techniques

- **Objective :** An intruder wants to gain access to a system.
- Access is generally protected by passwords. System maintains a file that associates a password with each authorized user.
- Password file can be protected with : **One-way encryption and access control**
  1. **One way function :** A system stores passwords only in encrypted form. When user presents a password, the system transforms that password and compares it with the stored value.
  2. **Access control :** Access to the password file is limited to very few people.

Techniques for guessing passwords

1. Try default passwords. (Used with standard accounts that is shipped with systems.)
2. Try all short words, 1 to 3 characters long.
3. Try all the words in an electronic dictionary.
4. Collect information about the user's hobbies, family names, birthday, etc.
5. Try user's phone number, social security number, street address, etc.
6. Try all license plate numbers (AP 12 AA 4453).
7. Use a trojan horse.
8. Tap the line between a remote user and the host system.

8.1.5 Security Design Principles

1. Keep security mechanisms as simple as possible.
2. Security should not depend on secrecy of design or implementation.
3. Separation of Privilege : To grant privilege, multiple conditions are applied.
4. System or company mechanisms should not be shared.
5. Security mechanisms should not add to difficulty of accessing resource.
6. A subject should be given only those privileges necessary to complete its task. If a subject does not need an access right, the subject should not have that right.

University Question

1. *Write a note on generic security attacks.* **GTU : Winter-13, Marks 4**

8.2 User Authentication

GTU : Summer-12

Authentication

- Authentication techniques are used to verify identity. The authentication of authorized users prevents unauthorized users from gaining access to corporate information systems.
- Authentication method is of validating the identity of user, service or application. The use of authentication mechanisms can also prevent authorized users from accessing information that they are not authorized to view.
- Data authentication means providing data integrity as well as that the data have been received from the individual who claimed to supply this information.

In authentication :

a) A Brute force attack is an automated process of trial and error used to guess a person's user name, password, credit-card number of cryptographic key.

b) Insufficient authentication occurs when a website permits an attacker to access sensitive content or functionality without having to properly authenticate.

c) Weak password recovery validation is when a website permits an attacker to illegally obtain, change or recover another user's password.

Authorization

- Authorization is a procedure of controlling the access of authenticated users to the system resources. An authorization system provides each user with exactly those rights granted to them by the administrator.
- Besides providing users with access rights to files, directories, printers etc, an authorization system might control user privileges, such as local access to the server, setting the system time, creating backup copies of the data and server shutdown.

In authorization :

a) Credential/session prediction is a method of hijacking or impersonating a website user.

b) Insufficient authorization is when a website permits access to sensitive content or functionality that should require increased access control restrictions.

c) Insufficient session expiration is when a website permits an attacker to reuse old session credentials or session IDs for authorization.

8.2.1 Password Based Authentication

- Password is a front line protection against the unauthorized access (intruder) to the system. A password authenticate the identifier (ID) and provides security to the system. Therefore almost all systems are password protected.

1. Password vulnerability

- Passwords are more common. Passwords are easily guessed. Use of mechanisms to keep passwords secret does not guarantee that the system security can not be broken. It only says that it is difficult to obtain passwords.
- The intruder can always use a trial and error method. A test of only a limited set of potential strings tends to reveal most passwords because there is a strong tendency for people to choose relatively short and simple passwords that they can remember.
- Some techniques that may be used to make the task of guessing a password difficult are as follows
  1. Longer passwords.
  2. Salting the password table.
  3. System assistance in password selection.
- The length of a password determines the ease with which a password can be found by exhaustion. For example, 3-digit password provides 1000 variations whereas a four digit passwords provides 10,000 variations. Second method is the system assistance.
- Password is generated by system or user.

2. Encrypted passwords

- Instead of storing the names and passwords in plain text form, they are encrypted and stored in cipher text form in the table.
- In this case, instead of directly using a user specified name and password for table lookup, they are first encrypted and then the results are used for table lookup.
- If the stored encoded password is read by someone, the password cannot be determined.

3. One time passwords

- Password sniffing problem is solved by password pair(login name and password). When a session starts, system selects password pair randomly and user supply remaining part of password pair.
- Here use is challenged for password, he will respond with correct answer.

- In this method, the password is different in each instance. One time passwords are among the only ways to prevent improper authentication due to password exposure. Commercial implementations of one time password system such as secur ID, use hardware calculators.

Password selection strategies

- Too short password is too easy to guess. If the passward is 8 random character, it is impossible to crack the password. In order to eliminate gaussable passwords four basic techniques are suggested.
  1. User education
  2. Computer generated password
  3. Reactive password checking
  4. Proactive password checking

University Question

1. *Define and explain following term : Authentication.* GTU : Summer-13, Marks 2

8.3 Protection Mechanisms

GTU : Summer-13, Winter-13

- Protection mechanisms are implemented in operating system to support various security policies.
- Computer system consists of hardware and software resources. Hardware resources/components are CPU, main memory segment, keyboard, mouse, printer and disk drives. Software resources/components are processes, files, database and semaphore. These all components are called as **object.**
- Unique name is assigned to each object. This name is referenced whenever any operation performed on the object. Read and write operation is performed on the files.
- Objects must be protected from subjects. **Access rights** define how various subjects can access various objects. Subjects may be users, processes, programs etc. subjects are active entities and objects are passive.
- **Protection domain** is a collection of access rights. The protection state can be conceptualized as an access matrix. Each access rights in a protection domain are represented as an order pair with fields for the object name and its corresponding privileges.
- Protection domains are extensions of the hardware supervisor mode ability.

- Protection domain is unique to a subject. A domain is a set of (object, rights) pairs. Each pair specifies an object and some subset of the operations that can be performed on it.
- For example, if a user can read and write the file "sorting.c", the corresponding ordered pair for this user's access right can be represented by <sorting.c, {read, write}>.
- Fig. 8.3.1 shows three protection domains. The association between domain and process is static or dynamic. Static mean the set of available resource to process is fixed through process lifetime.

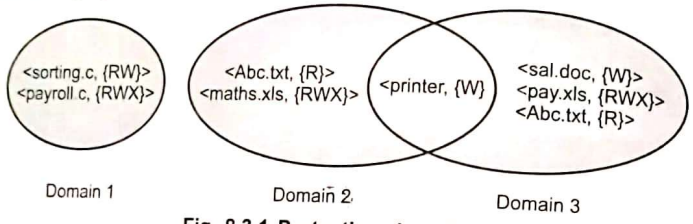


Fig. 8.3.1 Protection domain

- At every instant of time, each process runs in some protection domain. Processes can also move from one domain to other domain during execution. The rules for domain switching are highly system dependent.
- Access rights may be copied, transferred or propagated from one domain to another. Coping of access rights simply granting a right of one user to another user. When a subject no longer needs access to an object, access rights can be revoked.
- In UNIX operating system, process domain is defined by using UID and GID. Given any UID and GID combination, it is possible to make a complete list of all objects that can be accessed. Two subjects with the same UID and GID combination will have access to exactly same set of objects.
- Each file management system has its own method to control file access. The four most commonly used methods are as follows :
  1. Access control matrix
  2. Access control lists
  3. Capability lists
  4. Lockword control.

8.3.1 Access Control Matrix

- Access control matrix is easy to implement. Access matrix contains row and columns. Row is represented by domains and columns represents by objects. Each entry in the matrix consists of a set of access rights.

- The intersection of the row and column contains the access rights for that subject to that objects. Following table shows access control matrix/ protection matrix
- Each cell in the matrix specifies the actions that a subject can perform on an object.

	Sorting.c	Payroll.c	Abc.txt	Maths.xls	Sal.doc	Pay.xls	Printer
Domain1	Read, write	Read, write, execute					Write
Domain2			Read	Read, write, execute			Write
Domain3			Read		Write	Read, write, execute	

- Access matrix works well only for systems with a few files and few users.

8.3.2 Access Control Lists

- **Access control list (ACL)** is a modification of the access control matrix. It maintains a record of only those entries that specify an access rights. It was used in MULTICS operating system.
- Each object has a security attribute that identifies its access control list. For each object in an operating system, an access control list contains entries for each subject and the privileges associated with that subjects for the object.
- When subject send request for accessing an object, the operating system searches the access control list for that object to find any privileges for the subject.

8.3.3 Implementation of Access Matrix

- Access control matrix is implemented by using four different methods :
  a) Global table
  b) Access lists for objects
  c) Capability list
  d) Lock key mechanism.

1. Global table

- Global table consists of three parameters i.e. <domain, object, rights-set>. Before performing operation on an object by subject in particular domain, the global table is searched for a triples <domain, object, rights-set>.
- If this is found, the operation is performed otherwise error condition is generated.

Drawbacks :

a) Global table cannot keep in main memory because of its large size.

b) It requires additional I/O operation.

2. Capability lists

- Capability list is used for controlling device and files. An object is often represented by its physical name or address called **a capability.**
- Capability is a pointer or token that grants privileges to a subject that possesses. It is implemented as a unique object identifier. Capabilities are granted to a subject, which presents the token for all subsequent accesses to the object.
- Each capability grants the owner certain rights on a certain object. Capability lists are themselves objects and may be pointed to from other capability lists.
- Token based and pointer based capabilities are used in the system.
- Pointer based capabilities : It simplify to the address at which the object is stored.
- Token based capabilities : Capabilities are independent for the objects location in memory.
- Upon creation of an object, a capability for that object is created.
- Capability provides two functions :
  1. It provides very large address space for a resource.
  2. Possession of the capability represents the authorization of the subject to the particular object.

Implementation of capabilities :

1. It is implemented within the operating system's address space.
2. Hardware may incorporate specialized support for capabilities.

- Mach operating system provides kernel level capabilities.
- Capabilities allow a process to be encapsulated easily. It do not support selective revocation of rights.

3. Lock key mechanism

- Lock is a unique bit pattern for each object.
- Key : each domain has a list of unique bit patterns.

- When a block of memory is allocated to a process, the lock is set to the same as the process's key. Process key and memory locks can be set only while the system is in supervisor mode.
- The access is checked by hardware on each memory access by keeping the key value in a CPU register. So memory locks can be through of as access control list.
- The value of lock and key is same then only accessing of object is allowed.
- This method is simple to implement. It is efficient at runtime.

University Questions

1. *Write short note on following :*
   Access metrics mechanism. **GTU : Winter-13, Marks 4**
2. *How access control list can be useful for managing file access ?* **GTU : Summer-13, Marks 4**

❑❑❑

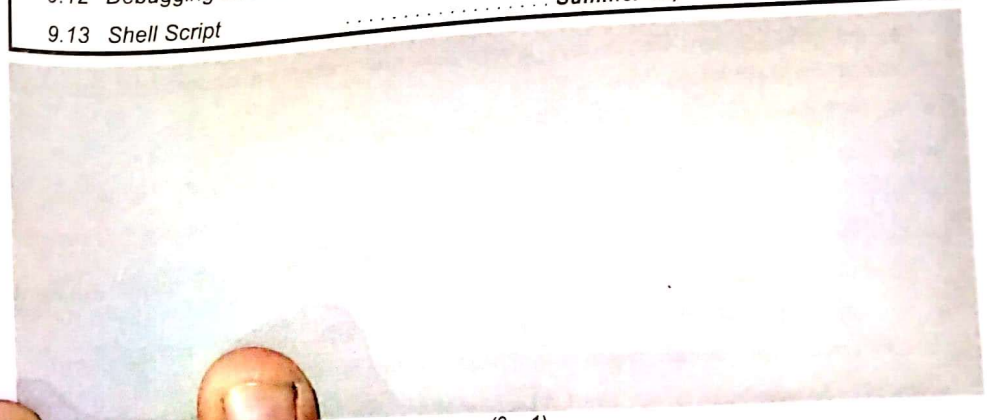
9 UNIX Operating System

Syllabus

Development of Unix, Role and function of Kernel, System calls, Elementary UNIX command and shell programming, Directory structure, System administration.

Contents

9.1 History of UNIX
9.2 Shell
9.3 UNIX Shell ..... **Winter-13** ..... Marks 4
9.4 Interfaces to UNIX
9.5 UNIX Utility Programs
9.6 UNIX File System ..... **Winter-13, Summer-13,14** · Marks 7
9.7 vi Editor
9.8 Shell Meta Characters
9.9 Environment Variable
9.10 Control Structure
9.11 Command Substitution
9.12 Debugging Shell Scripts
9.13 Shell Script ..... **Summer-13,** ..... Marks 4



9.1 History of UNIX

- In 1965 Bell Laboratories joins with MIT and General Electric in the development effort for the new operating system, multics, which would provide multi-user, multi-processor and multi-level file system, among its many forward-looking features.
- In 1969, Ken Thompson, Dennis Ritchie, Rudd Canaday and Doug McIlroy designed and implemented the first version of the Unix file system on a PDP-7 along with a few utilities. It was given the name UNIX by Brian Kernighan as a pun on Multics.
- 1970, January 1 time zero for UNIX and in 1971 the system now runs on a PDP-11, with 16 kbytes of memory, including 8 kbytes for user programs and a 512 kbyte disk. UNIX caught on among programmers because it was designed with these features :
  a. Programmers environment
  b. Simple user interface
  c. Simple utilities that can be combined to perform powerful functions
  d. Hierarchical file system
  e. Simple interface to devices consistent with file format
  f. Multi-user, multi-process system
  g. Architecture independent and transparent to the user.
- In 1973 UNIX is re-written mostly in C, a new language developed by Dennis Ritchie.
- In 1974 Thompson and Ritchie publish a paper in the communications of the ACM describing the new Unix OS. This generates enthusiasm in the academic community which sees a potentially great teaching tool for studying programming systems development. Since AT&T is prevented from marketing the product due to the 1956 Consent Decree they license it to universities for educational purposes and to commercial entities.
- In 1977 there are now about 500 Unix sites world-wide.
- In 1980 BSD 4.1 (Berkeley Software Development).
- In 1983 SunOS, BSD 4.2, SysV.
- In 1984 there are now about 1,00,000 Unix sites running on many different hardware platforms, of vastly different capabilities.
- In 1988 AT&T and Sun Microsystems jointly develop System V Release 4 (SVR4). This would later be developed into UnixWare and Solaris 2.
- In 1993 Novell buys UNIX from AT&T.

- In 1994 Novell gives the name "UNIX" to X/OPEN.
- Reasons for the popularity and success of the UNIX system are as follows :
  1. The system is written in a high level language, making it easy to read, understand, change and move to other machines.
  2. It has simple user interface.
  3. It uses a hierarchical file system that allows easy maintenance and efficient implementation.
  4. It uses a consistent format for files, the byte stream, making application programs easier to write.
  5. It provides a simple, consistent interface to peripheral devices.
  6. It is a multiuser, multiprocess system.
  7. It hides the machine architecture from the user.

9.1.1 UNIX Features

- Unix was designed to be a time sharing system.
- Unix has a simple user interface that has the power to provide the services that user want.
- It uses a hierarchical file system that allows easy maintainance and efficient implementation.
- Unix file system is a multilevel tree.
- Each user data file is simply a sequence of bytes.
- It provides a simple, consistent interface to peripheral device.
- Unix supports multiple processes.
- Unix is a multiuser, multiprocess system.
- CPU scheduling in Unix is a simple priority algorithm.
- Swapping is used if a system is suffering from excess paging.
- Unix hides the machine architecture from the user.
- The system is written in high level language.
- Unix consists of two separable parts : The Kernel and the systems programs.
- The Kernel interacts with the machines hardware.
- Shell interacts with the user.
- Viewing [illegible]ystem as a set of layers, the operating system is commonly called the Kerne[illegible]

- The Kernel provides the file system, CPU scheduling, memory management through system calls.

9.1.2 General UNIX Architecture

- UNIX is a layered operating system. The innermost layer is the hardware that provides the services for the operating system.
- The operating system, referred to in UNIX as the Kernel, interacts directly with the hardware and provides the services to the user programs. These user programs do not need to know anything about the hardware. They only know how to interact with the Kernel and it's up to the Kernel to provide the desired service.
- User programs interact with the Kernel through a set of standard system calls. These system calls request services to be provided by the Kernel.
- These services would include accessing a file : open, close, read, write, link or execute a file; starting or updating accounting records; changing ownership of a file or directory; changing to a new directory; creating, suspending, or killing a process; enabling access to hardware devices; and setting limits on system resources.
- UNIX is a multi-user, multi-tasking operating system. Many users logged into a system simultaneously, each running many programs.
- It is the Kernel's job to keep each process and user separate and to regulate access to system hardware, including CPU, memory, disk and other I/O devices.
- Fig. 9.1.1 shows the general architecture of the UNIX system.
- The underlying hardware is surrounded by the operating system software. The operating system is often called the system Kernel or simply the Kernel, to emphasize its isolation from the user and applications.
- UNIX comes equipped with a number of user services and interfaces that are considered part of the system. These can be grouped into the shell, other interface software and the components of the C compiler.

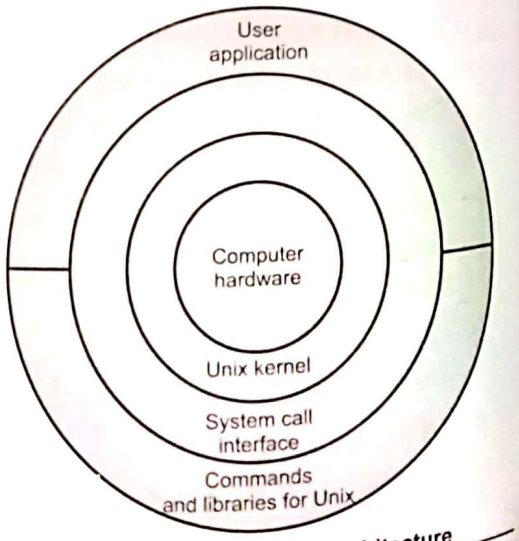


Fig. 9.1.1 General unix architecture

- The layer outside of this consists of user applications and the user interface to the C compiler.
- Programs such as the shell and editors interact with the Kernel by invoking a well defined set of system calls.

9.1.3 Architecture of UNIX OS

- Fig 9.1.2 shows traditional UNIX Kernel architecture.
- There are three levels :
  a. User level b. Kernel level c. Hardware level.
- The file subsystem manages files, allcoating file space, controlling access to files and retrieving data for users. Processes interact with the file subsystem via a specific set of system calls.
- The file subsystem accesses file data using a buffering mechanism that regulates flow between the Kernel and secondary storage devices.
- The buffering mechanism interact with block I/O device drivers to initiate data transfer to and from the Kernel.
- Hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing.
- The Kernel may resume execution of the interrupted process after servicing the interrupt.
- The process control subsystem is responsible for process synchronization, IPC, memory management and process scheduling.
- The file subsystem and the process control subsystem interact when loading a file into memory for execution. The process subsystems reads executable files into memory before executing them.
- The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the Kernel moves them between main memory and secondary memory so that all processes get a fair chance to execute.
- The scheduler module allocates the CPU to processes.
- The **scheduler** module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the Kernel preempts them when their recent run time exceeds a time quantum.

- The hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing.

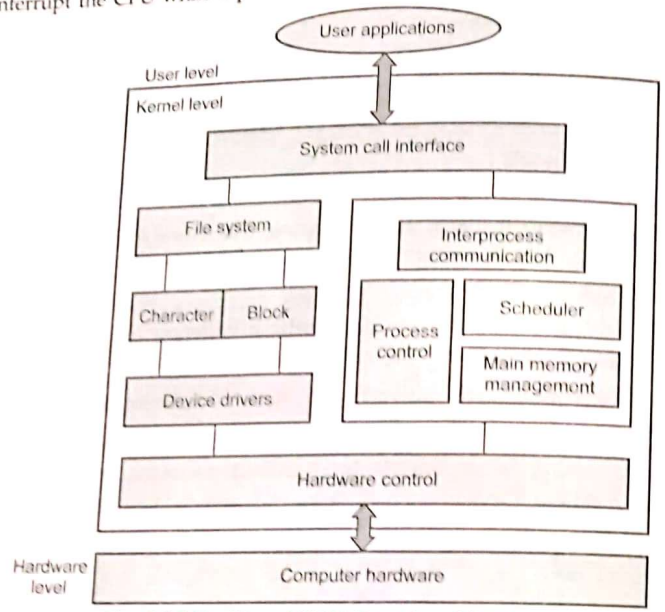


Fig. 9.1.2 Traditional UNIX Kernel architecture

9.1.4 Modern UNIX Kernel

- Fig. 9.1.3 shows the modern UNIX Kernel system. (See Fig. 9.1.3 on next page)
- There is a small core of facilities, written in a modular fashion, that provide functions and services needed by a number of OS processes.
- Each of the outer circles represents functions and an interface that may be implemented in a variety of ways.
- Examples of modern UNIX systems are as follows.

System V Release 4 (SVR4)

- Developed jointly by AT&T and Sun Microsystems and combines features from SVR3, 4.3BSD, Microsoft Xenix System V and SunOS.
- New features in the release include real-time processing support, process scheduling classes, dynamically allocated data structures, virtual memory management, virtual file system and a preemptive Kernel.

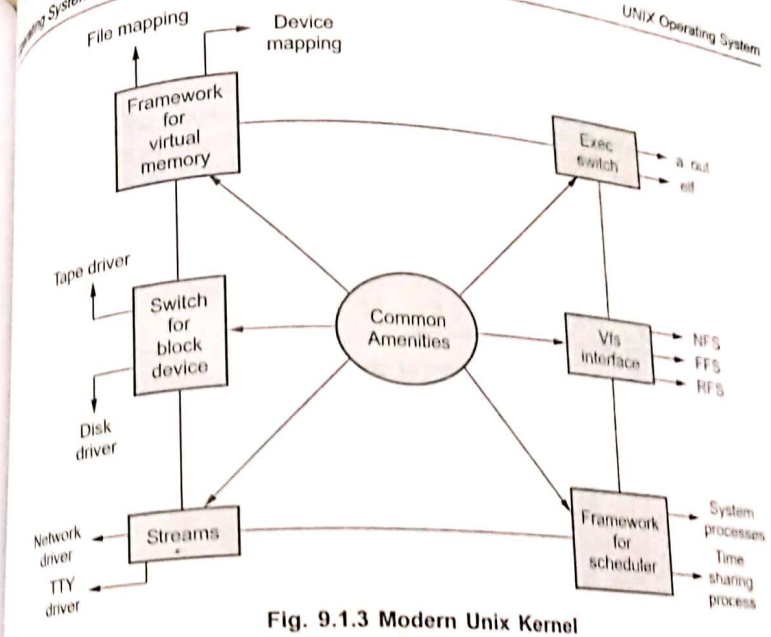


Fig. 9.1.3 Modern Unix Kernel

- SVR4 is running on machines ranging from 32-bit microprocessors up to supercomputers.

4.4 BSD

- BSD series of UNIX releases have played a key role in the development of OS design theory.
- Most enhancements to UNIX first appeared in BSD versions.
- It is a major upgrade to 4.3BSD and includes a new virtual memory system, changes in the kernel structure and a long list of other feature enhancements.

9.2 Shell

- A shell is a program that acts as the interface between you and the UNIX system, allowing you to enter commands for the operating system to execute.
- A UNIX shell is both a command interpreter and a programming language. As a command interpreter, the shell provides the user interface to the rich set of gnu utilities. The programming language features allow these utilities to be combined.
- Files containing commands can be created and become commands themselves. These new commands have the same status as system commands in directories such as **/bin**, allowing users or groups to establish custom environments to automate their common tasks.

- Shells may be used interactively or non-interactively. In **interactive mode**, they accept input typed from the keyboard. When executing **non-interactively**, shells execute commands read from a file.
- A shell allows execution of gnu commands, both synchronously and asynchronously. The shell waits for synchronous commands to complete before accepting more input, asynchronous commands continue to execute in parallel with the shell while it reads and executes additional commands.
- The redirection constructs permit fine-grained control of the input and output of those commands. Moreover, the shell allows control over the contents of commands' environments.
- Fig. 9.2.1 shows shell and Linux kernel.
- Shells also provide a small set of built-in commands implementing functionality impossible or inconvenient to obtain via separate utilities. For example, cd, break, continue and exec cannot be implemented outside of the shell because they directly manipulate the shell itself.
- The history, getopts, kill or pwd builtins, among others, could be implemented in separate utilities, but they are more convenient to use as builtin commands.

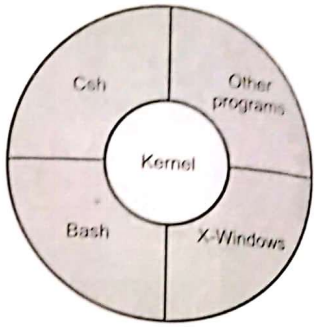


Fig. 9.2.1

- While executing commands is essential, most of the power of shells is due to their embedded programming languages. Like any high-level language, the shell provides variables, flow control constructs, quoting and functions.
- Shells offer features geared specifically for interactive use rather than to augment the programming language. These interactive features include job control, command line editing, command history and aliases.

9.2.1 Shell's Responsibilities

- Fig. 9.2.2 shows shell responsiilities

1. **Program execution :** The shell is responsible for the execution of all programs. Each time you type in a line to the shell, the shell analyzes the line and then determines what to do. The line that is typed to the shell is known more formally as the command line.

2. **Variable and filename substitution :** Like any other programming language, the shell lets you assign values to variables. Whenever you specify one of these variables on the command line, preceded by a dollar sign, the shell substitutes the

value assigned to the variable at that point. The shell also performs filename substitution on the command line.

3. **I/O Redirection :** It scans the command line for the occurrence of the special redirection characters <, >, or >>

4. **Pipeline hookup :** Just as the shell scans the command line looking for redirection characters, it also looks for the pipe character |. For each such character that it finds, it connects the standard output from the command preceding the | to the standard input of the one following the |. It then initiates executior both programs.

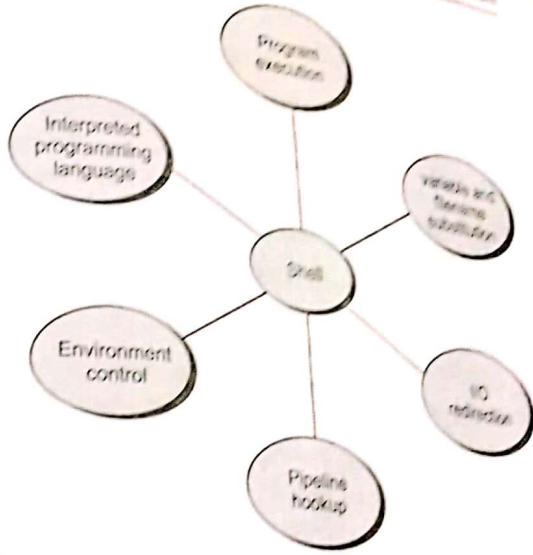


Fig. 9.2.2

5. **Environment control :** The shell provides certain commands that let you custor your environment. Your environment includes your home directory, the chara that the shell displays to prompt you to type in a command, and a list o directories to be searched whenever you request that a program be executed.

6. **Interpreted programming language :** The shell has its own built-in program language. This language is interpreted, meaning that the shell analyzes statement in the language one line at a time and then executes it. Pro developed in interpreted programming languages are typically easier to debu modify than compiled ones.

9.3 UNIX Shell

GTU : Wint

- The shell is an interactive interpreter : It reads commands, finds the corresp programs, runs them and displays output.
- The shell interprets the first word of a command line as a command nam supports the three types of commands :

1. Executable file command : It contains object code produced by compilation of source code.
2. Executable file : It contains sequence of shell command lines.
3. Internal shell command : It makes the shell a programming language in addition to a command interpreter and includes commands for looping.

- The most commonly used shells are SH(Bourne SHell) CSH(C SHell) and KSH(Korn SHell). KSH is based on SH and so is BASH (Bourne again shell). TCSH (Extended C SHell) is based on CSH.
- A built-in command is part of the shell itself. A utility is an executable program that is installed usually in a system directory such as "/bin" directory which the shell executes.
- Any process executing a shell can itself execute another shell that will perform some task and return to the **original shell**. This is similar to the parent process and child process paradigm. The first shell is called the **parent** to the shell it executes. The new shell is the **child** to the original shell.

9.3.1 Bourne Shell

- The Bourne shell is one of a number of UNIX shell. Like the others, it is both a command language and a programming language. As a command language it provides a user interface to Unix/Linux. It executes commands entered by the user or from a file.
- Files containing commands allow users build their own commands thus tailoring the system to their own needs. Such files are called : **shell scripts, shell programs** or **command files.** These commands have access to the command line parameters and have the same status as other UNIX commands.
- As a programming language, each shell provides I/O, variables, conditionals, loops, and switches. The syntax is aimed at ease of use at a terminal so that strings for example do not have to be quoted. Each shell (Bourne, C, Bash etc) has its own syntax.
- System administrators can build shell programs to add new users, to shutdown the system and so on. Some examples are given in these notes.

9.3.2 C Shell

- The C shell was written by Bill Joy at the University of California at Berkeley. *Csh* is a new command language interpreter for UNIX systems. It incorporates good features of other shells and a *history* mechanism similar to the *redo* of INTERLISP. While incorporating many features of other shells which make writing shell

programs easier, most of the features unique to *csh* are designed more for the interactive UNIX user.

The C shell has three separate files which are used for customizing its environment. These three files are *.cshrc*, *.login*, and *.logout*. Because these files begin with a period (.) they do not usually appear when one types the **ls** command. In order to see all files beginning with periods, the **-a** option is used with the **ls** command.

The *.cshrc* file contains commands, variable definitions and aliases used any time the C shell is run. When one logs in, the C shell starts by reading the *.cshrc* file, and sets up any variables and aliases.

The C shell reads the *.login* file after it has read the *.cshrc* file. This file is read once only for login shells. This file should be used to set up terminal settings, for example, backspace, suspend and interrupt characters.

The *.logout* file contains commands that are run when the user logs out of the system.

9.4 Bash

- Bash is the shell or command language interpreter, for the **gnu** operating system. The name is an acronym for the '**Bourne-Again SHell**'.
- Bash is largely compatible with sh and incorporates useful features from the Korn shell ksh and the C shell csh.
- Bash is based on the Bourne shell, sh, originally written by Stephen Bourne.
- Bash is extremely portable and can be built on most UNIX systems because many of the environment dependent variables are determined at build time. Bash has been ported as a shell for several non-UNIX platforms like QNX, Minix, OS/2, Windows 95 and Windows NT.

9.5 Shell as a Programming Language

- User can type in a sequence of commands and allow the shell to execute them interactively. Also user can store these commands in a file which you can invoke as a program.

1. **Interactive programs :** Interactive shell determines the command-line dialect and user will use to interact with machine. Shell also performs wildcard expansion.
2. **Creating a script :** User can create script file using any text editor. User can create file using any text editor. The line #!/bin/sh is special and tells the system to use the */bin/sh* program to execute this program.
3. **Making a script executable :** There are two ways to execute the script.

a. Invoke the shell with the name of the script file as a parameter, thus :

/bin/sh first.sh

b. Change the mode of the script to executable and then after execute it by just typing its name.

chmod +x first.sh
first.sh

University Question

1. *Give the functions of following UNIX commands : grep, cat, chmod.*

GTU : Winter-13, Marks 3

9.4 Interfaces to UNIX

- *It is important to recognize that in UNIX the GUI is separate from the operating system itself. This is unlike Windows OS and earlier which won't run without the GUI.*
- *The GUI on UNIX is a separate layer that sits on top of the core operating system. In principle, you can uninstall the GUI and still have a completely functional UNIX system sans any graphical programs.*
- *GUI that runs on Linux and many UNIX operating systems. Programmers may develop applications that run on the X Window system and support GUI components, such as windows, dialog boxes, buttons, and pull-down menus*
- The X Window system was developed as part of Project Athena at MIT. In 1987, X Version 11 is released. X is now controlled and maintained by the Open Group.
- X Window uses a bit-mapped display where each pixel can be manipulated individually. The entire display is known as the root window, and individual applications are displayed as windows on this root window.
- It is possible to change the appearance of a window instantly by running a separate program after starting X. This program is called the window manager.
- X splits an application into two components : ***client and server***
- The server program controls the monitor, keyboard and mouse, while the application itself is the client. The X also runs in a TCP/IP network, it is possible for a client to run on one machine and have its display on another. The x-host client controls access to the server.
- The desktop system from which user run a program is called the **X server**. The system that hosts and executes the program is called the **X client**. This is the opposite of normal networking terminology

- The X Window System is a Graphical User Interface (GUI) that runs many on UNIX and Linux systems. The top layer of the X Window System is the Windows Manager. Desktops are used with a Window Manager, providing specific appearance, applications, and resources.
- The **window manager** is a special X client that controls the placement and movement of applications, provides title bars and control buttons, menus and taskbars. Some support virtual desktops and very fancy graphics. Classic window managers include : *twm, mwm, olwm, fvwm.*

9.5 UNIX Utility Programs

- From a certain point of view, Linux can be seen primarily as a set of utilities bonded together by a common kernel.
- Linux utilities come in two basic flavors : open source and commercial. Some commercial products, such as RealPlayer, are available for free, while others, such as the BRU tape backup software, are available for a fee. The majority of Linux utilities are free, open-source packages.
- Traditional UNIX utilities evolved in the command-line environment. Because of that they can be joined together to form powerful shell scripts and time based jobs. Some of the newer, graphical utilities are designed to be clicked and pointed at in the X Window System.

9.5.1 Internal and External Commands

- Linux commands are nothing but the text written in the shell or terminal that the terminal understands. Linux commands usually perform certain specific operations such as editing a text file, making, removing and moving a directory etc. The shell or the terminal of Linux provides a use set of commands and allows us to write the script that is able to perform various operations.
- Commands in Linux are usually divided into two sections internal commands and the external commands.

1. **External commands :** These are the most commonly used utilities and programs such as **cat, ls** and so forth. External commands are the commands that are executed by the kernel. These commands will have a process id running for it.
2. **Internal commands :** The shell has a number of built-in commands such as **cd** and **echo** which don't generate a process. Internal commands are the commands that are executed directly by the shell. These commands will not have a separate process running for each.

9.5.2 File Handling Utilities

- We discuss some of the more important directory and file handling commands.

1. pwd : Print (current) working directory

- *pwd* displays the full absolute path to your current location in the file system. So,

```
$ pwd <Enter>
/usr/bin
```

implies that */usr/bin* is the current working directory.

2. ls : list directory

- *ls* lists the contents of a directory. If no target directory is given, then the contents of the current working directory are displayed. So, if the current working directory is /,

```
$ ls <Enter>
bin  dev  home  mnt  share  usr  var
boot  etc  lib  proc  sbin  tmp  vol
```

- Actually, *ls* doesn't show you all the entries in a directory - files and directories that begin with a dot (.) are hidden.
- The reason for this is that files that begin with a. usually contain important configuration information and should not be changed under normal circumstances. If you want to see all files, *ls* supports the *-a* option :

```
$ ls -a
```

- Even this listing is not that helpful - there are no hints to properties such as the size, type and ownership of files, just their names.
- To see more detailed information, use the *-l* option (long listing), which can be combined with the *-a* option as follows :

```
$ ls -l /user/rakshita <Enter>
total 3
-rw-r--r--   1   rakshita student  13   Jan 1    10:39   data1
-rwxrwxrwx   1   rakshita student  19   Feb 10   21:35   data2
drwxrwxr--   2   rakshita student  32   Jan 16   16:50   prog-files
$
```

- The first character in column 1 can be '-' (ordinary file), 'd' (directory file) or 'b', 'c', 'p' (special file).
- The output of the ls -l command can be explained as in Table 9.5.1.

Column	Description
1	File type and FAP
2	Number of links
3	File Owner
4	Group Owner (group name)
5	File size (in bytes)
6, 7 and 8	Day and time of last modification in the file
9	Name of file

Table 9.5.1 Output of ls -l command

3. cd : Change directory

- Allows you to change your current working directory to any other you have permission to enter. Change directory by itself sends you to your home dir. The address can be relative or absolute.

```
$ pwd <Enter>
/home/iresh
$ cd program <Enter>
$ pwd <Enter>
/home/iresh/program
$
```

Another use of cd command

```
$ pwd <Enter>
/user/rakshita
$ cd /user <Enter>
$ pwd <Enter>
/user
$
```

- Note that the complete path-name has been specified with the cd command. UNIX also allows relative path-names with commands. For example, rakshita can enter the following command after logging in, to change to the parent directory of his HOME directory :

```
$ pwd <Enter>
/user/rakshita
$ cd .. <Enter>
$ pwd <Enter>
/user
$
```

- The two dots refer to the parent directory of the current directory. The cd command without any path-name always takes the user back to the HOME directory.

4. mkdir : Make directory

- The *mkdir* command is used to create directories.

```
$ mkdir prog-files <Enter>
$
```

- The subdirectory prog-files is created under the current directory. However, the new directory does not become the current directory. Complete path-names can be specified with *mkdir*

```
$ mkdir /user/rakshita/prog-files <Enter>
```

5. rmdir : Removing a directory

- The *rmdir* removes the directory specified.

```
$ rmdir cob-prog <Enter>
$
```

where cob-prog is the directory which is deleted

- A directory can be deleted only if it is :

i. Empty (does not contain files or subdirectories)

ii. Not the current directory.

- Complete path-names may also be specified with rmdir.

```
$ rmdir /user/gomes/cob-prog <Enter>
```

6. cat : Concatenate

- The *cat* command displays the contents of the file specified.

```
$ cat data1 <Enter>
A sample file
$
```

- The command assumes that data1 is in the current. Complete path-names can also be specified to display a file in another directory. The *cat* command can also display more than one file as shown in the following command :

```
$ cat data1 data2 <Enter>
A sample file
Another sample file          (contents of data2)
$
```

7. cp : Copy

- The *cp* command duplicates the contents of the source file into a target file.

```
$ cp data1 data3 <Enter>
```

- In the above example, the contents of data1 are copied to a new file data3. If data3 already exists, its contents will be overwritten by the contents of data1.
- Complete path-names can be specified with the *cp* command to copy files across directories

8. rm : Removing files

- The *rm* (remove) command is used to delete files.

```
$ rm data1 <Enter>
$
```


- If the file is not located in the current directory, the complete path-names have to be given.

```
$ rm /user/gomes/data1 <Enter>
```

9. mv : Move (Renaming files)

- The *mv* command changes the name of a file or directory. It can also be used to move files from one directory to another.

```
$ mv data3 newfile <Enter>
$
$ mv /user/rakshita/prog-files  /user/rakshita/programs <Enter>
$
```

- In UNIX, a file can be moved to another directory (not copied) as shown below :

```
$ mv data3 /user/rakshita/programs/data3 <Enter>
$
```

9.5.3 Security File Permissions

- File Access Permissions (FAP) refer to the permissions associated with a file with respect to the following :

1. The File Owner
2. The Group Owner
3. Other Users.

Table 9.5.2 summarizes the access permissions available.

Access type	Denoted by	What it means
Read	r	Allows displaying, copying and compiling a file
Write	w	Allows editing and deleting of a file
Execute	x	Allows execution of a file

Table 9.5.2

- The File Access Permissions for a directory is to be interpreted in the following manner :

a. Without read permission, the user is not allowed to :
   - List contents of directory
   - Remove the directory using *rmdir*.

b. Without *write* permission, the user is not allowed to :
   - Copy files to the directory
   - Remove files from the directory
   - Rename files in the directory

- Make a subdirectory
- Remove a subdirectory from the directory
- Move files to and from the directory.

c. Without *execute* permission, the user is not allowed to :

- Display the contents of a directory file from within the directory
- Change to the directory
- Display a file in the directory
- Copy a file to or from the directory.

1. Wildcard characters

- The command is executed on the files with names that match the pattern.
- Table 9.5.3 lists the wildcards with a description.

Character	Purpose
*	Matches none or one character or a string of more than one character
?	Matches exactly one character
[ ]	Matches exactly one of a specified set of characters

Table 9.5.3

2. *chmod* command

- Access permissions associated with a file or directory can be changed using the *chmod* command. Permissions associated with a file can be changed only by the owner of the file.
- Consider a file **prime.c** with the following permissions :

```
-r w x r w x r w x   1   rakshita   student   20   Jan   1   11:20   prime.c
```

- To change the permissions for the file, the File Owner has to specify :
  1. The type of user for whom permission is to be changed
  2. The type of permission which is to be changed
  3. Whether the permission is to be given or revoked
  4. The name of the file for which permissions are to be changed.
- UNIX allows you to change the FAP for a specific user-type. The command to do that is :

```
$ chmod u+r prime.c <Enter>
$
```

- Here,

'u' = indicates File Owner,

'+' = indicates that the permission is to be given,

'r' = indicates the read permission and prime.c is the filename.

- The File Access Permissions for the file prime.c will now appear as follows

```
-r w x r w x r w x   1   rakshita   student   20   Jan   1   11:20   prime.c
```

- To remove the read permission from the Group Owner, the command is

```
$ chmod g-r prime.c <Enter>
$
```

Here,

'g' = indicates Group Owner

'-' = indicates that the permission is to be removed.

- The permissions will now appear as follows :

```
-r w x - w x r w x   1   rakshita   student   20   Jan   1   11:20   prime.c
```

- To remove the execute permission from the Other Users, the command is :

```
$ chmod o-x prime.c <Enter>
$
```

Here, 'o' indicates Other Users

- The permissions will now appear as follows :

```
-r w x - w x r w-   1   rakshita   student   20   Jan   1   11:20   prime.c
```

- UNIX also allows the File Owner to change permissions for all categories of users for a file.
- To remove *write* permission from all users, the command is :

```
$ chmod -w prime.c <Enter>
$
```

- The FAP for the file prime.c will now appear as follows :

```
-r - x - - x r - -1   rakshita   student   20   Jan   1   11:20   prime.c
```

- To add back the *write* permission, the File Owner can give the following command :

```
$ chmod +w prime.c <Enter>
$
```

9.5.4 Process Utilities

1. ps : Process status

- *ps* command display some process attributes. The basic unit of execution in UNIX is called a process.

- If you run the Linux ps command by itself, it only shows very basic information about the processes you are currently running. For example, if you issue the basic command like this without any arguments :

```
$ ps
```

Output from this command looks something like this :

```
  PID  TTY          TIME CMD
 4343  ttys000    0:00.35 -bash
 2617  ttys001    0:00.65 -bash
18501  ttys003    0:00.27 -bash
```

- The PID column shows the process-id, the second column shows the TTY (terminal) the process is attached to, the TIME column shows how much CPU time the process has used, and the CMD column shows the command that is running.
- *ps command* **options** are :

Options	Meaning
-a	Displays all processes on a terminal, with the exception of group leaders.
-c	Displays scheduler data.
-d	Displays all processes with the exception of session leaders.
-e	Displays all processes.
-f	Displays a full listing.
-glist	Displays data for the list of group leader IDs.
-j	Displays the process group ID and session ID.
-l	Displays a long listing.
-plist	Displays data for the list of process IDs.
-slist	Displays data for the list of session leader IDs.
-tlist	Displays data for the list of terminals.
-ulist	Displays data for the list of usernames.

Example

```
$ ps -ef <Enter>
```


This leads to much more output :

UID	PID	PPID	C	STIME TTY	TIME CMD
root	1	0	0	Oct21 ?	00:00:01 init [3]
root	2	1	0	Oct21 ?	00:00:00 [migration/0]
root	3	1	0	Oct21 ?	00:00:00 [ksoftirqd/0]
root	4	1	0	Oct21 ?	00:00:00 [watchdog/0]
root	5	1	0	Oct21 ?	00:00:00 [migration/1]
root	6	1	0	Oct21 ?	00:00:00 [ksoftirqd/1]
root	7	1	0	Oct21 ?	00:00:00 [watchdog/1]
root	8	1	0	Oct21 ?	00:00:00 [events/0]
root	9	1	0	Oct21 ?	00:00:00 [events/1]
root	10	1	0	Oct21 ?	00:00:00 [khelper]
root	11	1	0	Oct21 ?	00:00:00 [kthread]
root	15	11	0	Oct21 ?	00:00:00 [kblockd/0]
root	16	11	0	Oct21 ?	00:00:00 [kblockd/1]
root	17	11	0	Oct21 ?	00:00:00 [kacpid]
root	91	11	0	Oct21 ?	00:00:00 [cqueue/0]
root	92	11	0	Oct21 ?	00:00:00 [cqueue/1]
root	95	11	0	Oct21 ?	00:00:00 [khubd]

2. nice

- Processes in the UNIX system are usually executed with equal priority. UNIX offers the nice command, which is used with the and operator to reduce the priority of the jobs.
- In the UNIX scheme of things, this priority is indicated by a number that can vary from a –20 to a +19 and is known as the nice number of the task. The programs with the highest priority have the lowest nice value, so a value of –20 makes a task important in the UNIX scheme.
- The nice command is used with its -n option, along with an argument in the range of –20 to 19, in order from highest to lowest priority (the lower the number, the higher the priority).

- Process started with a value of 18 cannot be lowered to 10 with nice –10, except by root.
- Nice is a built-in command in the C-shell. Nice values are system dependent.
- Syntax : *nice -number argument*
- For instance, to execute a program called a.out at a low priority, enter

```
$ nice -n 19 a.out <Enter>
```

- If you have already started a process without the nice command, you can use the *renice* command to change the priority of the process.

Syntax : *renice -number process_id*

9.5.5 Disk Utilities

1. The df Command

- The df command lists the current total disk usage for all file systems accessible by your workstation.

Syntax : df -k

2 The *du* command

- To determine your current disk quota usage, use the du command.

Syntax : *du [options] directory*

- By default, the *du* command by itself will give a summary of quota usage for the directory specified and all subdirectories below it.

du /users/username

- For a summary of total disk usage, use the *-s* option.

du -s /users/username

Networking commands

1. FTP : File Transfer Protocol

- Transfer files to or from a remote computer on the Internet. Some computers permit anonymous FTP that is accessible to anyone on the network, even if you don't have an account on that computer.
- To connect your local machine to the remote machine, type.

ftp machinename

where machinename is the full machine name of the remote machine, e.g. vtubooks.com.

- If the name of the machine is unknown, you may type

ftp machinennumber

- where *machinennumber* is the net address of the remote machine, e.g. 129.13.45.123.

- In either case, this command is similar to logging onto the remote machine. If the remote machine has been reached successfully, FTP responds by asking for a loginname and password.
- When you enter your own loginname and password for the remote machine, it returns the prompt

ftp>

and permits you access to your own home directory on the remote machine. You should be able to move around in your own directory and to copy files to and from your local machine using the FTP interface commands given below :

- **Common FTP commands**

Commands	Meaning
?	To request help or information about the FTP commands
ascii	To set the mode of file transfer to ASCII
binary	To set the mode of file transfer to binary
mput	To copy multiple files from the local machine to the remote machine
open	To open a connection with another computer
bye	To exit the FTP environment
cd	To change directory on the remote machine
close	To terminate a connection with another computer
delete	To delete (remove) a file in the current remote directory
get	To copy one file from the remote machine to the local machine
help	To request a list of all available FTP commands
lcd	To change directory on your local machine
ls	To list the names of the files in the current remote directory
mkdir	To make a new directory within the current remote directory
mget	To copy multiple files from the remote machine to the local machine
put	To copy one file from the local machine to the remote machine
pwd	To find out the pathname of the current directory on the remote machine
quit	To exit the FTP environment
rmdir	To remove (delete) a directory in the current remote directory

2. Telnet

- Telnet operates in a client/server environment, meaning that the remote machine is configured as a server, and consequently waits for the other machine to request a service from it.
- The remote machine is sending data to be displayed; the user feels like they are working directly on the remote machine.
- In UNIX, the service is provided by what is called a daemon, a small task that runs in the background.
- The command to initiate a Telnet session is usually :

```
telnet server_name
```

server_name represents the name or IP address of the remote machine that the user wants to connect to.

- You can also give its IP address, for example :

```
telnet 125.63.121.77
```

- Finally, you can also specify which port to use by putting the port number after the IP address or server name :

```
telnet 125.63.121.77 80
```

Telnet commands

Command	Description
?	show help
close	Close Telnet session
display	Show connection settings onscreen
environ	For defining the operating system's environmental variables
logout	For logging out
mode	Switches between the transfer modes ASCII and BINARY
open	Opens another connection from the current one
quit	Leaves the Telnet application
set	Changes the connection settings

9.6 UNIX File System

GTU : Winter-13, Summer-13,14

- Root file system contains the root directory and a minimal set of files and subdirectories. File containing the Kernel is called **unix** or **vmunix,** and it resides in either the root directory or in a subdirectory such as **/Kernel** or **/stand.**
- All of the files in the UNIX file system are organized into a multi-leveled hierarchy called a directory tree. A family tree is an example of a hierarchical structure that represents how the UNIX file system is organized. The UNIX file system might also be envisioned as an inverted tree or the root system of plant.
- At the very top of the file system is single directory called "root" which is represented by a / (slash). All other files are "descendents" of root. The number of levels is largely arbitrary, although most UNIX systems share some organizational similarities.
- A file is logically a container for data. Users may create a file and store data by writing it to the file. UNIX file does not contain the end of mark. File size is not stored in the file and even its name.
- File permits both sequential and random access to its data. The Kernel provides control operations to name, organize, and control access to files. The Kernel does not interpret the content or structure of a file.
- A directory is a file that holds numbers and names of the other files. UNIX treats directories and devices as files as well. All physical devices like hard disk, printer, CD-ROM and modem are treated as files. The shell is also file.
- Three important device files found in both UNIX and Linux are */dev/console*, */dev/tty*, and */dev/null*.
- ***/dev/console :*** This device represents the system console. Error messages and diagnostics are often sent to this device.
- ***/dev/tty :*** It is logical device used for controlling terminal of a process.
- ***/dev/null :*** The */dev/null* file is null device. All output written to this device is discarded. An immediate end of file is returned when the device is read.
- UNIX organizes files in a hierarchical, tree structured name space. Fig. 9.6.1 shows the file organization.
- The tree consists of files and directories, with the files at the leaf nodes. The directory contains name information about the files and other directories that resides in it.

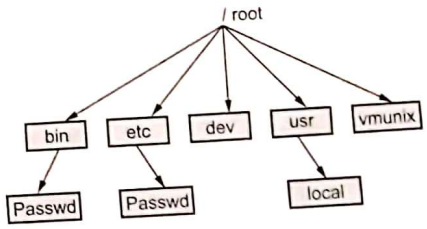


Fig. 9.6.1

- A directory entry for a file is called a **hard link** to the file. Any file may have one or more links to it, either in the same or in different directories.
- Symbolic links are developed to overcome several shortcomings of hard links :
  1. Symbolic links can link files across file systems
  2. Symbolic links can link directory files
  3. Symbolic links always reference the latest version of the files to which they link.
- A directory entry for a file is called a **hard link.** Modern UNIX file systems also provide another type of link called a **symbolic link.**
- The *symlink* system call creates a symbolic link. It is a special file that points to another file. The file type attribute identifies it as a symbolic link. A pathname contained in the symbolic link may be absolute or relative.

Difference between hard link and symbolic link

Hard link	Symbolic link
Does not create a new inode	Create a new inode
Cannot link directories unless it is done by root	Can link directories
Cannot link files across file systems	Can link files across file system
Increase hard link count of the linked inode	Does not change hard link count of the linked inode

- Part of the root filesystem are :
  1. ——**/dev** -- for device files.
  2. ——**/etc** -- for critical system files.
  3. ——**/sbin** and **/bin** -- for important utilities.
  4. ——**/tmp** -- for temporary files.
- Some systems keep shared libraries and the C preprocessor in the /lib directory. Others have moved this into **/usr/lib.**
- **/usr** contains most of the user programs and online manuals etc.
- **/var** provides a home for spool directories, log files, accounting information etc.
- Home directories of users should be kept on different file systems and mounted beneath **/usr.**

9.6.1 File Types

- File is divided into following types :
  1. Ordinary file 2. Directory file 3. FIFO file

4. Character device file 5. Block device file

Ordinary file

- Also called regular file. It contains only data as a stream of characters. An ordinary file cannot contain another file, or directory.
- It is either text file or a binary file.
- Examples of ordinary files include simple text files, application data files, files containing high-level source code, executable text files, and binary image files.
- **Text file :** It contains only printable characters.
- **Binary file :** It contains printable and unprintable characters that cover the entire ASCII range.
- Regular files may be created, browsed through, and modified by various means such as text editors.

Directory file

- A directory file is like a folder that contains other files, including subdirectory files.
- Directory files act as a container for other files, of any category. Directory files don't contain data in the user sense of data; they merely contain references to the files contained within them.
- Directory is created in UNIX by the *mkdir* command.
- The UNIX directory is considered empty if it contains no other files except the "." and ".." .
- Directory is removed by using *rmdir* command. Content of directory is displayed by ls command.

Device file

- Special files are also known as device files. In UNIX all physical devices are accessed via device files; they are what programs use to communicate with hardware.
- Files hold information on location, type, and access mode for a specific device.
- There are two types of device files; **character** and **block**.
- **Block device** files are used to access block device I/O. Block devices do buffered I/O, meaning that the data is collected in a buffer until a full block can be transferred.
- **Character device** files are associated with character or raw device access. They are used for un-buffered data transfers to and from a device. Rather than transferring data in blocks the data is transferred character by character.

- One transfer can consist of multiple characters.
- Some devices, such as disk partitions, may be accessed in block or character mode. Because each device file corresponds to a single access mode, physical devices that have more than one access mode will have more than one device file.
- Device files are found in the */dev* directory. Each device is assigned a major and minor device number.
- The major device number identifies the type of device, i.e. all SCSI devices would have the same number as would all the keyboards.
- The minor device number identifies a specific device, i.e. the keyboard attached to this workstation.
- Device files are created using the *mknod* command.
- Example of character device files is printers, modems and consoles.

FIFO file

- FIFO file is special pipe device file which provides temporary buffers for two or more processes to communicate by writing data to and reading data from the buffer.
- A FIFO is created by the *mknod* system call and may be opened and accessed by any process that knows the name and has the permissions.
- The buffer associated with a FIFO file is allocated when the first process opens the FIFO file for read or write. The buffer is discarded when all processes which are connected to the FIFO close their reference to the FIFO file.
- FIFO file may be removed like any regular file. FIFO files can be removed in UNIX via the *rm* command.

9.0.2 File System Structure

- Unix file system is characterized by
  1. A hierarchical structure.
  2. Consistent treatment of file data.
  3. The ability to create and delete files.
  4. Dynamic growth of files.
  5. The protection of file data.
  6. The treatment of peripheral devices as files.
- The UNIX file system is a methodology for logically organizing and storing large quantities of data such that the system is easy to manage. A file can be informally

- A relative path name specifies the path relative to another, usually the current working directory that you are at. Two special directory entries should be introduced now:

```
.    The current directory
..   The parent of the current directory
```

- The path names "/etc/passwd", "/bin/who" and "/usr/src/cmd/who.C" designate file in the tree.
- Programs in the Unix system have no knowledge of the internal format in which the Kernel stores file data, treating the data as an unformatted stream of bytes.
- Permission to access a file is controlled by access permission associated with the file. Access permissions can be set independently to control read, write and execute permission for three classes of users : The file owner, a file group and everyone else.

9.6.3 Directories

- Files on a UNIX system are organized into groups called directories; directories are containers that can hold files, and other directories.
- Each line in a directory file contains only the name of the item, and information on the location of the item.
  An ordinary file cannot contain another file, or directory.
- Directories are arranged into a hierarchy, which is a tree of a file system. A directory that is inside directory is referred to as a subdirectory.

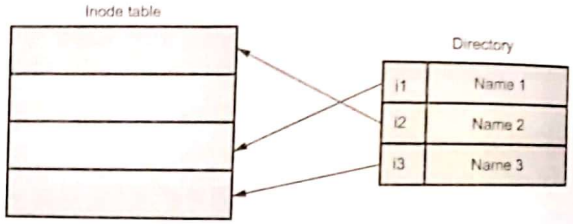


Fig. 9.6.3 Unix directories and inodes

- Fig. 9.6.3 shows the UNIX directories and inode.
- A directory is a file whose data is a sequence of entries, each consisting of an inode number and the name of a file contained in the directory.
- Each directory entry contains a name for the associated file or subdirectory plus an integer called the i-number. When the file or directory is accessed, its i-number is used as an index into the inode table.
- Unix system V restricts components names to a maximum of 14 characters, with a 2 byte entry for the inode number, the size of a directory entry is 16 bytes.

- Following table shows the directory layout for "etc". Every directory contains the file names dot and dot-dot ("." and "..") whose inode numbers are those of the directory and its parent directory, respectively.
- The inode number of "." in "/etc" is located at offset 0 in the file and its value is 83.

Byte offset in directory	Inode number (2 bytes)	File names
0	83	.
16	2	..
32	1798	init
48	1276	fsck
64	85	clri
80	1268	motd
96	1799	mount
112	88	mknod
128	2114	passwd
144	1717	umount
160	1851	checklist
176	92	fsdblb
192	84	config
208	1432	getty
224	0	crash
240	95	mkfs
256	188	inittab

- Kernel stores data for a directory just as it stores data for an ordinary file, using the inode structure and levels of direct and indirect blocks.

9.6.3.1 Special Directories

- There are four special directories that play an important role in the directory structure : Root directory, home directory, working directory, and parent directory.

1. Root directory : The root directory is the highest level in the hierarchy. It is the root of the whole file structure; therefore it does not have a parent directory. In the UNIX environment, the root directory is designated by a slash "/", and always

has several levels of subdirectories. The root directory belongs to the system administrator and can only be changed by the system administrator.

2. **Home directory** : One such subdirectory is your own personal directory, assigned to you with your account and denoted by your login id. This is called your home directory, and you can add to it new files, and new subdirectories which in turn can have more files and subdirectories. When your log in you are immediately placed in your home directory.

3. **Working directory** : The working or current directory is the one that we are in at any point of a session. When we log in, the working directory is your home directory.

4. **Parent directory** : The parent directory is immediately above the working directory. When we are in our home directory, its parent directory is one of the system directories.

- Some of the UNIX system directories are :

Directory	Contents
bin (binaries)	Contains the executable versions of the UNIX utilities.
etc	Contains system files and data files used by the system administrator or superuser.
lib	Contains the archive for utilities and system calls.
usr	The usr directory contains the files for each user.
dev	Contains all device drivers stored as separate files.
lost+found	Used for storage of recovered files after a system failure.

4 Files and Devices

Even hardware devices are represented or mapped by files in UNIX. For example, as root, you mount a CD-ROM drive as a file,

```
$ mount -t iso9660 /dev/hdc /mnt/cd_rom
$ cd /mnt/cd_rom
```

n UNIX, each and every hardware device treated as a file. A device file allows to ccesses hardware devices so that end users do not need to get technical details out hardware. In short, a device file (also called as a special file) is an interface a device driver that appears in a file system as if it were an ordinary file. This ws software to interact with the device driver using standard input/output em calls, which simplifies many tasks.

- There are two types of device files based upon how data written to them and read from them is processed by the operating system and hardware :
  1. Character devices 2. Block devices
- **Character Device** : Character device files represent a physical device that transmits data in a character-based manner. Examples of character devices : Virtual terminals, terminals and serial modems etc.
- **Block Device** : Block device files represent a physical device that transmits data a block at a time. Examples of block devices : Hard disk, DVD/CD ROM, and memory regions etc.
- Device file allows transparent communication between user space applications and computer hardware. An application can perform read and write on device files like on regular files and the OS will automatically invoke appropriate device driver to perform actual data transfer.
- All device files are stored in */dev* directory. Use *cd* and *ls* command to browse the directory :

```
$ cd /dev/
$ ls -l
```

9.6.5 File Descriptor

- Internal representation of a file is given by an inode which contains a description of the disk layout of the file data and other information such as the file owner, access permissions and access times.
- Every file has one inode, but it may have several names, all of which map into the inode. Each name is called a link.
- If a process calls,
  Open ("/fs2 / mjb / rje / source file", 1);
  the Kernel retrives the inode for
  "/fs2/ mjb / rje / source file ".
- When a process creates a new file, the Kernel assigns it an unused inode. Inode are stored in the file system.
- Kernel contains two other data structures, the file table and the user file descriptor table. The file table is a global Kernel structure, but the user file descriptor table is allocated per process.
- File descriptor is used by the Kernel to index a table of open files for the current process.
- Fig. 9.6.4 shows table and their relationship.

- The Kernel returns a file descriptor for the open and create system calls, which is an index into the user file descriptor table.
- File descriptor table is maintained separately for every process that runs in the system as a component of a separate structure called the U area.

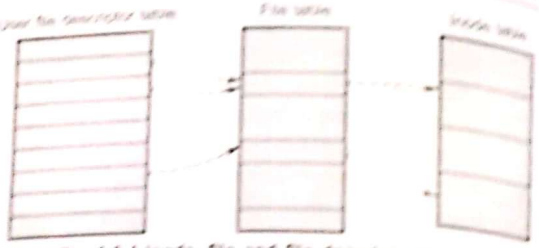


Fig. 9.6.4 Inode, file and file descriptor table

- U area contains all attributes of the currently running process.
- Every entry in the file descriptor table points to a file table.
- File table contains all data that are relevant to an opened file.
- The file subsystem manages files, allocating file space, administering free space, controlling access to files and retrieving data for users. Processes interact with the file subsystem via a specific set of system calls.
- The buffering mechanism interacts with block I/O device drivers to initiate data transfer to and from the Kernel. Device drivers are Kernel modules that control the operation of peripheral devices.
- The process control subsystem is responsible for process synchronization, interprocess communication, memory management and process scheduling. The file subsystem and the process control subsystem interact when loading a file into memory for execution.
- The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the Kernel moves them between main memory and secondary memory so that all processes get a fair chance to execute.
- The scheduler module allocates the CPU to processes. It schedules them to run. The hardware control is responsible for handling interrupts and for communicating with the machine.
- File system has the following structure which is shown in Fig. 9.6.5.
- The boot block occupies the beginning of a file system, typically the first sector and may contain the bootstrap code that is read into the machine to boot or initialize the OS.
- The super block describes the state of a file system.
- The inode list is a list of inode that follows the super block in the file system.

- The data block start at the end of the inode list and contain file data and administrative data.

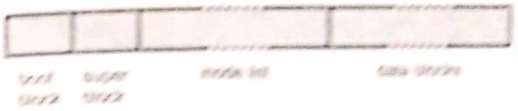


Fig. 9.6.5 File system layout

- Inode contains information about each file in the data block. The file information is stored in records known as **inodes**. There is one inode for each file on the disk. They contain information about the file, its file type, permission and address.

Advantages of inode :

1. Data in small files can be accessed directly from the inode.
2. Larger files can be accessed efficiently, because an indirect block points to many data blocks.
3. Disk can be filled completely, with little wasted space.

Disadvantages of inode :

1. Because inode information is kept separately from data, access of data often requires a long seek when file is initially accessed.
2. Inodes of files in a common directory not kept together, leading to low performance when searching directories.
3. Original file system only used 512-byte blocks, an inefficient transfer size.
4. Free list quickly becomes scrambled increasing overhead of finding free blocks.
5. Original file systems used as little as 2 % of the available disk bandwidth.

9.6.3 Inode

- An inode is a control structure that contains the key information needed by the operating system for a particular file. Inodes exist in a static form on disk, and the Kernel reads them into an in-core inode to manipulate them.
- Several file names may be associated with a single inode, but an active inode is associated with exactly one file and each file is controlled by exactly one inode.
- Disk inode consist of the following fields :
  1. **File owner identifier** : Owner are of two types i.e. individual owner and group owner. The superuser has access rights to all files in the system.
  2. **File type** : File types are regular, directory, character or block special or pipes.

3. **File access permissions :** The system protects files according to three classes :

   a. Owner b. Group owner of the file c. Other user

   Each class has access right to read, write and execute the file, which can be set individually.

4. **File access times :** It give the time of last file modified. When it was last accessed and when the inode was last modified.

5. **Number of links to the file :** Representating the number of names the file has in the directory hierarchy.

6. Table of contents for the disk addresses of data in a file.

7. **File size :** Data in a file is addressable by the number of bytes from the beginning of the file, starting from byte offset 0, and the file size is 1 greater than the highest byte offset of data in the file.

- The inode does not specify the path name that access that file. Fig. 9.6.6 shows the disk inode of a sample file.

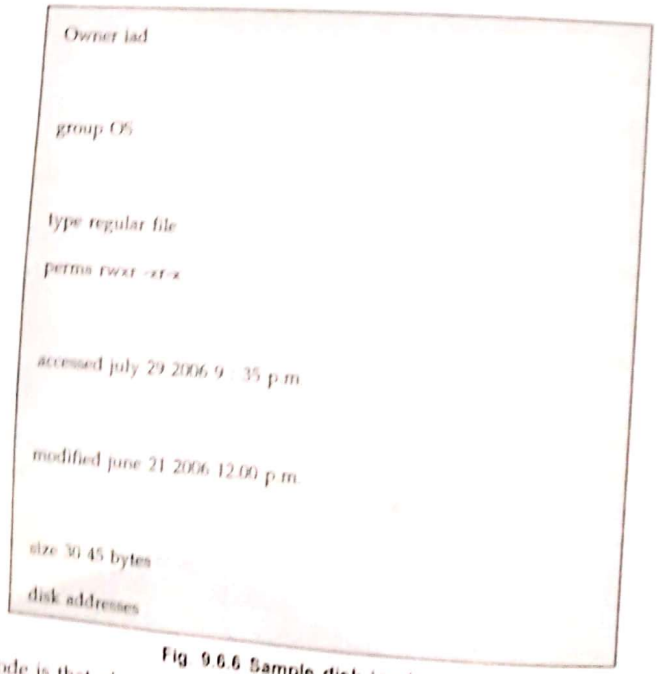


Fig. 9.6.6 Sample disk inode

- This inode is that of a regular file owned by iad which contains 3045 bytes.

- The attributes of files as well as its permissions and other control information are stored in the inode. Following table list the file attributes stored in the inode of a typical Unix implementation.

Parameter	Description	
File mode	16	bit flag that stored access and execution permissions associated with the file
	12 - 14	File type
	9 - 11	Execution flags
	8	Owner read permission
	7	Owner write permission
	6	Owner execute permission
	5	roup read permission
	4	Group write permission
	3	Group execute permission
	2	Other read permission
	1	Other write permission
	0	Other execute permission
Link count	Number of directory references to this node	
Owner ID	Individual owner of file	
Group ID	Group owner associated with this file	
File size	Number of bytes in file	
File addresses	39 bytes of address information	
Last accessed	Time of last file access	
Last modified	Time of last file modification	
Inode modified	Time of last inode modification	

- Changing the contents of a file automatically implies a change to the inode but changing the inode does not imply that the contents of the file change.
- The in-core copy of the inode contains the following fields in addition to the fields of the disk inode.

- At any instant, the third Kernel register triple refers to the U area of the currently running process, but the Kernel can refer to the U area of another process by overwriting the entries for the U area page table address with a new address.
- The entries for register triples 1 and 2 do not change for the Kernel, because all processes share Kernel text and data.

6.6.3 File Allocation

- File allocation is done on a block basis and allocation is dynamic. UNIX uses a multilevel indexing mechanism for file allocation on disk. Addresses of first 10 data blocks + 3 index blocks (first, second, and third level of indexing)
- In UNIX System V the length of a block is 1 kbyte and each block can hold a total of 256 block addresses.
- According to above parameters, maximum size for a file is slightly over 16 Gbytes.
- A method is needed to keep track of which blocks are allocated to a certain file.
- **Contiguous allocation** : All blocks in a file are placed in a sequence on the disk. Give high bandwidth but difficult to locate free space.
- **Linked allocation** : Each block may be located anywhere on the disk and every block contains the address to the next block. Easy to locate free blocks, but very inefficient for sequential access.
- **File Allocation Table (FAT)** : The disk keeps an area reserved for the File Allocation Table (FAT). The table has an entry for every block in the file system. Free blocks are marked with 0. For blocks belonging to a file, every entry in the FAT gives the entry for the next block in the file. The last block in every file contains an end marker.
- **Indexed allocation** : The block pointers are kept in special index blocks. A 512 byte block contains pointers to 128 data blocks. Several variants exist.

University Questions

1. Explain files and directory management in UNIX operating system. **GTU : Winter-14, Marks 7**
2. Short note on i-Node. **GTU : Summer-14, Marks 4**
3. What is "inode" ? Explain file and directory management of Unix operating system. **GTU : Summer-14, Marks 7**

vi Editor

The vi editor (short for visual editor) is a screen editor which is available on almost all UNIX systems. It has no menus, but instead uses combinations of commands in order to accomplish commands.

There are three basic modes of vi:

Command mode This is the default mode when you enter vi. In command mode, your letters or short sequences of letters, that you type will be interpreted as commands, without explicitly pressing Enter. If you press Esc when you're in command mode your terminal will beep at you. This is a very good way to tell when you're in command mode.

Insert mode In insert mode whatever you type is inserted in the file at the cursor position. Type a lowercase letter a, for append, to enter insert mode from command mode, press Esc to end insert mode and return to command mode.

Line mode Use line mode to enter line oriented commands. To enter line mode from command mode, type a colon (:). Your cursor moves to the bottom of the screen, for a colon prompt. Type a line mode command, and then press Enter. Any valid command from the UNIX line editor ex will work and a few are given to you above. These commands are indicated in this handout by a colon in front of the command. Each time you see a line mode command, you must type a colon to enter line mode, then type the command to the colon prompt at the bottom of the screen, then press Enter when you finish typing the command.

Entering and Saving Files

vi filename	Start editing filename; create it if necessary
:wq	Write the file to disk and quit
:q!	Quit without saving any changes
:w myfile	Write all lines from the entire current file into the file myfile, overwriting any existing myfile
:5,9w myfile	Write the lines from 5 to 9, inclusive, into the file myfile, overwriting any existing myfile

Moving the Cursor

Type :	To move to :
h	One space to the left (also try left arrow)
j	One line down (also try down arrow)
k	One line up (also try up arrow)
l	One space to the right (also try right arrow)
$	End of current line
^	Beginning of current line
Enter	Beginning first word on the next line
G	End of file
:0	Line # one. 0 is above the beginning of the file
w	Beginning of next word. 5w moves to the beginning of the 5th word to the right
e	End of next word
b	Beginning of previous word
Ctrl-b	One page up
Ctrl-f	One page down
%	The matching ( ) { } [ ], or < >

Searching for Text

Type :	To :
/string	Search down for string
?string	Search up for string
n	Repeat last search from present position

Inserting Text

Type :	To :
a	Append starting right of cursor
A	Append at the end of the current line
I	Insert starting left of cursor

I	Insert at beginning of the current line.
o	Open line below cursor, then enter insert mode.
O	Open line above cursor, then enter insert mode.
:r newfile	Add the contents of the file *newfile* starting below the current line.

Deleting Text

Type :	To :
x	Delete single character; 5x deletes 5 characters.
dw	Delete word; 5dw deletes 5 words.
dd	Delete line; 5dd deletes ... well you get the idea!
cw	Delete word, leaves you in insert mode (i.e. change word).
cc	Change line – delete line and start insert mode.
s	Change character – delete character and start insert mode.
D	Delete from cursor to end of line.
C	Change from cursor to end of line – delete and start insert mode.
u	Undo last change.
U	Undo all changes to current line.
J	Join current line with line that follows (press **Enter** in insert mode to split line).

9.8 Shell Meta Characters

- **Metacharacters** are characters that specially processed by a shell. When you enter a command, shell do the following :
  a. Scans for metacharacters,
  b. Specially processes them,
  c. Then executes command.
- (Shell) metacharacters are characters with special meaning and cannot be used in shell commands as literal characters without specifying

- For example, they cannot be used in filenames.

Character	Purpose
$	To end the line and dereference a shell variable.
&	To provide the background execution of the command.
#	To start a comment.
'	To quote multiple characters.
()	To execute a command list in a subshell.
*	To match a zero or more characters.
[ ]	To insert wildcards.
^	To begin a line and negation symbol.
`	To substitute a command.
{ }	To execute a command list in the current shell.
|	To create a pipe between commands.
;	To separate commands in sequential execution.
<	To redirect input for a command.
>	To redirect output for a command.
?	To substitute a wildcard for a single character.
/	To be used as root directory and directory separator.
\	To allow continuation of statement on the following line.
!	To start an event specification in the history list.
%	Tcsh prompt, or the starting character for job number.
~	To name home directory.

Special Keys

- UNIX has a number of special keys that perform particular functions. Some important ones are the keys necessary to backspace over a character when entering a command, to delete the whole line being entered and to interrupt execution.
- These are user-specifiable and have different defaults based on the shell, the version of UNIX and the login files.

Name	Control char	Function
erase	DEL or backspace	Erase character. Backspace and erase one character.
werase	<Ctrl-w>	Delete the rightmost word typed in.
kill	<Ctrl-u>	Kill (erase) the line typed in so far.
intr	<Ctrl-c>	Interrupt the program currently running.
rprnt	<Ctrl-r>	Reprint the line typed in so far.
flush	<Ctrl-o>	Stops terminal output until you press a key.
susp	<Ctrl-z>	Suspend the program currently running and put it in the background.
stop	<Ctrl-s>	Stop the display. To resume, press the start key.
start	<Ctrl-q>	Start the display after stop.
eof	<Ctrl-d>	Send the program an end-of-file character.

9.9 Environment Variable

- A special class of variables is **environment** variables. It has special meaning for the shell and defines the working environment.
- It can be modified at the command line level or through the startup files.
- Environment variable is used by some programs to customize them to the user. A number of utilities may need to use the editor; the environment variable EDITOR allows you to customize the utilities so that you always work with your favorite editor.
- An example is the *environment* variable DISPLAY which is used by most of the X programs.
- You can see the current setting of all environment variables by using the command *printenv* or *env*.
- Environment variables are managed by the shell.

Environment Variables

- The Bourne shell predefines a set of global variables known as environment variables. They define default values for the current account's home directory, the strings to use as the primary and secondary prompts, a list of directories to search through when attempting to find an executable, the name of the account and the current shell to name a few.
- The values for these variables are generally set at login when the shell reads the .login script. This script resides within the account's home directory.

- Environment variables are by no means set in stone. They may be change session to session, but generally, they remain constant and keep a user's a configured for the system. A short listing of some common environment va is shown in Table 9.9.1.

Variable	Meaning
HOME	The absolute path to the user's home directory.
IFS	The internal field separator characters. Usually contains space, tab and newline
PATH	A colon separated list of directories to search through when trying to execut command.
PS1 PS2	The strings to use as the primary, PS1 (S) and secondary prompts, PS2 (>).
PWD	The current working directory.
SHELL	The absolute path to the executable that is being run as the current shell session
USER	The name of the current account.

Table 9.9.1 Common environment variables

- Changing an environment variable is as easy as assigning it a new value. As example, the current directory could be appended to PATH's value. Doing would allow a user to execute programs in the current directory witho specifying a fully qualified path to the executable.

```
$ echo ${PATH}
/bin:/usr/bin:/usr/ucb/bin
$ PATH=${PATH}:${PWD}
$ echo ${PATH}
/bin:/usr/bin:/usr/ucb/bin:/home/rupali
```

9.9.1 Shell Variables

- A shell variable is a name with a value assigned to it. Most variable values consis of text and most variable names are all upper case. An environment variable i defined as part of initializing the operating system or a shell.
- Environment variables can also be defined after the shell has been created Programs use environment variables to find information about the curren environment, such as the user's home directory.
- Some programs require that specific environment variables be defined before the program is started. If these variables have no value, the program will terminate with an error message.



- The **env** command can be used to define an environment variable setting for the program being launched. The **set** command is used to display a list of current environment variables. The value of a single environment variable can be displayed using the echo command.
- The name of the environment variable must be preceded with a $. The export command is used to make the value of a new environment variable available to the environment.
- The PATH environment variable contains a list of directories that UNIX searches each time a command is executed. Any executable program must either be placed in a directory on the PATH or a path to the file must be provided.

9.9.2 Difference between Shell Variables and Environment Variables

- Shell variables are local to a particular instance of the shell, such as shell scripts.
- Environment variables are inherited by processes started from the shell, including other shells.
- In Unix, every process (shell or otherwise) passes the environment variables to its children. The passing of variables is one way only, variables cannot be passed back to parent shell; neither a change in their value reflected back in parent shell.
- Environment variables are printed by using the command *env* and shell variables are printed by using the command set.

9.10 Control Structure

9.10.1 If Statements

- The first branching method uses the if-then-elif-else-fi construct. Read elif as "else if;" it makes more sense. An if statement reads similarly to classical logic. If something is true, then do the following or if another statement is true, then do this instead; otherwise, just do this.
- The if-then-elif-else-fi block has the following syntax :

```
if [ condition ] then
  command1
     :
  commandn
elif [ condition ] then
        command1
          :
        commandn
          :
```



```
          :
        elif [ condition ] then
        command1
          :
        commandn
    else
        command1
          :
        commandn
        fi
```

- An if or elif is followed by a test. Whenever the test evaluates to true, the shell executes the commands that follow and then continues with the rest of the script after the block's end point, the fi.
- The else statement is a default action. Given that neither of the if or elifs pass, the shell performs the commands within the else block and the script again continues execution after the fi.
- Usage of elifs and elses in an if statement are optional. As a bare minimum, an if block must utilize an if-then-fi construct.
- Here is a popular example :

```
if [ "${USER}" != "root" ]; then
echo "Sorry, ${USER}, you must be the superuser to run this program."
exit 1
fi
echo "Executing program."
```

- This simple little subroutine checks if the current operator is the root user. If not, then it prints a message indicating that the script may only be executed by those with root privileges and promptly terminates.
- In the event that the operator is logged in as root, the script moves to the fi and then prints a friendly message indicating that it is continuing execution.

9.10.2 Case Statements

- The second form of branching available is the case statement. A case differs from an if block in that it branches based upon the value of one variable.
- An if does not have such restrictions because it uses the test function and a test can be any string of logical expressions as has been shown.
- The best way to illustrate this difference is to examine the syntax :

```
case (variable) in
pattern1 )
      command1
```

```
        :
        command_n
        ;;
        :
pattern_n )
        command_1
        :
        command_n
        ;;
esac
```

- The case block takes as its argument a variable. To denote the variable, it must be surrounded by parentheses. The case compares the variable's value against each pattern.
- The pattern may be any legal regular expression. If the variable's value matches the pattern, then the shell executes the command block immediately following the pattern. The command block terminates with a pair of double semi-colons. As soon as it reaches them, the shell continues past the end of the case block as denoted by the esac.
- A good use for a case statement is the evaluation of command line arguments :

```
option="${1}"
case ${option} in
  -f) FILE="${2}"
      ...
  -d) DIR="${2}"
      ...
  *) echo " basename ${0} : usage: [ -f filename ] | [ -d directory ]"
     exit 1
     ;;
esac
```

- In the above example, the script expects two arguments. The first is one of two option indicators, -f or -d, that tell the program whether the second argument is a file or a directory.
- The case statement switches upon the first argument's value stored in option. If option equals -f, then it assigns the option's argument, the second positional parameter, to FILE. Then the script continues after the esac.
- On the other hand, if option is -d, the shell assigns the second argument's value to DIR and continues execution.
- Finally, this case statement has a default action. In the event the user does not use either the -f or -d options, it detects the error with the wild card, *, which matches any pattern. The script prints a message instructing the user how to use it properly and then exits.

9.10.3 For Loops

- In the Bourne shell, the first type to consider is the for loop. A for loop processes multiple commands on a list of items. This is a bit non-traditional in that languages such as C and FORTRAN employ for loops as a sort of counting mechanism; the loop continues until the variable reaches a target value.
- But here, the for loop uses the intermediate variable to execute actions on a set of items such as files or the results of a command. In this way, the shell's for loop is a more flexible and useful than in most other languages.
- The syntax of a for loop is :

```
for variable in list
do
        command_1
        :
        command_n
done
```

- The for loop's *variable* may be any legal shell variable. It need not be declared prior to the loop and in fact, good shell programming practice mandates that the scripter should choose a unique variable that is not and will not be used elsewhere.
- As the for loop executes the command block, the value of variable changes into the next item found in list. Items in list must be separated by white space. The list may be generated by many different methods including File expansion, variable substitution, positional parameters, command substitution or by simply providing a list of strings immediately.

Example :

Here is a simple example that uses for loop to span through the given list of numbers :

```
#\/bin/sh
for var in 0 1 2 3 4 5 6 7 8 9
do
        echo $var
done
```

This will produce following result :

```
0
1
2
3
4
5
```

```
6
7
8
9
```

9.10.4 While Loops

- A while loop is different from a for loop in that is uses a test condition to determine whether or not to execute its command block. As long as the condition returns true when tested, the loop executes the commands.
- If it returns false, the script skips the loop and proceeds beyond its termination point. Consequently, the command set could conceiveably never run. This is further illustrated by the loop's syntax :

```
while condition
do
        command1
        :
        commandn
done
```

- Because the loop starts with the test, if the condition fails, then the program continues at whatever action immediately follows the loop's done statement. If the condition passes, then the script executes the enclosed command block.
- After performing the last command in the block, the loop tests the condition again and determines whether to proceed beyond the loop or fall through it once more.

Example :

Here is a simple example that uses while loop to span through the given list of numbers :

```
i=1
while [ ${i} -le 5 ]
do
echo ${i}
i=`expr ${i} + 1`
done
```

This will produce following result :

```
1
2
3
4
5
```

- The above example demonstrates a while loop. Given the variable i whose initial value is one, the loop starts by checking if i is less than or equal to five.

- Since it is, the loop prints i's value, increments it by one as shown with the expr command and then tests its value once more. This continues until i reaches six. At that point, the loop terminates and the shell returns control to the user.

9.10.5 Until Loops

- Until loops are the other while loops. They work similarly in nature to while loops except for one major difference.
- Until loop executes its command block only if the test condition is false. In this sense, an until loop can be viewed as the logical negation of a while loop.
- Syntax

```
until condition
do
        command1
        :
        commandn
done
```

- Once again, the command block may never be executed as shown in its syntax. If the condition evaluates to true, then the script proceeds beyond the loop.

9.11 Command Substitution

- The standard output from a command can be substituted in a similar way to parameters. The command **pwd** prints on its standard output the name of the current directory.
- For example, if the current directory is */usr/fred/bin* then the command

  d=`pwd`

  is equivalent to

  d=/usr/fred/bin
- The entire string between grave accents (`...`) is taken as the command to be executed and is replaced with the output from the command. The command is written using the usual quoting conventions except that a ` must be escaped using a \.
- **For example :**

  ls `echo "$1"`

  is equivalent to

  ls $1

- Command substitution occurs in all contexts where parameter substitution occurs (including here documents) and the treatment of the resulting text is the same in both cases.
- This mechanism allows string processing commands to be used within shell procedures. An example of such a command is base-name which removes a specified suffix from a string.
- **For example :**

```
basename main.c .c
```

will print the string main. Its use is illustrated by the following fragment from a cc command.

```
case $A in
...
*.c)    B=`basename $A .c`
...
esac
```

that sets B to the part of $A with the suffix .c stripped.

9.12 Debugging Shell Scripts

- Two methods of tracing shell script execution are useful for debugging. The script may be called with a verbose flag **(-v)** or execution trace flag **(-x)**.
- The **-v** flag causes each line of the shell script to be echoed as it is read but after all substitutions are performed.
- The **-x** flag causes each line to be echoed just before it is executed.
- **For example :**

```
sh -v do_it
sh -x do_it
```

To turn on both flags, use **-vx.**

9.12.1 Running Programs on UNIX

- To run programs written in other languages, see the appropriate man pages.

C Programs

- The C compiler on UNIX is cc, which is called with the command

```
cc [options] file ...
```

- CC gives special interpretations to filename suffixes. It treats filenames ending in **``.c"** as C source programs. When it compiles these programs, it writes object programs to files that have the same name but with the **``.c"** suffix replaced by **``.o"**.
- Unless options on the **cc** command prevent it, the **``.o"** file is loaded into an executable file named **a.out** and the **``.o"** file is then removed.

- To execute the resulting program, simply type ***a.out*** as a command.
- Here are some of the most useful options to the **cc** command :

  -c = Suppresses loading and compiles only : it merely creates ``.o" files.

  -g = Generates a special symbol table that can be used by debugging pro

  -o *outfile* = Names the resulting executable file outfile instead of a.out, thu preventing you from overwriting any existing file named a.ou

9.13 Shell Script

GTU : Summ

Program No. 1

Write a shell script for employee database

```
echo "Enter Database Name :"
read fname
while true
do
echo "************************************"
echo "*** WELCOME TO EMPLOYEE DATABASE ***"
echo "************************************"
echo "***     1> Add Records          ***"
echo "***     2> Display Records      ***"
echo "***     3> Search String        ***"
echo "***     4> Delete Record        ***"
echo "***     5> View Source Code     ***"
echo "***     6> Exit                 ***"
echo "************************************"
echo "Enter Your Choice :"
echo "************************************"
read choice
case $choice in

    1)    #if choice is 1
          echo "*** ADD RECORD ***"
          flag=0
          while [ $flag -eq 0 ]
          do
                echo "Employee Id :          "
                read eid
                if test -e $fname
                then      #primary key check
                    if grep $eid $fname >> /dev/null
                    then
                        echo "ERROR PK violation"
                        read tp
```

```
    ;;
  esac
done
```

Program No. 2

To count the number of lines and words

```
if [ $# -eq 0 ]
then
  echo "You have not entered the filename as arguments!!"
else
if [ ! -e $1 ]
then
  echo "File does not exist"
elif [ ! -f $1 ]
then
  echo "File is not ordinary file"
else
echo "The number of lines are " ; wc -l < $1
echo "the number of words are " ; wc -w < $1
fi
fi
```

Program No. 3

Program to find prime numbers

```
echo " Program to find prime numbers"
echo ""
echo "Enter the no of terms : "
read n
echo ""
echo "the list of the prime numbers is "
echo "_____"
echo ""
a=`expr $n - 1`
j=2
r=0
while [ $j -le $a ]
do
  i=2
  while [ $i -<p> $j ]
  do
    b=`expr $j % $i`
    if [ $b -eq 0 ]
    then
      r=1
    fi
    i=`expr $i + 1`
```



```
  done
  if [ $r -eq 0 ]
  then
    echo "        "$i
  fi
  i=`expr $i + 1`
  r=0
done
echo "_____"
echo""
```

Program No. 4

Program to display the type of file.

```
echo "Enter Your File Name :"
read f_name

if [ -z $f_name ]
then
      echo "file name is null "
elif [ ! -e $f_name ]
then
      echo "file is not exist"
elif [ -f $f_name ]
then
      echo " Regular File "
      if [ -r $f_name ]
      then
            echo "File is Readable"
      fi
      if [ -w $f_name ]
      then
            echo "File is Writable"
      fi
      if [ -x $f_name ]
      then
            echo "File is Executable"
      fi
elif [ -d $f_name ]
then
      echo "File is Directory"
fi
```

University Question

1. Write a shell script to find greater number out of 3 numbers. GTU : Summer-13, Marks 4

Notes

10 LINUX Operating System

Syllabus

Development of Linux, System administration case study : Linux.

Contents

10.1 Introduction to Linux

10.2 System Administration ..... **Summer-13,** ..... Marks 4

Introduction to Linux

Linux is developed in 1991 by Linus Torvalds. It is used in most of the computers, ranging from super computers to embedded system. Linux is a free operating system based on UNIX standards.

Linux is multi user, multi tasking, time sharing and monolithic kernel etc.

Linux is Free Open Source Software. Free means liberty and not related to price or cost and Open means source code is available and anybody can contribute to the development.

The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.

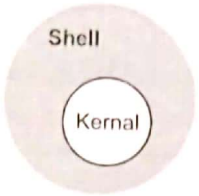


Fig. 10.1.1 Shell and Kernel

Kernel is a main program of UNIX system. It controls hardwares, CPU, memory, hard disk, network card etc.

Shell is an interface between user and kernel. Shell interprets your input as commands and passes them to kernel.

Principles

Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools.

Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model.

Main design goals are speed, efficiency and standardization.

Linux is designed to be compliant with the relevant POSIX documents; at least one Linux distributions have achieved official POSIX certification.

The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behaviour.

Components of Linux System

Linux operating system consists of three components :

1. Kernel 2. System library 3. System utility

- Fig 10.1.2 shows Linux operating system components.

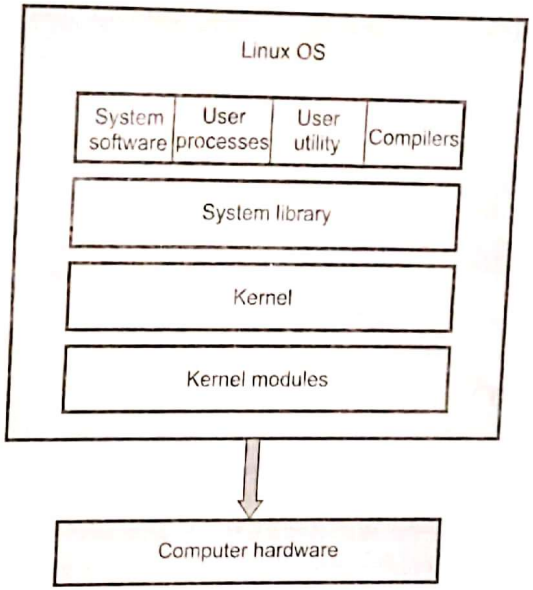


Fig. 10.1.2 Linux OS components

1. **Kernel :** Kernel is the heart of the Linux operating system. It is responsible for all major activities of Linux OS. It provides important abstraction to hide low level hardware information to user or application program. Kernel is combination of various modules and it interacts directly with the underlying hardware. The Linux kernel consists of several important parts : Process management, memory management, hardware device drivers, file system drivers, network management etc.
2. **System Library :** System libraries are special functions or programs. Libraries are used for accessing kernel's features. These libraries implements most of the functionalities of the operating system and do not requires kernel module's code access rights.
3. **System Utility :** System utility programs are responsible to do specialized, individual level tasks.

- Privileged mode is called kernel mode in Linux system. All the kernel code executes in the privileged mode with full access to the hardware resource of the computer. In Linux operating system, no user mode code is built into the kernel.

10.2 System Administration

GTU : Summer-13

- System administrator is a person who is responsible for the configuration, installing and maintaining reliable operation of computer systems. To meet these needs, a system administrator may acquire, install, or upgrade computer components and software; provide routine automation; maintain security policies; troubleshoot etc.
- System admin is a person who has a root access means root user or super-user.
- Most major administrative tasks require that the system administrator operates in the superuser mode with root privileges. These tasks include starting up and

shutting down a system, opening an account for a new user and giving him a proper working set-up.

- Administration tasks also involve installation of new software, distributing user disk space, taking regular back-ups, keeping system logs, ensuring secure operations and providing network services and web access.

10.2.1 Requirements for a Linux System Administrator

1. System admin manages the large Linux server environment, with an emphasis on performance monitoring, tuning and management.
2. It maintains database physical design, documentation and administration.
3. It also provides network troubleshooting and monitoring the mission-critical systems.
4. Admin is responsible for guidance and direction of technology solutions for the organization.
5. Linux server infrastructure to maintain stability as well as maximize efficiencies in the computing environment.
6. Install, configure and troubleshoot all hardware, peripherals and equipment necessary to meet integrated systems objectives.
7. Manage all aspects of the integrity of the environment, including security, monitoring, change control and software management.
8. Administer infrastructure services like DNS, NIS, LDAP, FTP, SMTP and application and database servers, with an emphasis on automation and monitoring.
9. Creating and managing user accounts.
10. Reconfiguring the kernel whenever required : Sometimes when new patches are installed or a new release of the OS is received, then it is imperative to compile the kernel. Linux users often need to do this as new releases and extensions become available.

- Linux administrators can configure distributions by choosing from a vast number of components that do similar jobs. For example, with almost every UNIX distribution, send mail is the only choice of mail transfer agent.

10.2.2 Adding Users : Basic Steps

- The process of adding a new user consists of several steps required by the system, two steps that establish a useful environment for the new user and several extra steps for your own convenience as an administrator.

- **Required steps :**
  1. Have the new user sign your policy agreement.
  2. Edit the **passwd** and shadow files to define the user account.
  3. Add the user to the **/etc/group file**.
  4. Set an initial password.
  5. Create, chown and chmod the user's home directory.
  6. Configure roles and permissions.
- **For the user :**
  1. Copy default startup files to the user's home directory.
  2. Set the user's mail home and establish mail aliases.
- For creating new user, you must be root user. The useradd command creates new user account using the values specified on the command line and the defa values from the system. The new user account will be entered into the system fi (/etc/passwd) as needed, the home directory (/home/username) will be creat and initial files copied, depending on the command line options.

Editing the passwd and group file

- To edit the **passwd** and **shadow** files, **vipw** command is used.
- On Solaris and Red Hat Systems, **vipw** command automatically asks if you wou like to edit the shadow file after you have edited the **passwd** file. SUSE a Ubuntu system use **vipw -s** for this function.
- Both HP-UX and AIX recommend that you not edit the password file by ha with or without **vipw**, but rather use **useradd.**
- If the new user should be a member of more group than just the default gro specified in the **passwd** file, you must edit the **/etc/group** file and add the us login name to each of the additional groups.

Setting Password

- Set a password for the new user with :

```
$ sudo passwd newusername
```

- Some automated systems for adding new users do not require you to provide initial password. Instead, they force the user to set a password on the first logi

Creating home directory and installing startup files

- You can create the new user's home directory with a simple **mkdir** command.
- Startup files traditionally begin with a dot and end with the letters "rc" command). The initial dot causes **ls** to hide these uninteresting files directory listings unless the **-a** options is used.

- Following is the list of common startup files and their uses :

Command	Filename	Typical Uses
Sh	.profile	Sets search path, terminal type and environment
Bash	.bashrc	Sets the terminal type Sets biff and mesg switches
	.bash_profile	Sets up environment variables Sets command aliases Sets the search path Sets the umask value to control permissions Sets CDPATH for filename searches
csh/tcsh	.login .cshrc	Similar to .bashrc for csh Similar to .login for csh
vi/vim	.exrc/ .vimrc	Set vi/vim editor options
Emacs	.emacs	Sets emacs editor options and key bindibgs
mail/mail x	.mailrc	Define personal mail aliases Sets mail reader options
GNOME	.gconf .gconfpath	GNOME environment : user configuration via gconf Path for additional user configuration via gconf
KDE	.kde/	KDE environment : directory of confuguration files

- Linux, Solaris and HP-UX kept sample startup files in **/etc/skel and /etc** for all other system. Depending on the user's shell, **/etc** may contain system wide startup files that are processed before the user's own startup files. For example, **bash** and **sh** read **/etc/profile** before processing **~/.profile** and **~/.bash_profile.**

Setting permissions and ownerships

- Now that the home directory is set up, turn it over to the user and make sure that the permission on it are appropriate. The command :

```
$ sudo chown -R newuser:newgroup  ~newuser
```

should set ownerships properly. Note that you cannot use

```
$ sudo chown  newuser:newgroup  ~newuser/.*
```

to **chown** the dot files because **newuser** would then own not only his own files but also the parent directory ".." as well.

Setting a mail home

- It is convenient for each user to receive email on only one machine. This scheme is often implemented with an entry in the global aliases file /etc/mail/aliases or sendmail userDB on the central mail server.

Configuring roles and administrative privileges

- Role based access control allows system privileges to be tailored for individual users.
- RBAC is not a traditional part of the UNIX or Linux access control model.

Final Steps

- To verify that a new account has been properly configured, first log out then log in as the new user and execute the following commands :

```
$ pwd          /* To verify the home directory */
$ ls -la       /* To check owner/group of startup files */
```

- User will need to notify new users of their login names and initial passwords.
- Remind new users to change their passwords immediately. If admin user wish, you can enforce this by setting the password to expire within a short time.

10.2.3 Adding Users with useradd

- Syntax is as follows for **useradd** command :

```
useradd <username>
```

By default user account is locaked, you need to setup a new password :

```
passwd <username>
```

- For example add a new user called **preeti** and set password to **technical :**

```
# adduser preeti
# passwd technical
```

- List of common options :

1. **-c comment :** Add a comment for the user.
2. **-d home-directory :** Create a new home-directory.
3. **-e yyyy-mm-dd :** Date for the account to be disabled.
4. **-f days :** Number of days after the password expires until the account is disabled.
5. **-g group-name :** Primary group name or group number for the user's default group.
6. **-G group-list :** List of secondary additional (other than default) group names or group numbers, separated by commas, of which the user is a member (group name must exist).
7. **-m :** Create the home directory.
8. **-M :** Do not create the home directory.
9. **-s :** User's login shell (default /bin/bash).

- To read all command line option type following command :

```
$ man adduser
```

- Following is the list of commands and configuration files for user management

System	Cmds	Configuration Files	Comments
Ubuntu	Useradd	/etc/login.defs /etc/default/useradd	
	Adduser	/etc/adduser.conf	Friendlier Perl Version
SUSE	Useradd	/etc/login.defs /etc/default/useradd /etc/default/passwd /usr/sbin/useradd.local /usr/sbin/userdel.local /usr/sbin/userdel-pre.local /usr/sbin/userdel-post.local	For local customizations For local customizations For local customizations For local customizations
Red Hat	Useradd	/etc/login.defs /etc/default/useradd	
Solaris	Useradd	/etc/default/login,passwd /etc/security/policy.conf	
WFADE	Useradd Smn	/etc/default/useradd /etc/default/security	GUI tool, also called sam.
AIX	Useradd mkuser chuser rmuser SMIT	/etc/security/user /etc/security/login.cfg /etc/security/mkuser.defaults	Called by useradd Called by usermod Called by userdel GUI tool

Ubuntu

- Ubuntu provides two ways to add users : **adduser** and **useradd.**
- **adduser** is configured in /etc/adduser.conf, which includes options such as these :
  1. Rules for locating home directories : by group, by username etc.
  2. Permission setting for new home directories.
  3. UID and GID ranges for system users and general users.
  4. An option to create individual groups for each user.

5. Disk quotas.
6. Regex based matching of usernames and group names.

SUSE

- SUSE's **useradd** does not create a new user's home directory or copy in startup files by default. It also does not create user specific groups; the default GID in the **passwd** file is set by the variable GROUP in **/etc/default/useradd.**
- New users are also added to the groups specified by the GROUPS variable. By default, these are the group video and dialout, which allow access to the systems frame buffer and the dialup IP software **pppd.**
- To make up for these deficiencies, SUSE's **useradd** calls **/usr/sbin/useradd.local,** a bash script to which you can add whatever customizations you wish.
- The **/etc/login.defs** file on SUSE controls the following types of issues :
  1. Whether to allow logins if a user's home directory does not exist.
  2. Degree of tolerance for failed login attempts.
  3. Location of the "message of the day" and **ttytype** files.
  4. Restrictions on the use of **chsh** and **chfn.**
  5. Password aging.
  6. Ranges of system and user UIDs and GIDs.
  7. Rules for forming valid user and group names.
  8. User's **umasks.**
  9. Local scripts that piggyback on the **useradd** and **userdel** commands.

Red Hat

- The **useradd** program on Red Hat Enterprise Linux takes its configuration parameters from **/etc/login.defs,** where issues such as password aging controls, encryption algorithms, mail spool files and UID/GID ranges are addressed.
- The default parameters are set in the file **/etc/default/useradd.** The **-D** option displays the defaults that **useradd** will use for new login.
- The **passwd** file entry uses "x" as a password place holder and the shadow file uses "!!", a code that disables the login and requires a **sysadmin** to set a password for the new user. MD5 algorithm is default encryption algorithm.

Solaris

- Default parameters related to login and password are stored in **/etc/default/login** and **/etc/default/passwd.** The option **useradd -D** shows the default values for several parameters.

- The format of the **default/login** and **default/passwd** files is similar to that of Linux's **login.defs** file in that blank line and lines beginning with # are ignored. The syntax is

```
NAME=value
Rather than
NAME <white-space>value
```

- The **/etc/default/passwd** file controls the following :
  1. Minimum password length
  2. Password aging
  3. Required password complexity
  4. Checking for crackable passwords
- **/etc/default/login** controls issues such as followings :
  1. The time zone.
  2. Limits on the size of files a user can create.
  3. Whether root can log in only on the console.
  4. Whether a password is required for each user.
  5. Handling a failed login attempts.
  6. User's initial search path.
  7. User's default umask.
  8. Whether to log root and failed logins through syslog.
- The files **/etc/security/policy.conf** and **/etc/security/crypt.conf** determine the encryption algorithms that can be used for passwords.

HP-UX

- HP-UX does not create home directories by default. For creating home directory, option **-m** is used for creating home directory and populates them with startup files from **/etc/skel.**
- Configuration parameters for **useradd** are set in the files **/etc/default/useradd** and **/etc/default/security.** It uses Linux style format for *NAME<white-space>value* and the security using the solaris *NAME=value* style.
- In this system, syntax error messages are not generated.
- The **/etc/default/useradd** file controls options such as :
  1. Default group and shell.
  2. Root of the home directory tree.
  3. Account expiration.
  4. Whether to create home directories.
  5. Whether to allow duplicate UIDs.

- The file **/etc/default/security** contains additional configuration parameters, some o which relate to user management :
  1. Whether to allow logins with a missing home directory.
  2. Whether to allow null passwords.
  3. Minimum password length.
  4. Handling of failed login attempts.
  5. Handling of inactive accounts.
  6. Default umask for new users.

AIX

- Configuration files control numerous aspects of logins and passwords and are kept in the **/etc/security** directory. There are three relevant files : **login.cfg**, **user** and **mkuser.default.**
- The **login.cfg** and **user** uses the * as a comment character and **mkuser.default** has no comments. These files are organized in the following format :

```
label:
    attribute  = value
    next attribute = value
next-label:
    attribute = value
```

- In the **/etc/security/user** file, the labels are usernames.
- The **useradd** on AIX does not provide a **-D** options to show the default values for new users. It puts new users in a single group and does not create their home directories unless invoked with the **-m** flag.

Example :

- To create a new user "secomp1" using the system defaults on the Linux system, you could simply run following command :

```
$ sudo useradd secomp1
```

- This command would create the following entry in **/etc/passwd :**

```
secomp1:x:1005:10::/home/secomp1:/bin/sh
```

- The useradd disables the account by putting an "x" in the password field. User must assign a real password to make the account usable.

See User List :

- /etc/passwd is only used for local users only. To see list of all users, enter :

$ cat /etc/passwd

To search for a username called tom, enter :

$ grep tom /etc/passwd

- **/etc/passwd file permission**
- The permission on the /etc/passwd file should be read only to users (-rw-r--r--) and the owner must be root :

```
$ ls -l /etc/passwd
```

Output :

```
-rw-r--r-- 1 root root 2659 Sep 17 01:46 /etc/passwd
```

10.2.4 Removing User

- User's login account and files are removed from the system when that user leaves the organization or college. This procedure involves the removal of all references to the login name that were added by you or your **useradd** program.
- Following checklist is used after removing user by hand :
  1. Remove the user from any local user databases or phone lists.
  2. Remove the user from aliases file or add a forwarding address.
  3. Remove the users **crontab** file and any pending **at** jobs or print jobs.
  4. Kill any of the user's processes that are still running.
  5. Remove the user from the **passwd, shadow, group and gshadow** files.
  6. Remove the user's home directory.
  7. Remove the user's mail spool.
  8. Clean up entries on shared calendars, room reservation system etc.
  9. Delete or transfer ownership of any mailing lists run by the deleted user.
- Instead of deleting files of removed user, it should kept without deleting because it may be needed by some other users.
- Once you have removed the user, you may want to verify that the user's old UID no longer owns files on the system. To find the paths of orphaned files, you can use the **find** command with the **-nouser** argument.

```
$ sudo find filesystem -xdev -nouser
```

University Question

1. *List the advantages of LINUX/UNIX operating system over Windows.*

GTU : Summer-13, Marks 4

11 Windows Operating Syste

Syllabus

Windows operating system

Contents

11.1 *History of Windows OS*

11.2 *Design Principles*

11.3 *Windows Architecture*

11.4 *File System*

11.1 History of Windows OS

- In June 1981, Gates and Allen turn Gates' former Harvard classmate Steve Ballmer to help run the company. The next month, IBM approaches Microsoft about a project code-named "Chess." In response, Microsoft focuses on a new operating system, the software that manages, or runs, the computer hardware and also serves to bridge the gap between the computer hardware and programs, such as a word processor. It's the foundation on which computer programs can run. They name their new operating system "MS-DOS."
- When the IBM PC running MS-DOS ships in 1981, it introduces a whole new language to the general public. MS-DOS stands for Microsoft Disk Operating System.

Name and version number	Year	Remarks
DOS 1.0	Aug-1981	• Intel 8086 microprocessor • 8 kbytes RAM • Assembly language source code about 4000 lines.
DOS 2.0	1983	• Support for hard disk • User hierarchical directories
DOS 3.0	1984	• Intel 80286 microprocessor
DOS 3.1	1984	• Support for networking
DOS 3.3	1987	• Support PS/2 computers
Windows NT 3.0	1990	• Support for GUI • Support 16-bit • Implemented as a layer on top of MS-DOS.
Windows NT 3.1	1993	• Support for GUI • Support 32-bit OS • Support PS/2 computers
Windows 95	1995	• 32-bit version • Led to the development of Windows 98 and Windows Me
Windows 2000	2000	• Support distributed processing • Support plug-and-play

		• Power management
Windows XP	2001	• Used for home PC • Also used for business workstation • Goal was to replace the versions of Windows on MS-DOS with an OS based on NT
Windows Server 2003	2003	• Support 32-bit and 64-bit processor • Windows 2000 • Included services and functions to sup distributed processing • Active directory • Plug-and-play and power-management facilities

- Windows Vista shipped in 2007
- Windows Server released in 2008
- Windows 7 shipped in 2009, as well as Windows Server 2008 R2

- In the late 1980s Microsoft, under the leadership of Dave Cutler, star development of a new single-user operating system to replace MS-DOS (Micros Disk Operating System). Windows NT (NT standing for New Technology) was result. Several versions of Windows NT followed and the name was changed Windows 2000. Windows XP (XP stands for eXPerience) was released in 2001.
- In 1999, Windows NT name was changed to Windows 2000 and was available four packages :
  1. Windows 2000 Professional
  2. Windows 2000 Server
  3. Windows 2000 Advanced Server
  4. Windows 2000 Datacenter Server.
- Designed for large data warehouses and other data-intensive business applicatio It supported up to 64 GB of physical memory. Windows Server 2003 was al released with these same four packages plus a web edition.

11.1.1 Microsoft Windows CE

- Windows CE is based on Windows 95 with the usual interface, adapted for sm devices. The development for this operating system under the code name Pegas began in 1995. Specially designed for micro-computers, the abbreviation **CE stan informal for "Compact Edition".**

- **Reliability :** Stable and most reliable windows operating system from Microsoft was windows 2000. Windows uses combination of hardware and software protection. Hardware protection for virtual memory and software protection for operating system resources.
- **Application compatibility of windows and POSIX :** IEEE 1003.1 is standard used by POSIX. Without changing source code of windows, user cannot run POSIX application on windows platform.
- **Performance :** Windows XP OS provides high performance for desktop systems. Message passing method is used for interprocess communication. Windows support preemption of low priority threads enables the system to respond quickly to external events.
- **Extensibility :** Extensibility means to support or add new technologies. Windows XP is layered architecture system. Kernel runs in protected mode. An environment subsystems operates in user mode providing different API.
- **Portability :** Windows XP support portability. Source code of windows XP is in C and C++ language. Windows 7 OS can be moved from one hardware platform to another with relatively few changes. Platform-dependent code is isolated in a Dynamic Link Library (DLL) called the "Hardware Abstraction Layer" (HAL).

11.3 Windows Architecture

- Windows architecture consists of hardware abstraction layer, kernel and executive. Fig. 11.3.1 shows windows XP architecture block diagram.
- Windows XP architecture is based on microkernel architecture. It is modular operating system. Windows XP is also layered operating system.

Hardware Abstraction Layer (HAL)

- HAL interacts directly with hardware, abstracting hardware specifies to the rest of the system. HAL Also interacts with the microkernel. The microkernel and the HAL combine to make Windows XP portable allowing it to run in many different hardware environments.
- HAL is responsible for mapping various low level, processor specific operations into a fixed interface that is used by the XP kernel and executive.
- HAL provides support for symmetric multiprocessing. Device drivers map devices and access them directly.
- The function of HAL is to present the rest of the operating system with abstract hardware that hides the specific details of processor versions, support chipset and other configurations.

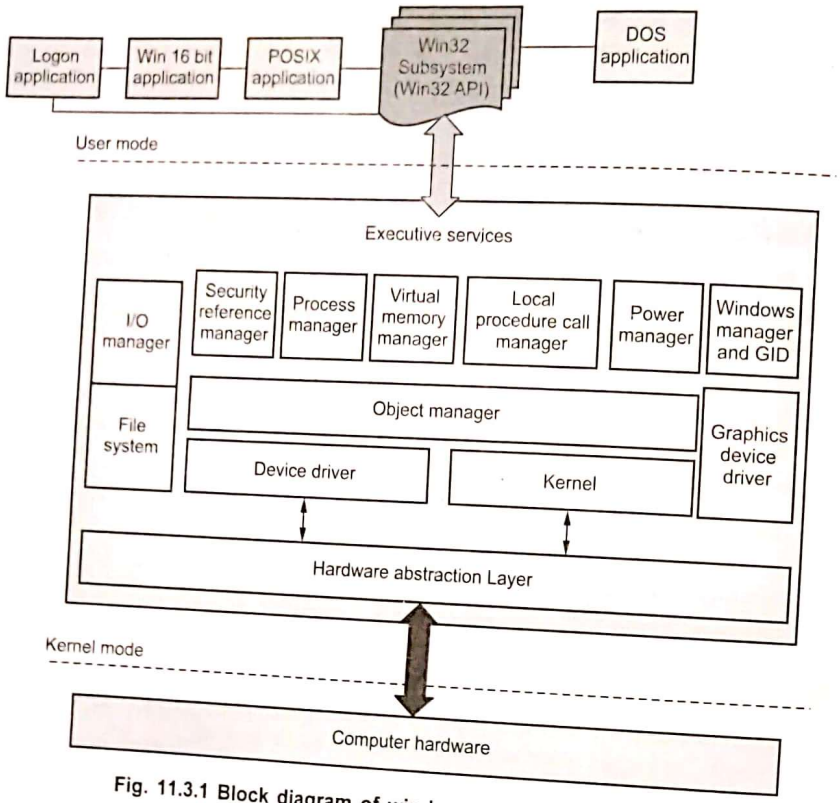


Fig. 11.3.1 Block diagram of windows XP architecture

- HAL does not provide services for specific I/O devices such as mouse, keyboards and disks.
- Layered on the top of the microkernel are the kernel mode components responsible for administering the operating system subsystems. Collectively these components are known as the **executive.**
- HAL also manages clocks and timers in a portable way. After booting the system, HAL talks to the BIOS and checks the system configuration to find out which buses and I/O devices the system contains and how they have been configured.

nel Layer

Kernel creates basic unit of computation and provides the foundation for multitasking support. The kernel provides objects and threads on top of the HAL and the hardware. Software that uses the kernel can be defined using objects and threads as primitives.

- Functions of kernel :
  1. Thread scheduling
  2. Interrupt and execution handling
  3. Low level processor synchronization
  4. Recovery after power failure.
- To implement objects and threads, the kernel must manage the hardware interrupts and exceptions, perform processor scheduling and handle multiprocessor synchronization.
- The kernel's thread scheduler is responsible for determining which thread is executing on each CPU in the system. When switching from one thread to another thread, the scheduler runs on the CPU and ensures that the registers and other hardware state have been saved.
- Thread state are ready, standby, running, waiting, transition and terminated. The dispatcher uses a 32 level priority scheme to determine the order of thread execution. Priorities are of two types: real time class and variable class.
- As in multiple-level queue scheduler, as long as there are threads in the highest priority queue, then only those threads will be allocated the processor. If there is no thread in that queue, then the scheduler will service the threads in the second highest priority queue.
- In the second highest priority queue, threads are not in the ready state, the scheduler will service the third highest priority queue and so on.
- Windows XP is not a real time system and cannot guarantee that threads running at high priority will receive the processor before any fixed deadline.
- Kernel is object oriented. Kernel objects are intended to be fast. They run in supervisor mode in the trusted context.
- Kernel object cannot be manipulated directly by user mode programs, only through function calls.
- Kernel objects are of two types : ***Control object and dispatcher object.***
- **Control object** implement mechanism to control the hardware and kernel resources. Control objects are data structures that the kernel layer provides as abstractions to the executive layer for managing the CPU.
- **Dispatcher objects** are used to implement threads along with their scheduling and synchronization operations.
- In windows, objects may be either named or unnamed.

Software Interrupts

- Software interrupts are of two types : ***Asynchronous and deferred procedure call.***

1. Asynchronous Procedure Call *(APC)*

- APC execute in the context of a specific thread. APC is used to break into the execution of a specified thread and to cause a procedure to be called in a specified processor mode.
- The system uses APCs to complete various functions that must be done in a specific threads context. Each thread maintains two APC queues : One for kernel mode and another for user mode APC.
- Software interrupts that are generated by kernel mode components are **the kernel mode APC**. After allocating processor, thread must process all pending kernel mode APC.
- User mode thread can create a user APC and request to queue this APC to another thread. A thread executes user mode APCs when the thread enters an alertable wait state.
- User mode APC can also be used to deliver notification of I/O completion in user mode to the thread that initiated the I/O.

2. Deferred Procedure Call (DPC)

- DPCs are software interrupts. They execute at dispatch interrupt request levels (IRQL). DPC are used to postpone interrupt processing to avoid delaying hardware interrupts.
- DPC is also used to implements timers and inter-process communication.
- When device driver generates a DPC, the system places the DPC in a queue associated with a specific processor. The driver routine can request a processor to which the DPC should be queued. If processor is not specified, the processor queues the DPC to itself. The routine can also specify the DPC priority.

Object Manager

- Object manager is the executive components.
- In Windows XP, each object is defined by an object type. Object types are created by executive components. The object manager maintains the windows XP internal name space.
- Object types in windows XP include devices, files, threads, processes, pipes, semaphores etc. each windows XP object is an instance of an object type.
- The object manager is responsible for creating and deleting objects and maintaining information about each object type.

Virtual Memory Manager (VMM)

- Windows XP allocates a unique 4GB virtual address space to each process. I default, a process can access only the first 2GB of its virtual address space. Th remaining 2GB address space is used for kernel components and this is called system space.
- Virtual memory manager uses a two level hierarchical addressing systen Windows XP divides virtual memory into fixed size pages which it stores either i page frames in main memory or files on disk.
- Virtual memory manager assigns each process one **page directory table.** The pag directory table contains page directory entries.
- Each page directory entries points to a **page table.** Page tables contain page table entries. Each page table entries points to a page frame in main memory or a location on disk.
- VMM uses copy on write pages and employs lazy allocation. Windows XP allows applications to allocate large pages. Large page is a set of contiguous pages. Operating system consider this as one page. The system manages copy on write pages using prototype page tables.

Memory Allocation

- Memory allocation is in three stages :
  1. Reserve
  2. Commit
  3. Access

  a. A process must first reserve space in its virtual address space.

  b. A process may not access space it has reserved until it commits the space. Process usually commits a page only when it is read to write to the page.

  c. When a process is ready to use its committed memory, it accesses the committed virtual memory.

Process Manager

- Process consists of program code, an execution context, resources and one or more threads. Process manger provides services for creating, deleting and using processes and threads.
- Process and threads is the object. Windows processes are implemented as objects. Both process and thread objects have built in synchronization capabilities.

File System

- Windows XP file system consists of three components
  1. Volume drivers for controlling interfaces
  2. Core Technology: File system (NTFS) or file allocation table
  3. File system filter drivers

NTFS

- Windows XP uses NTFS file system
- Features
  1. NTFS uses the windows object model to enforce security
  2. In NTFS it is possible to define multiple data streams for a single file
  3. Volume is fundamental entity for NTFS. It keeps a log of all changes made to files on the volume
  4. It provides file-level security, compression and auditing
  5. It support for large volumes and powerful storage solutions such as built-in RAID support
  6. The ability to encrypt files and folders to protect user sensitive data
  7. It is more efficient drive management due to its smaller cluster size capabilities
- Windows XP/2000 supports compression on individual files, folders, and entire NTFS volumes. Files compressed on an NTFS volume can be read and written by any Windows-based application without first being decompressed by another program.
- The Encrypting File System (EFS) provides the core file encryption technology used to store encrypted files on NTFS volumes. EFS keeps files safe from intruders who might gain unauthorized physical access to sensitive, stored data.
- NTFS makes use of sector, cluster and volume. The cluster is the fundamental unit of allocation in NTFS which does not recognize sectors.
- Master file table (MFT) is important file on the NTFS volume. It contains information about all files on the volume. Information includes metadata of file, creation time, name, attributes etc. The MFT is divided into fixed size records. Each file has an entry in MFT.
- NTFS stories directories as files. Each directory file contains an attribute called index that stores the list of files inside the directory. Windows XP allows multiple directory entries to point to the same physical file.

- NTFS permits users to compress files and folders through the GUI interface or system calls. It uses the Lempel-Ziv algorithm to compress files. NTFS supports segmented compression dividing files into compression units and compressing one unit at a time.
- The actual file data of an NTFS file are stored in one or more data streams. A data stream is simply an attribute in NTFS file that contains file data.
- Security of an NTFS volume is derived from the Windows object model. Each object has a security descriptor attribute stored in the MFT record. NTFS uses public/private key pair to encrypt files.
- NTFS makes it possible to recover the file system to a consistent state following system crash or disk failure.