Syllabus

Mumbai University
Revised syllabus (Rev-2016) from Academic Year 2017-18

Operating System

Course Code	Course Name	Credits
CSC405	Operating System	4

Course Objectives :

1. To introduce basic concepts and functions of operating systems.
2. To understand the concept of process, thread and resource management.
3. To understand the concepts of process synchronization and deadlock.
4. To understand various Memory, I/O and File management techniques.

Course Outcomes : At the end of the course student should be able to

1. Understand role of Operating System in terms of process, memory, file and I/O management.
2. Apply and analyse the concept of a process, thread, mutual exclusion and deadlock.
3. Evaluate performance of process scheduling algorithms and IPC.
4. Apply and analyse the concepts of memory management techniques.
5. Evaluate the performance of memory allocation and replacement techniques.
6. Apply and analyze different techniques of file and I/O management.

Prerequisite : Computer Organization & Architecture

Sr. No.	Module	Detailed Content	Hours
1.	**Operating System Overview**	Operating System Objectives and Functions, The Evolution of Operating Systems, OS Design Considerations for Multiprocessor and Multicore architectures, Operating system structures, System Calls, Linux Kernel and Shell. **(Refer Chapter 1)**	**8 hrs.**
2.	**Process Concept and Scheduling**	**Process :** Concept of a Process, Process States, Process Description, Process Control Block, Operations on Processes. **Threads :** Definition and Types, Concept of Multithreading, Multicore processors and threads. **Scheduling :** Uniprocessor Scheduling - Types of Scheduling : Preemptive and, Non-preemptive, Scheduling Algorithms: FCFS, SJF, SRTN, Priority based, Round Robin, Multilevel Queue scheduling. Introduction to Thread Scheduling, Multiprocessor Scheduling and Linux Scheduling. **(Refer Chapter 2)**	**8 hrs.**
3.	**Synchronization and Deadlocks**	**Concurrency :** Principles of Concurrency, Inter-Process Communication, Process/Thread Synchronization.	**12 hrs.**
		Mutual Exclusion : Requirements, Hardware Support, Operating System Support (Semaphores and Mutex), Programming Language Support (Monitors), Classical synchronization problems : Readers/Writers Problem, Producer and Consumer problem. **Principles of Deadlock :** Conditions and Resource Allocation Graphs, Deadlock Prevention, Deadlock Avoidance : Banker" s Algorithm for Single & Multiple Resources, Deadlock Detection and Recovery. Dining Philosophers Problem. **(Refer Chapter 3)**	
4.	**Memory Management**	**Memory Management :** Memory Management Requirements, **Memory Partitioning :** Fixed Partitioning, Dynamic Partitioning, **Memory Allocation Strategies :** Best-Fit, First Fit, Worst Fit, Next Fit, Buddy System, Relocation. Paging, Segmentation. **Virtual Memory :** Hardware and Control Structures, Demand Paging, Structure of Page Tables, Copy on Write, Page **Replacement Strategies :** FIFO, Optimal, LRU, LFU, Approximation, Counting Based. Allocation of frames, Thrashing. **(Refer Chapter 4)**	**8 hrs.**
5.	**File Management**	**File Management :** Overview, File Organization and Access, File Directories, File Sharing, Secondary Storage Management, Linux Virtual File System. **(Refer Chapter 5)**	**6 hrs.**
6.	**Input / Output Management**	**I/O Management and Disk Scheduling :** I/O Devices, Organization of the I/O Function, Operating System Design Issues, I/O Buffering, Disk Scheduling algorithm: FCFS, SSTF, SCAN, CSCAN, LOOK, C-LOOK. Disk Management, Disk Cache, Linux I/O. **(Refer Chapter 6)**	**6 hrs.**

□□□

Lab Code	Lab Name	Credit
CSL404	Operating System Lab	1

Lab Outcomes :

1. Understand basic operating system commands.
2. Understand and explore various system calls.
3. Write shell scripts and shell commands using kernel APIs.
4. Implement and analyze different process scheduling algorithms.
5. Implement and analyze different memory management algorithms.
6. Evaluate process management techniques and deadlock handling using simulator.

Descriptions :

Sr. No.	Contents
1.	Explore the internal commands of linux like ls,chdir,mkdir,chown,chmod,chgrp,ps etc.
2.	Write shell scripts to do the following : ➢ Display top 10 processes in descending order ➢ Display processes with highest memory usage. ➢ Display current logged in user and logname. ➢ Display current shell, home directory, operating system type, current path setting, current working directory. ➢ Display OS version, release number, kernel version. ➢ Illustrate the use of sort, grep, awk, etc.
3.	a) Create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system call. Explore wait and waitpid before termination of process. b) Explore the following system calls : open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid.
4.	Implement basic commands of linux like ls, cp, mv and others using kernel APIs.
5.	Write a program to implement any two CPU scheduling algorithms like FCFS, SJF, Round Robin etc.
6.	Write a program to implement dynamic partitioning placement algorithms i.e Best Fit, First-Fit, Worst-Fit, etc.
7.	Write a program to implement various page replacement policies.
8.	Using the CPU-OS simulator analyze and synthesize the following : a. Process Scheduling algorithms. b. Thread creation and synchronization. c. Deadlock prevention and avoidance.

□□□

Module 1

Module 2

Module 3

Syllabus : Concurrency : Principles of Concurrency, Inter-Process Communication, Process/Thread Synchronization

Mutual Exclusion : Requirements, Hardware Support, Operating System Support (Semaphores and Mutex), Programming Language Support (Monitors), Classical synchronization problems: Readers/Writers Problem, Producer and Consumer problem.

Principles of Deadlock : Conditions and Resource Allocation Graphs, Deadlock Prevention, Deadlock Avoidance: Banker" s Algorithm for Single and Multiple Resources, Deadlock Detection and Recovery, Dining Philosophers Problem.

Module 4

Syllabus : Memory Management : Memory Management Requirements, Memory Partitioning: Fixed Partitioning, Dynamic Partitioning, Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit, Next Fit, Buddy System, Relocation. Paging, Segmentation.

Virtual Memory : Hardware and Control Structures, Demand Paging, Structure of Page Tables, Copy on Write, Page Replacement Strategies: FIFO, Optimal, LRU, LFU Approximation, Counting Based. Allocation of frames, Thrashing

❑❑❑

List of Practical's

Module 1

CHAPTER 1

Operating System Overview

Syllabus Topics

Operating system Objectives and Functions, The Evolution of Operating Systems, OS design considerations for multiprocessor and Multicore architectures, Operating system structures, System Calls, Linux Kernel and Shell.

1.1 Introduction to Operating System

→ (Dec. 14, June 15, Nov. 15)

Q. What is operating system? MU - Dec. 2014, June 2015, Nov. 2015, 2 Marks

- An operating system is system software which manages, operates and communicates with the computer hardware and software.
- To complete the execution user program need many resources.
- The main Job of the operating system is to provide resources and services to the user program. So without operating system, a computer would be useless.

☞ **Definition of operating system**

- An operating system acts as an interface between the user and hardware of the computer and also controls the execution of application programs.
- Operating system is also called as resource manager.

☞ **Need of Operating System**

- Basically operating systems perform tasks, for example identifying input from the input devices such as keyboard, mouse etc and sending output to the output devices such as monitor, printer etc and keeping track of files and directories on the disk, and controlling peripheral devices such as secondary storage devices, printers, scanners, audio mixer.
- The heart of a computer system is a processing unit called CPU.
- User's program should get the processing unit to complete the execution and computer system should offer the service to allocate processing unit to users program.
- The operating system allocates memory to user program as per need. In the same way, user programs interact with the other user programs through devices like key board, or a mouse or even a joy stick.

Syllabus Topic
Operating System Objectives and Functions

1.2 Operating System Objectives and Functions

→ (June 15, Nov. 15)

Q. Explain different objectives of operating system. MU - June 2015, Nov. 2015, 4 Marks

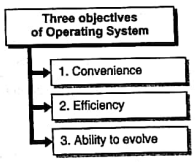


Fig. C1.1 : Three objectives of operating system

→ **1. Convenience**

Computer system can be conveniently used due to operating system.

➜ **2. Efficiency**

Computer system comprises of many resources. All these resources are utilized by user's application in efficient manner due to operating system.

➜ **3. Ability to evolve**

The design of the operating system permits the efficient development, testing. It is also supports for flexibility by allowing for addition of new system functions without interfering with service.

Functions of Operating System

➜ **(Dec. 14, June 15, Nov. 15)**

Q. Explain different functions of OS? **MU - Dec. 2014, June 2015, Nov. 2015. 5 Marks**

- Operating system comprises different modules each of having its own collection of defined inputs and outputs.
- These different modules or components of operating system carry out specific tasks to offer the complete functionality of the operating system.
- The most important functions of the operating system are shown in Fig. C1.2.

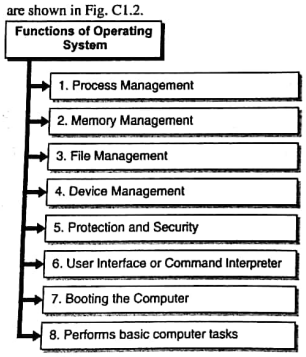


Fig. C1.2 : Functions of the operating system

➜ **1. Process Management**

The process management activities involves :

1. To provide control access to shared resources like file, memory, I/O and CPU.
2. Control the execution of user applications.
3. Creation, execution and deletion of user and system processes.
4. Resume a process execution or cancel it.
5. Scheduling of a process.
6. Synchronization, interposes communication and deadlock handling for processes.

➜ **2. Memory Management**

Following memory management related functions are carried out by OS :

1. It allocates the primary memory as well as secondary memory to the user and system processes.
2. Reclaim the allocated memory from all the processes that have finished its execution.
3. Once used block becomes free, OS allocates it again to the processes.
4. Monitoring and Keeping track of how much memory used by the process.

➜ **3. File Management**

The file management activities of operating system consist of :

1. Files and directories are created and deleted by OS.
2. OS offer the service to access the files and also it allocates the storage space for files by using different methods of allocation.
3. It keeps back-up of files.
4. It offers the security for files.

➜ **4. Device Management**

The device management tasks include :

1. Device drivers are opened, closed and written by OS.
2. Keep an eye on device driver. Communicate, control and monitor the device driver.

➜ **5. Protection and Security**

- The resources of the system are protected by the operating system.
- In order to offer the needed protection, operating system makes use of user authentication, file attributes such as read, write, encryption, and back-up of data.

➜ **6. User Interface or Command Interpreter**

User interacts with computer system through operating system. Hence OS act as an interface between the user and the computer hardware. This user interface offered through set of commands or a graphical user interface

(GUI). Through this interface user makes interaction with the applications and the machine hardware.

➜ **7. Booting the Computer**

- The process of starting or restarting the computer is known as booting.
- If computer is switched off completely and if turned on then it is called cold booting. A warm booting is the process of using the operating system to restart the computer.

➜ **8. Performs basic computer tasks**

- The management of various peripheral devices such as the mouse, keyboard and printers is carried out by operating system.
- Today most of the operating systems are plug and play. These operating systems automatically recognize and configure the devices with no user interference.

Syllabus Topic
The Evolution of Operating Systems

1.3 The Evolution of Operating Systems

Q. Explain Evolution of operating system. **(5 Marks)**

Following are the developments that have occurred in computing facility in the last four to five decades. In the 1960s, people were using mainframe computer system as a computing facility.

- The mainframe computer system would be normally kept in a computer center with a controlled environment.
- The users used to prepare their program as a deck of punched cards with encoded list of program instructions.
- Among the deck header cards which were "job control cards" would specify compilers needed to compile the program.
- The operators at the computer system would group the job as per programming languages. The jobs which required long processing time were classified as long jobs and which had short processing time was classified as "short jobs".
- The entire processing was batch processing on the basis of set of jobs.
- There was no user interaction in processing the jobs. The processor would remain busy at a time in processing one program.
- If any I/O required then processor would wait as processor speed is high with compare to I/O speed.
- As a consequence of this processor utilization was poor and would remain idle for most of the time. In this scenario only one program was kept in the memory at the time of execution.
- So it was needed to have better utilization of processor and multiple programs in memory at a time so that multiple users would connect to the system.
- In the decade of 1970s, the operating systems designers developed the concept of multiprogramming. In multiprogramming, physical memory was divided in many partitions.
- Each partition was holding exactly one program and operating system was residing in exactly one partition.
- Along with this policy of memory allocation, a policy was needed to switch the processor from one program to other.
- It has solved the problem of processor remaining idle while I/O for program is going in.
- This is what the operating system provided leading to the high throughput. A system still was operating in batch processing mode.
- In the early decade of 1980s, system designers added a feature to give interactive access to the system. In this period time sharing systems came in to picture.
- Idea behind the time sharing is to let the use fill that all system resources is given to his program while execution. To achieve it, each user program was allocated a slice of time of processor.
- In this allocated time, user program partly complete the execution.
- When time slice ends, the processor switches to other program one by one and again come back to first program in very short time.
- This gives the illusion to user that, the system is only available to him alone.
- In the decade of 1970-80, there was an exceptional growth in bulk storage. It has helped to realize the concept to propose the idea of extended storage.
- Again the concept of "virtual storage" helped to utilize this extended storage as an enhanced address space.
- By swapping in the active part of the program in memory from disk and swapping out suspended program to the disk supported to enhance the concept

of multiprogramming. This led to the large number of users using the system.

- In the mid of 1970 decade, Massachusetts Institute of Technology (M.I.T.) developed first time sharing system called Compatible Time Sharing System (CTSS).
- In decade of 1980s microcomputers came in scene and CP/M was first operating system for this platform. It was developed on Intel 8080 in 1974. MP/M, the version of CP/M, was designed as multiuser, timesharing with real time capabilities.
- After the arrival of IBM PC based on Intel 8086, PC-DOS of IBM and MS-DOS was developed.
- After the arrival of IBM 286, IBM and Microsoft developed OS/2 for 286 and 386-machines. It was multiuser system.
- After the arrival 386 and 486-based computers, to facilitate and reduce the development time, Microsoft developed Graphical User Interface (GUI) based MS-WINDOWS. It was used on top of DOS to provide user friendly GUI.
- After arrival of Pentium processors, Microsoft developed multiuser Windows NT and single user Windows 95, then to Windows 98 and Windows XP.
- After this, Windows 2000 was developed as a single user operating system for desktop users and as a multiuser system for business users to combine the both streams.
- Now Linux is emerged as more successful operating system which is a variant of UNIX. Table 1.3.1 summarizes this evolution of operating system.

Table 1.3.1

Sr. No.	Decade	Machines	Operating systems and features
1.	1970	Mainframes	MULTICS. Multiuser, timesharing.
2.	1980	Minicomputers	UNIX. Multiuser, timesharing and networked.
3.	1990	Desktop Computers	DOS, MS-WINDOWS, Wndows-95, Windows 98, Windows XP (for desktop users), Windows NT,
			UNIX/Linux etc. Multiuser, timesharing, networked, clustered.
4.	2000	Handheld Computers	Windows XP (for desktop user), Windows 2000, Linux, Multiprocessor, Multiuser and networked.

1.4 Types of Operating System

The classification of the different operating systems is given below along with a few examples of operating systems that fall into each of the categories. Many computer operating systems will fall into more than one of the following categories.

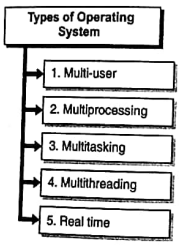


Fig. C1.3 : Types of operating system

➔ **1. Multi-user**

- Operating system is multiuser means it supports for many number of users to work on same computer at the same time.
- Some of the operating systems permit hundreds or even thousands of users working on the system concurrently.
- Some examples of multi-user operating systems : Linux, UNIX, Windows 2000, Windows.NET.

➔ **2. Multiprocessing**

- Multiprocessing operating system supports for multiple CPU in a system. It supports for allocation of different CPU to multiple threads of the program.

- Due to parallel execution of the threads, efficiency and performance can be achieved.
- Following are some examples of multiprocessing operating systems :

  Linux, UNIX, Windows 2000.

➔ **3. Multitasking**

- Operating system which supports for multitasking, permits for execution of many software processes at the same time. Multiple tasks are handled concurrently.
- Following are some examples of multitasking operating systems.
  - o UNIX
  - o Windows 2000.

➔ **4. Multithreading**

- A single program can have multiple threads that can run in parallel.
- The operating system supporting for multithreading can divide the program in multiple threads that can run in parallel.
- Such operating systems that would fall into this category are :
  - o Linux
  - o UNIX
  - o Windows 2000.

➔ **5. Real Time**

- For Real time operating systems time constrains are a key parameter.
- If the completion of particular task should happen in given time constraints or action completely must take place at a certain instant or within a certain range.
- Real time operating systems must respond quickly.
- All above OS like UNIX, Windows are not real time OS.

1.5 Operating System Services

➔ **(May 16)**

Q. Explain services provided by operating system. **MU - May 2016, 5 Marks**

Following are the six services provided by operating system for efficient execution of users application and to make convenience to use computer system :

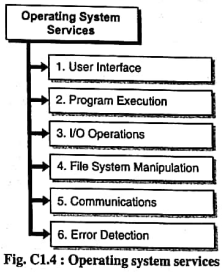


Fig. C1.4 : Operating system services

➔ **1. User Interface**

- User interface is essential and all operating systems provide it.
- Users either interface with the operating system through command-line interface or graphical user interface or GUI. Command interpreter executes next user-specified command.
- A GUI offers the user a mouse-based window and menu system as an interface.

➔ **2. Program Execution**

- The operating system provides an environment to run users programs efficiently.
- The resources needed to the programs to complete execution are provided by operating system ensuring optimum utilization of computer system.
- Memory allocation and deal location, processor allocation and scheduling, multitasking etc functions are performed by operating system.
- The operating system has all rights of resource management. User program does not given these rights.

➔ **3. I/O Operations**

- Each program requires cannot produce output without taking any input. This involves the use of I/O. During execution, program requires to perform I/O operations for input or output.
- Until I/O is completed, program goes in waiting state. I/O can be performed in three ways i.e. programmed I/O, Interrupts driven I/O and I/O using DMA.
- The underlying hardware for the I/O is hidden by operating system from users.

- These I/O operations make it convenient to execute the user program. Operating system provides this service to user program for efficient execution.

➜ **4. File System Manipulation**

- Program takes input, processes it and produces the output. The input can be read from the files and produced output again can be written into the files.
- This service is provided by the operating system. All files are stored on secondary storage devices and manipulated in main memory.
- The user does not have to worry about secondary storage management. Operating system accomplishes the task of reading from files or writing into the files as per the command specified by user.
- Although the user level programs can provide these services, it is better to keep it with operating system. The reason behind this is that, program execution speed is fundamental to I/O operations.

➜ **5. Communications**

- Cooperating processes communicates with each other. If communicating processes are running on different machines then messages are exchanged among them and they gets transfer through network.
- This communication can be realized by making the use of user programs by customizing it to the hardware that facilitates in transmission of the message.
- Customization of the user programs can be done by means of offering the service interface to operating system to realize communication among processes in distributed system.
- In this way the communication service will be provided by operating system and user programs from will be free from taking care of communications.

➜ **6. Error Detection**

- In order to prevent the entire system from malfunctioning, the operating system continually keeps watch on the system for detecting the errors.
- User programs kept free from such error detection to improve the performance.
- On the contrary, if this right would have been kept with user programs, most of the user program would have wasted time in error detection and actual work would be minimized resulting in performance degradation.
- Operating system needs to carry out complex tasks during the process of error detection.
- Such task comprises deal location of many resources such as processor, memory etc. If these tasks are again kept with user programs, then it may disturb the normal operation of operating system.

Syllabus Topic : OS Design Considerations for Multiprocessor and Multicore

1.6 OS Design Considerations for Multiprocessor and Multicore Architectures

Q. Explain OS Design Considerations for Multiprocessor and Multicore Architectures.

Two types of OS are considered for design considerations. These are

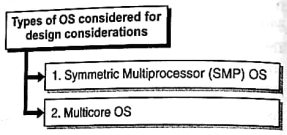


Fig. C1.5 : Types of OS

➜ 1.6.1 Symmetric Multiprocessor (SMP) OS

- In SMP system, each processor can carry out scheduling by its own from group of available processes or threads. Any processor can be chosen by kernel to execute.
- The kernel is built as multiple threads or processes so that these can run in parallel. This approach increases the complexity of OS.
- The same data structures and resources can be required by different parts of the OS simultaneously.
- The management of processors and many resources is carried out by symmetric OS in such a way that it appears like single processor system.
- The applications built by user may contain may processes or threads irrespective of several processors or single processor.
- Hence, multiprocessor OS should offer the functionality of multiprogramming system along with supporting the multiple processors.

☞ **Key design issues of multiprocessor OS**

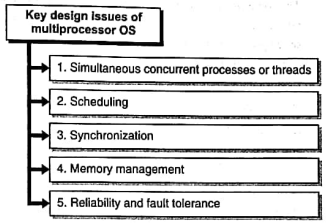


Fig. C1.6 : Key design issues of multiprocessor OS

➜ **1. Simultaneous concurrent processes or threads**

The data corruption must be avoided when multiple processors execute same or different part of the kernel, kernel tables and other management structures.

➜ **2. Scheduling**

- As scheduling can be carried out by any processor, the task of putting scheduling policy into effect becomes difficult.
- The guarantee should be given to avoid the corruption of scheduler data structures.

➜ **3. Synchronization**

As several processes can be active simultaneously and shares address space and I/O processes, the effective synchronization mechanism is necessary to implement for mutual exclusion.

➜ **4. Memory management**

- The memory management must be in line with issues related to uniprocessor system.
- As multiple processors may share page or segment, the paging mechanism should ensure coordination among different processors to achieve consistency.
- The OS should take advantage of available parallelism in hardware.
- The paging mechanisms also must ensure not to access the physical page with its old contents before the page is put to a new use.

➜ **5. Reliability and fault tolerance**

- The OS should elegantly degrade the failure of processors.
- Schedulers should know the unavailability of processor and hence, reorganize management tables for the same purpose.

➜ 1.6.2 Multicore OS

- A single processor chip can have 8 or more core on it.
- The design challenge for such many-core multi core system is to improve the processing power of multi core along with managing the considerable on-chip resources.
- It is necessary to match intrinsic parallelism existing in many-core system with the performance necessities of applications.
- Following three levels of parallelism exist in modern multicore system.
  1. Hardware parallelism in each core called as instruction level parallelism. Application programmer or compiler may or may not take advantage of it.
  2. There can be multiprogramming or multithreaded execution within each processor.
  3. The single application can be executed as many threads or processes in parallel across many cores.
- Unless and until OS provide the strong support for second and third types of parallelism, the optimum use of hardware can not be possible.

☞ **Ways to extract parallelism from given application**

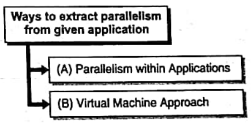


Fig. C1.7 : Ways to extract parallelism from given application

➜ 1.6.2(A) Parallelism within Applications

- The developer of the application must decide about how to split up the application's work in independently executing tasks.
- The parallel programming design process is mainly supported by language features and compilers.

- This design process also must support by OS so that the parallel task can make optimum use of resources in order to complete the execution in efficient way.
- Grand Central Dispatch (GCD) is the multicore support capability of Mac OS X 10.6. The Mac OS X 10.6 is the most recent release of the UNIX-based Mac OS X operating system.
- The GCD helps for effective execution of independent tasks that are decided by developer of application. GCD makes use of available parallelism of multicore system. GCD maps tasks on to threads to achieve concurrent execution.

➜ 1.6.2(B) Virtual Machine Approach

- The overhead required in switching of the tasks and scheduling can be avoided if one or more cores are fully dedicated to particular process.
- The allocation of cores to application is done by the OS and does little in resource allocation.
- In uniprocessor system, processor switches in user mode and kernel mode. It is just like two virtual processors.
- There is overhead in such switching in terms of responsiveness if multicore systems are introduced.
- This distinction can be dropped in multiple core system where OS acts more like hypervisor.
- The responsibility of resource management is carried out by program itself and OS allocates the processor and memory to application.
- Program takes decision about how to use resources by using metadata generated by compiler.

Syllabus Topic : Operating System Structures

1.7 Operating System Structures

Q. Explain various operating system structures.

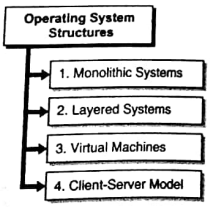


Fig. C1.8 : Operating system structures

➜ 1.7.1 Monolithic Systems

- This type of operating system can be treated as having no structure. The operating system is constructed as a set of procedures and each procedure can call any other ones whenever needed.
- Each procedure in the system has a precise interface in terms of parameters and results.
- Every procedure can call any other one, if the called one offers some useful computation that the calling procedure needs.
- In this technique, compilation of all the individual procedures or files containing the procedures is done. After this, linker is used to bind them all together into a single object file.
- Every procedure is able to be seen to every other procedure so there is no information hiding.
- A little structure is possible to impose in monolithic systems. The system calls offered by the operating system are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction.
- Due to this instruction machine gets switched from user mode to kernel mode and transfers control to the operating system.
- The operating system then fetches the parameters and find out which system call is to be executed.
- This organization suggests a basic structure for the operating system :
  - A main program that calls the demanded service procedure.
  - A collection of service procedures that perform the system calls.
  - A collection of utility procedures that assist the service procedures.
- In this representation, there is single service procedure for each system call which takes care of it.
- The utility procedures perform things that are desired by several service procedures, such as fetching data from user programs.
- This separation of the procedures into three layers is shown in Fig. 1.7.1.

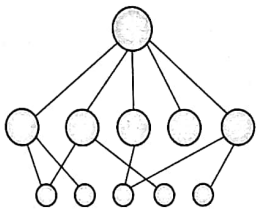


Fig. 1.7.1: A simple structuring model for a monolithic system

➜ 1.7.2 Layered Systems

Q. Write note on "Layered System".

- Another approach of organizing the operating system is as a hierarchy of layers, each one constructed upon the one below it.
- The first system built in this way was the THE system developed at the Technische Hog school Eindhoven in the Netherlands by E. W. Dijkstra (1968) and his students.
- As shown in Fig. 1.7.2 the system had total 6 layers.
  - **Layer 0 :** This layer was responsible to handle allocation of the processor, context switching when interrupts occurred or timers expired. This layer was responsible to offer the basic multiprogramming of the CPU.
  - **Layer 1 :** The memory management is handled by this layer. It was responsible to allocate space for processes in main memory. If there is no space in main memory, then it also allocates space on a 512K word drum used for holding parts of processes (pages). The layer 1 software was concerned of making needed pages were brought into memory whenever they were required.
  - **Layer 2 :** Handled communication between each process and the operator console. Above this layer each process effectively had its own operator console.
  - **Layer 3 :** The management of I/O devices and buffering the information streams to and from them was handled by this layer. On top of layer 3 each process could be able to deal with abstract I/O devices with good properties, instead of real devices with many peculiarities.
  - **Layer 4 :** The user programs were resided in this layer. They did not have to be concerned about process, memory, console, or I/O management.
  - **Layer 5 :** The system operator process was located in layer 5.

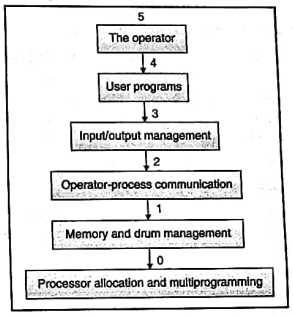


Fig. 1.7.2 : Structure of the THE operating system

- The layered approach is one of the ways to make system modular.
- In layered approach shown in Fig. 1.7.3, the operating system is broken up into a number of layers.
- The bottom layer which is layer 0 , is the hardware and the highest layer is layer N, which is the user interface.

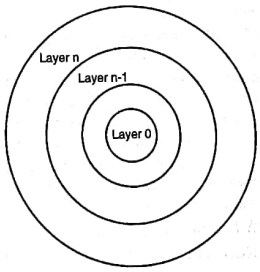


Fig. 1.7.3: A layered operating system

- A characteristic operating system layer consists of data structures and a set of routines that can be invoked by higher-level layers.
- The same layer can too invoke operations on lower-level layers.
- An operating system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.

☞ **Advantages of layered approach**

- It keeps much better control over the computer and over the applications that utilizes that computer.
- Implementers have more liberty in changing the internal workings of the system and in creating modular operating systems.
- Ease of construction and debugging.

☞ **Limitations of layered approach**

- The major trouble with the layered approach is to define various layers properly. As a layer can use only lower-level layers, cautious planning is necessary.
- Layered implementation inclined to be less efficient than other types.

➜ 1.7.3 Virtual Machines

- The early releases of OS/360 were strictly batch systems. But, due to requirement, many 360 users, decided to have timesharing, so various groups, both inside and outside IBM decided to write timesharing systems for it.
- The TSS/360, the first time sharing system, arrived late and it was so big and slow that few sites converted to it. It was finally neglected.
- And it its development cost was very high around some $50 million (Graham, 1970).
- But a group at IBM's Scientific Center in Cambridge, Massachusetts, produced a radically different system that IBM eventually accepted as a product, and which is now widely used on its remaining mainframes.
- This system, originally called CP/CMS and later renamed VM/370 (Seawright and MacKinnon, 1979), was based on an astute observation :
  - o A timesharing system provides
  - o Multiprogramming and
  - o An extended machine with a more convenient interface than the bare hardware.

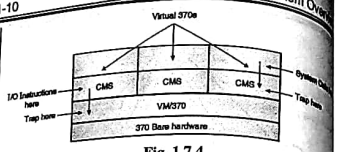


Fig. 1.7.4

- The essence of VM/370 is to completely separate the two functions. The heart of the system, known as the **virtual machine monitor,** runs on the bare hardware and does the multiprogramming, providing not one, but several virtual machines to the next layer up, as shown in Fig. 1.7.4
- However, unlike all other operating systems, these virtual machines are not extended machines, with files and other nice features.
- Instead, they are *exact* copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has.

➜ 1.7.4 Client-Server Model

- As a large part of the traditional operating system code moved into a higher layer CMS, the VM/370 got a lot of simplicity.
- However, VM/370 itself is still a complex program because simulating a number of virtual 370s in their entirety is a tough and complex task.
- A trend in modern operating systems is to take the idea of moving code up into higher layers even further and remove as much as possible from kernel mode, leaving a minimal **microkernel.**
- The usual approach is to implement most of the operating system in user processes.
- To request a service, such as reading a block of a file, a user process **(client process)** sends the request to a **server process,** which then does the work and sends back the answer.
- In this model, shown in Fig 1.7.5, all the kernel handles the communication between clients and servers.
- By splitting the operating system up into parts, each of which only handles one facet of the system, such as file service, process service, terminal service, or memory service, each part becomes small and manageable.
- As all servers run as user-mode processes, and not in kernel mode, they do not have direct access to the hardware.

- As a consequence, if a bug in the file server is triggered, the file service may crash, but this will not usually bring the whole machine down.
- Another advantage of the client-server model is its adaptableness to use in distributed systems.
- The communication of client with a server takes place by sending messages.
- When client sends a message to server, it is not necessary for client to know whether the message is processed in its own local machine, or whether it was sent to a server on a remote machine in network.
- Either server is present on client machine or on a remote machine in network, from clients point of view it appears to be : a request was sent and a response came back.

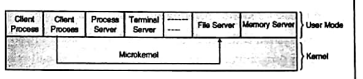


Fig. 1.7.5: Client-Server Model

1.7.5 Monolithic Kernel vs. Microkernel

- The majority of CPUs support at least two modes of operation i.e. kernel mode and user mode.
- In **kernel mode**, all instructions are allowed to be executed, and the entire memory and set of all registers is accessible throughout the execution.
- On the contrary to this, in **user mode**, memory and register access is restricted.
- The CPU gets switched to kernel mode while executing operating system code.
- The only means to switch from user mode to kernel mode is through system calls as implemented by the operating system.
- The operating system can be put in full control as system calls are the only basic services an operating system provides.
- The operating system can also be put in full control as hardware helps to restrict memory and register access.
- If virtually entire operating system code is executed in kernel mode, then it is a monolithic program that runs in a single address space.
- The disadvantage of this approach is that it is difficult to change or adapt operating system components without doing a total shutdown and perhaps even a full recompilation and reinstallation.
- Monolithic operating systems have drawbacks from the viewpoint of openness, software engineering, reliability, or maintainability.
- If the operating system is organized into two parts, it can provide more flexibility.
- In the first part, set of modules for managing the hardware is kept which can uniformly well be executed in user mode.
- For example, memory management module keeps track allocated and free space to the processes.
- It is required to execute in kernel mode at the time when registers of the Memory Management Unit (MMU) are set.
- A small **microkernel** contains only code that must execute in kernel mode. It is the second part of the operating system.
- Actually, a microkernel require only contain the code for, context switching, setting device registers, manipulating the MMU, and capturing hardware interrupts.
- Microkernel also contains the code to pass system calls to calls on the proper user level operating system modules, and to return their results.
- Following is the organization of operating system with this approach.

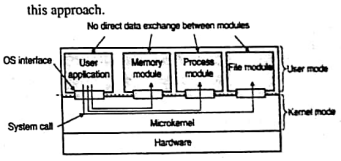


Fig. 1.7.6 : Separating applications from operating system code through a microkernel

☞ **Difference between monolithic and microkernel**

➜ **(June 15, Nov. 15)**

Q. Differentiate between monolithic and microkernel. MU - June 2015, Nov. 2015, 5 Marks

Sr. No.	Parameters	Microkernel	Monolithic kernel
1.	Definition	In microkernel, set of modules for managing the hardware is kept which can uniformly well be executed in user mode. A small	If virtually entire operating system code is executed in kernel mode, then it is a monolithic program

Sr. No.	Parameters	Microkernel	Monolithic kernel
		microkernel contains only code that must execute in kernel mode. It is the second part of the operating system.	that runs in a single address space
2.	Address Space	There is separate address space for user services and kernel services.	There is same address space for user services and kernel services
3.	Size	Microkernel size is smaller than monolithic kernel.	It has a larger size compared to microkernel
4.	Execution speed	Execution speed is slower than monolithic kernel.	Execution speed is faster than microkernel.
5.	Flexible	It is more flexible. Any module can be replaced.	It is not flexible as compared to microkernel. Its component cannot be changed, without doing a total shutdown and perhaps even a full recompilation and reinstallation.
6.	Crash	Crash of one service does not affect working of other part of microkernel.	After crash of service complete operating system fails.
7.	Communication	In microkernel, communication is through message passing.	Kernel calls the function directly.
8.	Drawbacks	Performance degradation due to context switching and communication overhead between different modules.	Monolithic operating systems have drawbacks from the viewpoint of openness, software engineering, reliability, or maintainability. It has lack of flexibility and having huge size.

Syllabus Topic : System Calls

1.8 System Calls

➔ (June 15, Nov. 15, May 16)

Q. What are system calls?

MU - June 2015, Nov. 2015, 4 Marks

Q. Write short note on system calls.

MU - May 2016, 5 Marks

- The interface between OS and user programs is defined by the set of system calls that the operating system offers. System call is the call for the operating system to perform some task on behalf of the user's program. Therefore system calls make up the interface between processes and the operating system.
- The system calls are functions used in the kernel itself. From programmer's point of view, the system call is a normal C function call.
- Due to system call, the code is executed in the kernel so that there must be a mechanism to change the process mode from user mode to kernel mode.
- In the UNIX operating system, user applications do not have direct access to the computer hardware. Applications first request to the kernel to get hardware access and access to computer resources.
- During execution when application invokes a system call, it is interrupted and the system switches to kernel space. The kernel then saves the process execution context of interrupted process and determines purpose of the call.
- The kernel warily makes sure that the request is valid and that the process invoking the system calls has sufficient privilege. Some of the system calls can only be invoked by a user with super user privilege.
- If the whole thing is fine, the kernel processes the request in kernel mode and can access the device drivers in charge of controlling the hardware. The data of the calling process can be read and modified by kernel, as it has access to memory in user space.
- But, it will not execute any code from a user application, for clear security reasons.
- When the kernel finishes the processing of request, it restores the process execution context that was saved when the system call was invoked.

- After this, control returns to the calling program which persists executing.

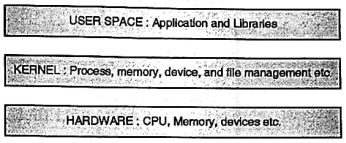


Fig. 1.8.1 : The kernel

- As shown in Fig. 1.8.1 the kernel is a central module of most computer operating systems. Its main responsibilities are shown in Fig. C1.9.

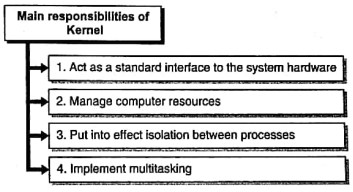


Fig. C1.9 : Responsibilities of Kernel

➔ **1. Act as a standard interface to the system hardware**

For example, while reading a file, application does not have to be aware of the hard-drive model or physical geometry as kernel provides abstraction layer to the hardware.

➔ **2. Manage computer resources**

As several users and programs shares machine and its devices, access to those resources must be synchronized. The kernel implements and ensures a fair access to resources such as the processor, the memory and the devices.

➔ **3. Put into effect isolation between processes**

The kernel guarantees that one process cannot corrupt the execution environment of another process. Any process cannot access the memory that the kernel has allocated to other process.

➔ **4. Implement multitasking**

Each process gets the false impression that it is running without interrupted on its own processor. Actually, several processes compete constantly for system resources : the kernel keeps switching the active process for each processor behind the scene

- There are two ways for the process to switch from the user mode to the kernel mode. These are :
  1. A user process can explicitly request to enter in kernel mode by issuing a system call.
  2. During the execution of user process kernel can take over to carry out some system housekeeping task.
- The kernel mode is both a software and hardware state. Modern processors offer a advantaged execution mode, called as Supervisor Mode in which only kernel runs.
- The privileged operations are such as modifying special registers, disabling interrupts, accessing memory management hardware or computer peripherals.
- If it is not in supervisor mode, the processor will reject these operations.
- System calls take place in different ways, depending on the computer in use.
- Apart from the identity of the desired system call, more information is needed.
- The precise type and amount of information differ according to the particular operating system and call.
- Consider the example of getting the input. We may require specifying the source, which can be file or device.
- We also need to specify the address and length of the memory buffer into which the input should be read.
- The device or file and length may be implicit in the call.
- Parameters can be passed to the operating system by following three different ways :
  1. Pass the parameters in registers.
  2. If there are more parameters than registers, then the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register. This is the approach taken by Linux and Solaris.
  3. Program can place the parameters onto the stack which then popped off the stack by the operating system.

Some operating systems favor the block or stack method, because those methods do not limit the number or length of parameters being passed.

1.9 Types of System Calls

➔ (June 15, Nov. 15)

Q. Explain any five system calls. MU - June 2015, Nov. 2015, 6 Marks

System calls are categorized in five groups. These are shown in Fig. C1.10.

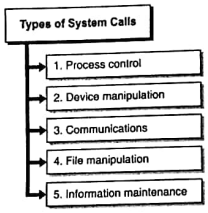


Fig. C1.10 : Types of system calls

Sr. No.	Group	Examples
1.	Process control	end, abort, load, execute, create process, terminate process, get process attributes, set process attributes, wait for time, wait event, signal event, allocate and free memory
2.	Device manipulation	request device, release device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices
3.	Communications	create, delete communication connection, send, receive messages, transfer status information, attach or detach remote devices.
4.	File manipulation	create file, delete file, open, close, read, write, reposition, get file attributes, set file attributes
5.	Information maintenance	get time or date, set time or date, get system data, set system data, get process, file, or device attributes, set process, file, or device attributes

Some Examples of System Calls

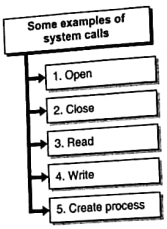


Fig. C1.11 : Examples of system calls

➔ **1. Open**

- Open system call request to the operating system for using a file.
- The files path is used as argument to specify the file. The 'flags' and 'mode' arguments to this call specifies the way of using the file.
- On successful approval by operating system, it returns a "file descriptor" which is a positive integer. Returning of value-1 indicates denial of access to file and "errno" needs to be checked to get reason of denial.

➔ **2. Close**

Close() informs to the operating system that file with said file descriptor is used. After this, same file descriptor can be reused by operating system.

➔ **3. Read**

- **Read()** specifies to the OS the number of (size) bytes to read from the file opened in file descriptor **"fd"**.
- It also specifies to put these bytes into the location pointed to by **"buf"**. The return value is number of bytes actually read.

➔ **4. Write**

Write() is similar to **read() system call**, only it writes the bytes instead of reading them. It returns the number of bytes actually written, which is almost invariably **"size"**.

➔ **5. Create process**

- During the execution process can create new process by using *create process* system call.

- The process which creates new process is called a parent process, and the new processes are called the children of that process.
- Newly created processes may in turn create other new processes, creating a tree of processes.

Syllabus Topic : Linux Kernel and Shell

1.10 Linux Kernel and Shell

Design principles of Linux

Q. Explain Linux Kernel.

- The design of Linux looks like any other traditional, non-microkernel UNIX implementation. It is a multiuser, multitasking system consisting of UNIX-compatible tools.
- The file system of Linux sticks on to traditional UNIX semantics. Linux also completely implements standard UNIX networking model.
- The internal details of Linux's design have been influenced heavily by the history of this operating system's development.
- Initially Linux was developed solely on PC architecture. From the beginning of the design, Linux tried to acquire as much functionality as possible from limited resources.
- Linux can run on multiprocessor system having main memory in several megabytes with disk space of several gigabytes. Apart from this, it can run on system with 4 MB of RAM.
- With the enhancement in machine configuration and efficiency, Linux kernels grew to implement more UNIX functionality. Apart from speed and efficiency, the current work and design focused on standardization.
- Due to variety of UNIX implementations at present available the problems faced are : source code written for one may not essentially compile or run correctly on another.
- The identical system calls on two different UNIX systems do not essentially perform equally. The POSIX standards include a set of specifications for diverse features of operating-system behavior.
- The POSIX documents are in existence for common functionality of operating-system and also for additional aspects such as process threads and real-time operations.
- Linux is designed to be agreeable with the related POSIX documents; at least two Linux distributions have obtained official POSIX certification.
- Fig. 1.10.1 shows layers in Linux system.

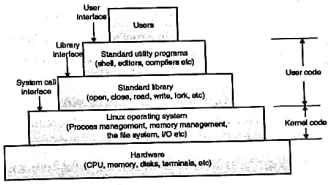


Fig. 1.10.1 : Layers in Linux System

System Hardware

- It consist of consisting of the CPU, memory, disks, terminals, and other devices.

Linux Operating System

- Its function is to control the hardware and make available a system call interface to all the programs. These system calls permit user programs to create, delete and manage processes, files, and other resources.

Standard Library

- Programs issues a trap instruction to switch from user mode to kernel mode and makes a system call by putting argument in stack or register. Library contains procedures implemented in assembly language for these system calls.

Standard Utilities

- These programs are invoked by users while working on terminal. It includes the shell, compilers, editors, text processing programs, and file manipulation utilities.
- **User :** All the users interacting with system.

There are three interfaces, that is, true system call interface, the library interface, and the interface formed by the set of standard utility programs

1.10.1 Linux Kernel Structure

➔ (Dec. 16)

Q. Explain LINUX operating system with Kernel MU - Dec. 2016, 10 Marks

- The structure of the Linux system is shown in Fig. 1.10.2.
- The kernel directly interact with hardware and facilitates interactions with I/O devices and the memory management unit and controls CPU access to them.
- As shown in Fig. 1.10.2, interrupt handlers are at lowest level and are the main means for interacting with devices.
- At lowest level, low-level dispatching mechanism is present. This dispatching takes place on the occurrence of interrupt.
- The running process is halted by low level code and saves process state in the kernel process structures. It then starts the appropriate driver.
- The dispatching of process is also carried out after the kernel finishes some operations and it is turn to start up a user process again.
- The code for dispatching is integrated in assembler and is rather different from scheduling.

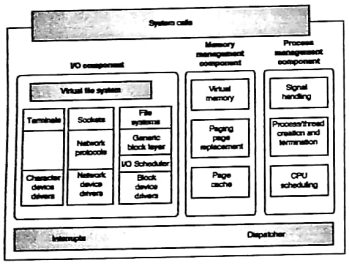


Fig. 1.10.2 : Linux Kernel Structure

The kernel subsystem is divided into three main components : I/O component, memory management component and process management component.

The I/O component

- This I/O component includes all kernel components which helps to interact with devices and carry out network and storage I/O operations.
- All the I/O operations are integrated under a **VFS (Virtual File System)** layer as shown at highest level in Fig. 1.10.2. At the lowest level, all I/O operations pass through some device driver.
- Linux drivers are either character-device drivers or block-device drivers.
- The seeks and random accesses are permissible on block devices but not on character devices. Network devices are in fact character devices, but they are managed somewhat in a different way.
- Above the device-driver level, the kernel code is different for each device type.
- The networking software is modular and have support for different devices and protocols.
- The layer above the network drivers handles a kind of routing function, ensuring the right packet is deliver to the right device or protocol handler.
- I/O scheduler which is situated on top of the disk drivers is order and issue the disk-operation requests in a manner that tries to preserve wasteful disk head movement or to meet some other system policy.
- File system is present on the top of block devices column. Linux can have multiple file systems coexisting concurrently.

Memory management Component

- This component carries out memory management tasks.
- Memory management tasks include maintaining the virtual to physical-memory mappings, maintaining a cache of recently accessed pages and implementing a good page-replacement policy, and on-demand bringing in new pages of needed code and data into memory.

Process Management Component

- Process-management component carry out creation and termination of processes.
- The process scheduler present in this component chooses the thread to be executed next. the processes and threads are treated as simply as executable entities by Linux, and global scheduling policy is used for scheduling purpose. Finally, code for signal handling also belongs to this component.

1.10.2 Shell

Q. Write short note on Shell.

- A user interface is provided by program called as shell. Users can type in commands and run programs on a UNIX system with a shell.
- Shell read in from the terminal the typed command by user, run these commands, and display the output of the commands.

- User can write programs using the C shell and these programs are called shell scripts. In other words
- The shell on its own has restricted capabilities. If we use it it as a "glue" language to combine the standard UNIX utilities, and custom software, then useful tool can be realized instead of using component parts alone.
- A shell script can be written by using any shell. For this the first line of every script is: #!/path/to/shell (e.g. #!/bin/ksh).
- A input to a shell can be a any file by using the syntax: ksh myscript
- If chmod is used to make file executable, it turns into a new command and available for use chmod +x myscript
- A shell script is a simple sequence of commands that user often types. If we put them into a script, it can be reduced to a single command. *sh the* aka "Bourne" shell, *csh* the "C" shell, *tcsh* the "TC" shell. *ksh* the "Korn" shell are the examples of different shells
  - Linux offer GUI but still it offer the facility of shell as many programmers and users prefer to use it. The command line interface is called the shell. Bourne shell (*sh*), *ksh* , *bash* are the examples of the shell.
  - Here we will discuss the Bourne shell. When the shell starts up, it initialize itself, then types a **prompt** character, often a percent or dollar sign, on the screen and waits for the user to type a command line. When the user gives command by using command line, the shell removes the first word from it.
  - After extraction of this first word, the shell assumes that, it is the name of a program to be run. Then it finds this program, and if found, it runs the program. Until the program terminates, shell suspends itself, at which time it attempts to read the next command.
  - The shell is just a simple common user program having capability to read from and write to the terminal, and the power to run other programs. Following is the description of bourne shell.

1.11 Exam Pack (University and Review Questions)

Q. What is operating system? *(Refer section 1.1)* (2 Marks) **(Dec. 2014, June 2015, Nov. 2015)**

Syllabus Topic : Operating System Objectives and Functions

Q. Explain different objectives of operating system. *(Refer section 1.2)* (4 Marks) **(June 2015, Nov. 2015)**

Q. Explain different functions of OS ? *(Refer section 1.2)* (5 Marks) **(Dec. 2014, June 2015, Nov. 2015)**

Syllabus Topic : The Evolution of Operating Systems

Q. Explain Evolution of operating system. *(Refer section 1.3)*

Q. Explain services provided by operating system. *(Refer section 1.5)* (5 Marks)

Syllabus Topic : OS Design Considerations for Multiprocessor and Multicore Architectures

Q. Explain OS Design Considerations for Multiprocessor and Multicore Architectures. *(Refer section 1.6)*

Syllabus Topic : Operating System Structures

Q. Explain various operating system structures. *(Refer section 1.7)*

Q. Write note on "Layered System". *(Refer section 1.7.2)*

Q. Differentiate between monolithic and microkernel. *(Refer section 1.7.5)* (5 Marks) **(June 2015, Nov. 2015)**

Syllabus Topic : System Calls

Q. What are system calls? *(Refer section 1.8)* (4 Marks) **(June 2015, Nov. 2015)**

Q. Write short note on system calls. *(Refer section 1.8)* (5 Marks) **(May 2016)**

Q. Explain any five system calls. *(Refer section 1.9)* (6 Marks) **(June 2015, Nov. 2015)**

Syllabus Topic : Linux Kernel and Shell

Q. Explain Linux Kernel. *(Refer section 1.10)*

Q. Explain LINUX operating system with Kernel *(Refer section 1.10.1)* (10 Marks) **(Dec. 2016)**

Q. Write short note on Shell. *(Refer section 1.10.2)*

□□□

Module 2

CHAPTER 2

Process Concept and Scheduling

Syllabus Topics

Process : Concept of a Process, Process States, Process Description, Process Control Block, Operations on Processes
Threads : Definition and Types, Concept of Multithreading, Multicore Processors and Threads.
Scheduling : Uniprocessor scheduling, Types of Scheduling: Preemptive and Non-preemptive Scheduling Algorithms: FCFS, SJF, SRTN, Priority Based, Round Robin, Multilevel Queue Scheduling, Introduction to Thread Scheduling, Multiprocessor Scheduling and Linux Scheduling.

2.1 Introduction

- Process manager implements the process management functions. In multiprogramming, single CPU is shared among many processes.
- If many processes remain busy in completing I/O, CPU is allocated to only one process at a given point of time.
- Here some policy is required to allocate CPU to process, called as CPU scheduling. If multiple users are working on the system, operating system switches the CPU from one user process to other.
- User gets the illusion that only he or she is using the system. Process synchronization mechanism is required to ensure that only one process should use critical section.
- Process communication, deadlock handling, suspension and resumption of processes and creation and deletion of the processes etc are some of the activities performed in process management.

Syllabus Topic : Process - Concept of Process

2.1.1 Process

→ (Dec. 14)

Q. What do you mean by process? MU - Dec. 2014. 4 Marks
Q. Explain the concept of process.

- A program or application under execution is called as process. A process includes the execution context. A program resides on the disk. On disk it does not require any resources. A program gets executed in main memory. So it should be transferred from disk to main memory.
- To complete execution, program needs many resources and competes for it. Now it becomes process.
- From the computation context point of view, a process is defined by CPU state, memory contents and execution environment.
- A CPU state is defined by the content of the various registers such as Instruction Register (IR), Program Counter (PC), Stack Pointer (SP) and general purpose registers.
- A small amount of data is stored in CPU registers. Memory contains program code and its predefined data structures.
- Heap is reserved memory area for dynamically allocation of memory to the program at run time.
- In stack, program local variables are allocated and return values of function call are stored. Some register values are also saved in stack.
- Execution environment includes open files, communication channels to other processes etc. following are the components of the process :
  - The object code that is to be executed.
  - Resources required by the program to complete the execution.
  - The data on which program will operate.
  - Program execution state.

2.2 Context Switch

Q. What is context switch?

- When CPU switches from one process to other, a context switch occurs. A context switch is the switching of the CPU (Central Processing Unit) from one process or thread to another.
- When context switch occurs, operating system restores the information of currently executing process for the later use.
- When CPU again gets allocated to the same process, the restored information is used to resume the execution.
- The context of a process is represented in the Process Control Block (PCB) of a process.
- The information needs to be restored includes address of the next instruction to be executed (program counter), CPU register contents, pointers to the memory allocated to the process, scheduling information, changed state, I/O state, accounting information etc.
- While context switch, system does not perform any useful work. So context switch is pure overhead on the system.
- The speed of context switching depends on the number of registers that must be copied, memory speed. So context switch speed varies from system to system.

Syllabus Topic : Process States

2.3 Process States

→ (Dec. 14, June 15, Nov. 15)

Q. Draw process state transition diagram. MU - Dec 2014. 6 Marks
Q. Draw and explain five state process model. MU - June 2015, Nov. 2015. 5 Marks
Q. Draw and explain process state transition diagram.

- During execution, process changes its state. Process state shows the current activity of the process. Process state contains five states.
- Each process remains in one of these five states. There is a queue associated with each state of the process.
- Process resides on that queue as per the state in which it resides.

1. **New state**
   The new process being created.
2. **Ready state**
   A process is ready to run but it is waiting for CPU being assigned to it.
3. **Executing state**
   A process is said to be in running state if currently CPU is allocated to it and it is executing.
4. **Waiting (blocked) state**
   A process can't continue the execution because it is waiting for event to happen such as I/O completion. Process is able to run when some external event happens.
5. **Terminated state**
   The process has completed execution.

The process state transition diagram is shown in Fig. 2.3.1.

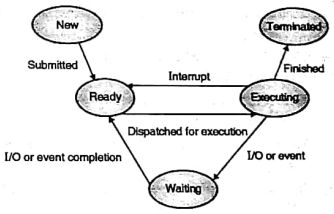


Fig. 2.3.1 : Process state transition diagram

- When the process is created, it remains in new state. After the process admitted for execution, it goes in ready state. A process in this state, wait in the ready queue. Scheduler dispatches the ready process for execution i.e. CPU is now allocated to the process.
- When CPU is executing the process, it is in executing state. After context switch, process goes from executing to ready state. If executing process initiates an I/O operation before its allotted time expires, the executing process voluntarily give up the CPU.
- In this case process transit from executing to waiting state. When the external event for which a process was waiting happens, process transit from waiting to ready state. When process finishes the execution, it transit to terminated state.

Syllabus Topic : Process Description

2.4 Process Description

- The resources required by process to complete the execution are provided by operating system.
- The OS manages all the system resources required by process. The resources that processes need are processor, I/O devices, and main memory and others.
- The OS controls the processes and provides resources to them. The OS should have some information to manage the processes and resources.
- The OS should know current status of each process and resource.
- The OS maintains tables of information about the entities. These entities are memory, I/O, files and process.
- Memory tables keep the information about main memory and secondary memory allocated to the processes.
- It also keeps information regarding protection attributes such as which processes can access the particular shared memory regions.
- The information needed to manage the virtual memory is also maintained by OS.
- Some portion of main memory is reserved for OS and in remaining part of the memory processes executes.
- The processes are swapped from main memory to secondary memory.
- The management of I/O devices and channels is carried out by OS by using I/O tables.
- If an I/O operation is currently taking place then OS should know its current status and its location in memory as source or destination.
- OS also should know about device: whether it is allocated or available.
- The file tables are maintained by OS to keep information regarding files, their location on secondary storage, their present status, and other attributes.
- This information is used by file management system. The management of processes is carried out by using information maintained in process tables.
- All these tables are linked and cross referenced in particular manner.

2.4.1 Process Control Structures

- In order to manage and control the process, the OS should be known to its location and attributes such as process ID, state etc.

Process Location

- Process contains its program and data. This program resides at certain locations in memory and data also have certain locations of local/global variables and constants.
- The stack is also provided to keep track on procedure calls and parameters passing between these procedures. The various attributes of the process is maintained in Process Control Block (PCB).
- Thus process image includes the above elements such as program, data, stack and process control block.
- The memory management scheme is responsible for deciding the location of process image.
- It can be contiguous or contiguous block of memory on secondary storage.
- For management of the process, the small portion of its image is kept in main memory.
- In order to execute the process, OS should be known about location of each process in main memory and secondary memory as well.
- Modern OS supporting paging allows some pages in main memory and remaining pages on secondary storage. The process table must show the location of each page.

Process Attributes

- In multiprogramming system, the information about each process is maintained in process control block (PCB).
- Following types of information is maintained in PCB.
  - Identification of process
  - Information of processor state
  - Process control information

Syllabus Topic : Process Control Block (PCB)

2.5 Process Control Block (PCB)

→ (Dec. 14,. Dec. 16)

Q. Explain role of process control block. MU - Dec. 2014, 5 Marks

Q. Write note on : Process Control Block. MU - Dec. 2016, 5 Marks

- Any process is identified by its Process Control Block (PCB). PCB is the data structure used by the operating system to keep track on the processes.
- All the information associated with process is kept in process control block. There is separate PCB for each process.
- Traffic controller module keeps track on the status of the processes. PCB's of the processes which are in the same state (ready, executing, waiting etc) are linked together giving a specific name to the list such as ready list.
- If many processes wait for the same device, then PCBs of all these processes are linked together in chain waiting for that device.
- If device becomes free then traffic controller checks the PCB chain to see if any process is waiting for the device.
- If the processes are available in that device chain, then again it is placed back to ready state to request that device again.
- A typical process control block is shown in Fig. 2.5.1.

Pointer	Current State
Process ID	
Priority	
Program counter	
Registers	
Accounting	
Memory allocations	
Event information	
List of open files	
. . .	

Fig. 2.5.1 : Process control block

Pointer

- This field points to other process's PCB. The scheduling list is maintained by pointer.

Current state

- Currently process can be in any of the state from new, ready, executing, waiting etc as described above.

Process ID

- Identification number of the process. Operating system assign this number to process to distinguish it from other processes.

Priority

- Different process can have different priority. Priority field indicate the priority of the process.

Program counter

- After context switch, CPU is given to other process. When turn of the previously executing process comes, again CPU gets allocated to it. Program counter contains address of the instruction from which execution resumes after context switch.

Registers

- It includes general purpose register, index registers, stack pointers and accumulators etc. number of register and type of register differs as per the architecture of computer. The state information stored after interrupt is again used by the process to resume the execution.

Accounting

- Information for calculating the process's priority relative to other processes. This may include accounting information about resource use so far. It also includes amount of CPU time and real time used, time limits, process numbers and so on.

Memory allocation

- This information may include the value of base and limit register. It includes paging, segmentation related information depending on the memory system used. Address space allocated to the process etc.

Event information

- For a process in the blocked state this field contains information concerning the event for which the process is waiting.

List of open files

- Files opened by the process.

After creation of the process, hardware registers and flags are set as per the details supplied by loaders or linkers.

At any time the process is blocked, the processor register's content are generally placed on the stack and the pointer to the respective stack frame is stored in the PCB. In this fashion, the hardware state can be restored when the process is scheduled and resume execution again.

Syllabus Topic : Operations on Processes

2.6 Operations on Processes

Q. What are the operations performed on process? Explain.

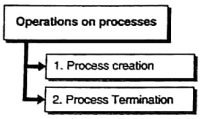


Fig. C2.1 : Operations on processes

→ 2.6.1 Process Creation

- Processes are created because of the following four principal events. These are shown in Fig. C2.2.

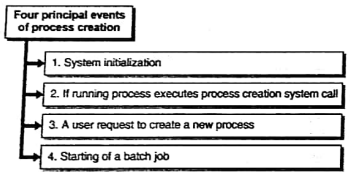


Fig. C2.2 : Principal events of process creation

→ 1. *System initialization*

After booting OS creates many processes

→ 2. *If running process executes process creation system call*

- Currently executing process creates one or more processes by issuing create process system call as it needs these processes to assist it in completion of execution.
- Creating new processes is mainly helpful when the work to be carried out can easily be formulated in terms of several related, but otherwise independent interacting processes.

→ 3. A user request to create a new process

In interactive systems, users can start a program by typing a command or (double) clicking an icon.

→ 4. Starting of a batch job

- In this case, users can submit batch jobs to the local or remote system.
- When the OS makes a decision that it has the resources to execute another job, it creates a new process to execute the next job from the input queue in it.
- During the execution process can create new process by using *create process* system call.
- The process which creates new process is called a **parent** process, and the new processes are called the **children** of that process.
- Newly created processes may in turn create other new processes, creating a **tree** of processes.
- In UNIX or the Windows family of operating systems processes are identified by unique **process identifier** (or **pid**), which is typically an integer number.
- The *ps* command is used in UNIX to obtain the listing of processes.
- By using command ps -el complete information for all processes currently active in the system can be obtained.
- Process requires certain resources like CPU time, memory, files, I/O devices to complete its task. The subprocess also needs these resources.
- Subprocess can get its needed resources directly from the operating system, or it may be forced to a subset of the resources of the parent process.
- The parent can share the resources among its children or it can divide the resources to allocate to its children.
- If child process is restricted to a subset of the parent's resources overloading the system by creating too many subprocesses can be avoided.
- The initialization data may be passed along by the parent process to the child process after creation of the process.
- After creation of a new process, two possibilities exist related to execution :
  - The parent and its children both executes in parallel.

  - Until few or all the children terminates, parent will wait and will not be terminated.

With respect to the address space of the new process again two possibilities exist :

  - Program and data of child and parent process is same. It means child is duplicate of its parent.
  - Program and data of child and parent process is different. It means child process has a new program loaded into it.

→ 2.6.2 Process Termination

- When process finishes the execution of last statement, it terminates. After this it request the operating system to delete it by using the *exit ()* system call.
- Just then, the process may return an integer to its parent process by using *wait()* system call.
- Then all allocated resources to the process like physical and virtual memory, open files, and I/O buffers are freed by the operating system.
- Termination can happen in other situation also.
- By executing the suitable system call, any process can terminate the other process.
- Normally, parent of the process to be terminated, executes this system call.
- Otherwise, users could randomly kill each other's jobs. It is necessary that a parent requires knowing the identities of its children.
- Thus, when a process creates a new process, the identity of the child process is passed to the parent.
- Any of the child processes execution can be terminated by child for a diversity of reasons, such as these :
  - If the allocated resource usage is exceeded by child. (The parent must have a means to examine the state of its children.)
  - There is no longer requirement of allocated task to the child.
  - In many systems, If the parent is terminated then OS does not permit a child to carry on execution.
- For example, VMS system does not permit a child to carry on execution after its parent process is terminated. If all children of particular parent are terminated then it is called as cascading termination. After normal or abnormal termination of parent, cascaded termination is carried out in many systems. It is initiated by the OS.
- In UNIX, exit () system call is used to terminate the process. The wait() system call is used by parent to wait until child is terminated.
- The terminated child's process identifier is returned by the wait() system call. Because of this process identifier, parent process comes to know about which child is terminated. After the termination of the parent, the *init* process is assigned as parent to all terminated children. This *init* process collects all terminated children's status and execution statistics after termination of their parent.

Syllabus Topic : Threads - Definition

2.7 Threads

2.7.1 Definition of Thread

- A thread is a single sequence stream within a process. Threads are also called as lightweight processes as it possess some of the properties of processes. Each thread belongs to exactly one process.
- In operating system that support multithreading, process can consist of many threads.
- These threads run in parallel improving the application performance.
- Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.
- Threads can share common data so they do not need to use interprocess communication.
- Like the processes, threads also have states like ready, executing, blocked etc. priority can be assigned to the threads just like process and highest priority thread is scheduled first.
- Each thread has its own Thread Control Block (TCB). Like process, context switch occurs for the thread and register contents are saved in (TCB).
- As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

2.8 Difference between Process and Thread

→ (May 16)

Q. Differentiate the Process vs Thread MU - May 2016. 5 Marks

Sr. No	Parameters	Process	Thread
1.	Definition	Program in execution called as process. It is heavy weight process.	Thread is part of process. It is also called as light weight process

Sr. No	Parameters	Process	Thread
2.	Context Switch	Process context switch takes more time as compared to thread context switch because it needs interface of operating system.	Thread context switch takes less time as compared to process context switch because it needs only interrupt to kernel only.
3.	Creation	New Process creation takes more time as compared to new thread creation.	New thread creation takes less time as compared to new process creation.
4.	Termination	New Process termination takes more time as compared to new thread termination.	New thread termination takes less time as compared to new process termination.
5.	Execution	Each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
6.	Implementation	If implementation is process based, then blocking of one process cause the blocking of other server process until the first process unblocked and these are not allowed to execute until blocked process is unblocked.	In multithreaded server implementation, if one thread is blocked and waiting, second thread in the same process could execute.
7.	Resources	Multiple redundant processes use more resources than multiple threaded process.	With compare to multiple redundant process, multiple threaded processes use fewer resources.
8.	Address Space	Context switch flushes the MMU (TLB) registers as address space of process changes.	No need to flush TLB as address space remains same after context switch because threads belong to the same process.

Syllabus Topic : Types of Threads

2.9 Types of Threads

Q. Explain different types of threads.

Threads can be implemented in two ways.

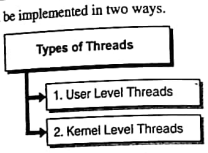


Fig. C2.3 : Types of threads

→ 2.9.1 User Level Threads

Q. Explain user level thread with advantages and disadvantages.

- In user level implementation, kernel is unaware of the thread. In this case, thread package entirely put in user space. Java language supports threading package.
- User can implement the multithreaded application in java language.
- Kernel treats this application as a single threaded application. In a user level implementation, all of the work of thread management is done by the thread package.
- Thread management includes creation and termination of thread, messages and data passing between the threads, scheduling thread for execution, thread synchronization and after context switch saving and restoring thread context etc.
- Creation and destroying of thread requires less time and is chip operation. It is the cost of allocating memory to set up a thread stack and deal locating the memory while destroying the thread.

- Both operations requires less time. Since threads belong to the same process, no need to flush TLB as address space remains same after context switch.
- User-level threads requires extremely low overhead, and can achieve high computational performance. Fig. 2.9.1 shows the user level threads.

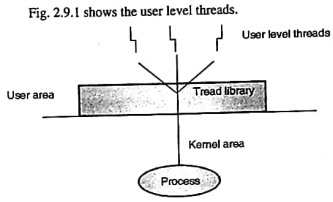


Fig. 2.9.1 : User level threads

☞ Advantages of user level threads

- Thread switching does not require flushing of TLB and doing CPU accounting. Only value of CPU register need to be stored and reloaded again.
- User level threads are platform independent. They can execute on any operating system.
- Scheduling can be as per need of application.
- Thread management are at user level and done by thread library. so kernels burden is taken by threading package. Kernels time is saved for other activities

☞ Disadvantages of user level threads

- If one thread is blocked on I/O, entire process gets blocked.
- The applications where after blocking of one thread other requires to run in parallel, user level threads are of no use.

→ 2.9.2 Kernel Level Threads

Q. Explain kernel level thread with advantages and disadvantages.

- In this, threads are implemented in operating system's kernel. The thread management is carried out by kernel.
- All these thread management activities are carried out in kernel space. So thread context and process context switching becomes same.
- Application can be written as multithreaded and threads of the application are supported as threads in single process.
- Kernel threads are generally requires more time to create and manage than the user threads.

☞ Advantages of kernel level threads

- The kernel can schedule another thread of the same process even though one thread in a process is blocked. Blocking of one thread does not block the entire process.
- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

☞ Disadvantages of kernel level threads

- Context switch requires kernel intervention.
- Kernel threads are generally requires more time to create and manage than the user threads.

Syllabus Topic : Concept of Multithreading

2.10 Concept of Multithreading

- In operating system that support multithreading, process can consist of many threads. These threads run in parallel improving the application performance.
- Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.
- Considering the advantages of user level and kernel level threads, a hybrid threading model using both types of threads can be implemented.
- The Solaris operating system supports this hybrid model.
- In this implementation, all the thread management functions are carried out by user level thread package at user space. So operations on thread do not require kernel intervention.
- The advantage of hybrid model of the threads is that, if the applications are multithreaded then it can take advantage of multiple CPUs if they are available. Other threads can continue to make progress even if the kernel blocks one thread in a system function,

Three types of multithreading models

Q. Explain multithreading models.

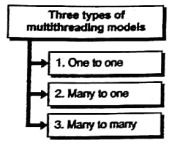


Fig. C2.4: Types of multithreading model

➜ 1. One to one model

- In this model relationship between user level thread and kernel level thread is one to one. It means that there is mapping of a single user-level thread to a single kernel-level thread.
- Because of such type of relationship multiple threads executes in parallel leading to more concurrency.
- However, since it is needed to create kernel thread for every new creation of user thread. So application performance will be degraded.
- Windows series and Linux operating systems try to minimize this problem by restricting the expansion of the thread count. OS/2, Windows NT and windows 2000 use one to one relationship model.

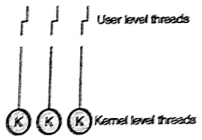


Fig. 2.10.1 : One to one model

➜ 2. Many to one model

- In this model relationship between user level thread and kernel level thread is many to one. It means that there is mapping of many user-level threads to a single kernel-level thread.
- In this model management is done in user space. When one thread makes a system call for blocking, the entire process gets blocked.
- At a time only one thread can access the Kernel thread at a time, so many other threads cannot execute in parallel on multiple processors.
- Therefore concurrent execution of threads cannot be achieved. This type of relationship facilitates an effective context-switching environment, easily implementable even on simple kernels with no thread support.

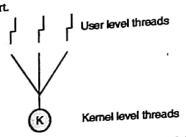


Fig. 2.10.2 : Many to one model

➜ 3. Many to Many Model

- Many to many association exist between user level thread and kernel level thread in this model.
- It means that more number of user-level threads are allied to equal or less number of kernel-level threads.
- The necessity of altering code in both kernel and user spaces leads to a level of complexity not present in the one tone and many to one model.
- Like many-to-one model, this model offers an efficient context-switching environment as it repel from system calls.
- The keen complexity offers the potential for priority inversion and suboptimal scheduling with minimum coordination between the user and kernel schedulers.

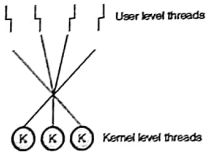


Fig. 2.10.3: Many to Many model

Syllabus Topic
Multicore Processors and Threads

2.11 Multicore Processors and Threads

Q. Explain Multicore Processors and Threads.

- When single application containing multiple threads needs to run on multicore system, there are some issues related to performance and design of application.

- The performance of the application running on multocore system depends on how this application takes advantage of parallel resources available.

Amdahl's Law

$$\text{Speedup} = \frac{\text{Time required to execute program on single process}}{\text{Time required to execute program on a n number of process}}$$

$$= \frac{1}{(1-f)+\frac{f}{n}}$$

- Where, f is the fraction of program which is inherently parallelizable. Hence, (1-f) is the fraction of program that is inherently run in serial. Practically, if small amount of serial code is present then it can be the great impact on performance of application.
- The overhead by software here includes communication overhead, overhead for assigning the work to multiple processors and overhead for cache coherence.
- In order to exploit the multicore system for parallel execution, software designers are trying to reduce the serial fraction within hardware architecture, OS, middleware and database applications. The number of classes of applications benefit by their ability to scale the throughput with number of cores. Following are the examples of these applications.

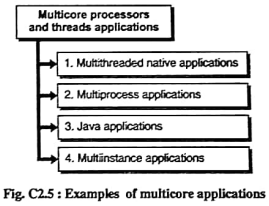


Fig. C2.5 : Examples of multicore applications

➜ 1. Multithreaded native applications

- These applications contain small number of processes which are highly threaded.
- Example is Lotus Domino or Siebel CRM (Customer Relationship Manager).

➜ 2. Multiprocess applications

- These applications contain many single threaded processes.
- Examples are Oracle database, SAP, and PeopleSoft.

➜ 3. Java applications

- Java language supports for development of multithreaded applications. Java virtual machine (JVM) is a multithreaded process that offers scheduling and memory management for Java applications.
- Java applications that can benefit directly from multicore resources comprise application servers.
- For example Sun's Java Application Server, BEA's Weblogic, IBM's Websphere, and the open-source Tomcat application server.

➜ 4. Multiinstance applications

- Single application can also be benefited from multicore architecture by running multiple instances of the application in parallel.
- If multiple application instances require some degree of isolation, virtualization technology (for the hardware of the operating system) can be used to offer each of them with its own separate and secure environment.

Syllabus Topic : Scheduling

2.12 Scheduling

2.12.1 Scheduling Queues and Schedulers

Q. With help of neat diagram explain the functions of different types of schedulers.

- System places all the entered processes in job queue. All the processes in the system reside in job queue.
- When processes are ready for the execution and wait for CPU, they kept in ready queue.
- Apart from the job queue, the operating system also has other queues.
- When processes wait for particular device, they are kept in respective devices queue.
- After allocation of CPU to the process start the execution. While executing the process, one of the numbers of events could occur.
- Until the process finishes the execution, it travels between various scheduling queues.
- The operating system selects the processes from queues based on some policy.
- This selection is carried out by the program called as scheduler.

- There are three types of schedulers to schedule a process. These are shown in Fig. C2.6.

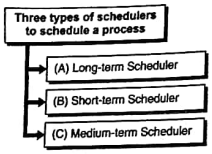


Fig. C2.6 : Types of schedulers

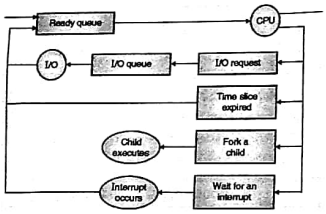


Fig. 2.12.1 : Queuing-diagram representation of process scheduling

→ 2.12.1(A) Long-term Scheduler

- When programs are submitted to the system for the purpose of processing, long term scheduler comes to know about it.
- Then its job is to choose processes from the queue and place them into main memory for execution purpose.
- CPU bound processes require more CPU time and less I/O time until execution completes.
- On the contrary, I/O bound processes use up more time in doing I/O and require less CPU time for computation.
- The main job of the long term scheduler is to provide a balance mix of I/O bound and CPU bound jobs.
- The number of processes in memory for execution and degree of multiprogramming is related to each other. More number of processes in memory for execution indicates degree of multiprogramming is high.
- Long-term scheduler controls the degree of multiprogramming.
- If the average rate of new process creation and average departure rate of processes leaving the system is equal then degree of multiprogramming is steady.
- The long term scheduler is not present in timesharing operating systems.
- When process state transition take place from new to ready, then there long term scheduler come into picture for scheduling purpose.

→ 2.12.1(B) Short-term Scheduler

- Processes which are in ready queue wait for CPU. A short term scheduler chooses the process from ready queue and assigns it to the CPU based on some policy.
- These policies can be First Come First Served (FCFS), Shortest Job First (SJF), priority based and round robin etc, Main objective is increasing system performance by keeping the CPU busy.
- It is the transition of the process from ready state to running state. Actual allocation of process to CPU is done by dispatcher.
- Short term scheduler is faster than long tern scheduler and should be invoked more frequently compare to long term scheduler.

→ 2.12.1(C) Medium-term Scheduler

- If the degree of the multiprogramming increases, medium-term scheduler swap out the processes from main memory.
- The swapped out processes again swapped in by medium-term scheduler.
- This is done to control the degree of multiprogramming or to free up a memory.
- This is also helpful to balance the mix of different processes, some time sharing operating system have this additional scheduler.

2.12.1(D) Comparison of Three Schedulers

Q. Compare the functions of different types of schedulers.
Q. Differentiate between long-term, short-term and medium-term scheduler.

Sr. No.	Long-term Scheduler	Short-term Scheduler	Medium-term Scheduler
1.	Selects processes from the queue and loads them into memory for execution.	Chooses the process from ready queue and assigns it to the CPU.	Swap in and out the processes from memory.

Sr. No.	Long-term Scheduler	Short-term Scheduler	Medium-term Scheduler
2.	Speed is less than the short term scheduler.	Speed is very fast and invoked frequently than long term scheduler.	Speed is in between both short term scheduler and long term scheduler.
3.	Transition of process state from New to Ready.	Transition of process state from Ready to Executing.	No process state transition.
4.	Not present in time sharing system.	Minimal in time sharing system.	Present in time sharing system.
5.	Supply a reasonable mix of jobs, such as I/O bound and CPU bound	Select a new process to allocate to CPU frequently.	Processes are swapped in and out for balanced process mix.
6.	It controls degree of multiprogram ming through placing processes in ready queue.	It has control over degree of multiprogrammi ng as it allocates processes to CPU for execution.	Reduce the degree of multiprogrammi ng by swapping the processes in and out.
7.	It is also called as job scheduler.	It is also called as CPU scheduler.	-

Syllabus Topic : Types of Scheduling Preemptive and Non-preemptive

2.12.2 Types of Scheduling

→ (May 16)

Q. Differentiate the Preemptive vs Non-Preemptive Scheduling. MU - May 2016, 10 Marks

Two types of scheduling algorithms are shown in Fig. C2.7.

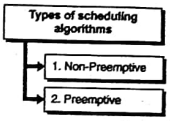


Fig. C2.7 : Types of scheduling

→ 1. Non-Preemptive

- Non-preemptive algorithms are designed so that once a process is allocated to CPU, it does not free CPU until it completes its execution.

→ 2. Preemptive

- Preemptive algorithms allow taking away CPU from process during execution. If highest priority process arrives in the system, CPU from currently executing low priority process is allocated to it. It ensures that always highest priority process will be executing.

Dispatcher and dispatch latency

- Short term scheduler allocates CPU to ready process based on some policy. It is the transition of the process from ready state to running state.
- Actual allocation of process to CPU is done by dispatcher. Short term scheduler is faster than long term scheduler and should be invoked more frequently compare to long term scheduler.
- The amount of time dispatcher needed to bring to stop one process and starts running of other process is called as dispatch latency.

Syllabus Topic : Uniprocessor Scheduling

2.13 Uniprocessor Scheduling

2.13.1 Scheduling Criteria → (May 16)

Q. Discuss various scheduling criteria. MU - May 2016, 5 Marks
Q. What are the criteria for evaluation of scheduling algorithm performance?

- In multiprogramming, many programs remain in memory at the same time. Processes carry out I/O operations while performing the execution.
- Since I/O operations generally consume more time to accomplish with compare to CPU instructions, multiprogramming systems hand over the CPU to

another ready process whenever a any process invokes an I/O operation.

- Short term scheduler allocates the processes to CPU as per some policy called as scheduling algorithms.
- The main aim of the scheduling is to improve performance by keeping CPU busy all the time.
- Criteria for the for performance evaluation of the scheduling strategy is :
  - **CPU Utilization :** It is amount of time CPU remains busy.
  - **Throughput :** Number of jobs processed per unit time.
  - **Turnaround time :** Time elapsed between submission of job and completion of its execution.
  - **Waiting time :** Processes waits in ready queue to get CPU. Sum of times spent in ready queue is waiting time.
  - **Response Time :** Time from submission till the first response is produced.
  - **Fairness :** Every process should get fair share of the CPU time.

Syllabus Topic : Scheduling Algorithms

2.13.2 Scheduling Algorithms

Q. Explain different types of scheduling algorithms.

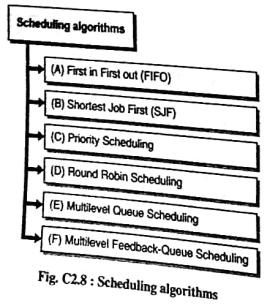


Fig. C2.8 : Scheduling algorithms

Syllabus Topic : Scheduling Algorithm - FCFS

→ 2.13.2(A) First in First out (FIFO)

- This is a Non-preemptive scheduling algorithm. FIFO strategy allocates the CPU to processes in the order of their arrival. This algorithm treats ready queue as FIFO. A process does not give up.
- CPU until it either terminates or performs I/O. If a longer job assigned to CPU then many shorter jobs have to wait.
- As long processes can hold the CPU, this algorithm gives fewer throughputs.
- This algorithm can be easily implemented using FIFO queue. In time sharing system every user should get a CPU time at regular interval.
- So FIFO algorithm is not useful in such situations. FIFO algorithm is inappropriate for interactive systems large fluctuations in average turnaround time are possible.
- If we assume that arrival time is zero. Consider the following example :

Process	Burst time
P1	24
P2	03
P3	05

Assume that processes arrive in the order P1, P2, and P3.The Gantt chart shows the result.

P1	P2	P3

0 24 27

Turnaround time for P2 = (3 – 0) = 3
Turnaround time for P3 = (8 – 0) = 8
Turnaround time for P1 = (32 – 0) = 32
Average Turnaround time = (3 + 8 + 32) / 3 = 14.33

Waiting time for P2 = 0
Waiting time for P3 = 3
Waiting time for P1 = 8
Average Waiting time = (0 + 3 + 8) / 3 = 3.66

- The result shows that there is significant reduction in average turnaround time and average waiting time. The result varies as order of job arrival varies.

Syllabus Topic
Scheduling Algorithms - SJF, SRTN

→ 2.13.2(B) Shortest Job First (SJF)

- Shortest Job First (SJF) may be preemptive or non preemptive. Reordering the jobs so as to run the shortest jobs first (SJF) improves the average response time.
- Ready queue is maintained in order of increasing job lengths.
- When a job comes in, insert it in the ready queue based on its length.
- SJF minimizes the average wait time because it gives service to less execution time processes before it gives service to large execution time processes.
- While it minimizes average wait time, it may punish processes with high execution time.
- If shorter execution time processes are in ready list, then processes with large service times tend to be left in the ready list while small processes receive service.
- It may happen in extreme case that always short execution time processes will be served and large execution time processes will wait indefinitely. This starvation of longer execution time processes is the limitation of this algorithm.
- Consider the example discussed for FCFS algorithm. In this example it is assumed that SJF is non preemptive. If the order of arrival is P2, P3, P1, the order of execution time will be 3, 5, and 24.
- There is significant reduction in average turnaround time and average waiting time.
- Consider again the following example of non preemptive shortest job first (SJF) algorithm.

Process	Arrival time	Burst time
P1	0	10
P2	1	5
P3	2	8
P4	3	15

P1	P2	P3	P4

0 10 15 23 38

Turnaround time for P1 = (10 – 0) = 10
Turnaround time for P2 = (15 – 1) = 14
Turnaround time for P3 = (23 – 2) = 21
Turnaround time for P4 = (38 – 3) = 35
Average Turnaround time = (10 + 14 + 21 + 35) / 4
= 20

Waiting time for P1 = 0
Waiting time for P2 = (10 – 1) = 9
Waiting time for P3 = (15 – 2) = 13
Waiting time for P4 = (23 – 3) = 20
Average Waiting time = (0 + 9 + 13 + 20) / 4
= 10.5

- In preemptive Shortest Job First (SJF), if newly arrived process has less execution time with compare to currently executing process, then the CPU will be given newly arrived process.
- The preemptive Shortest Job First (SJF) is called as **Shortest Remaining Time Next Scheduling (SRTN).** Consider the above example for preemptive Shortest Job First (SJF). The Gantt chart shows the result.

P1	P2	P3	P1	P4

0 1 6 14 23 38

- P1 arrives at time 0, so get the CPU as it is only process in the queue. At time 1, process P2 arrives having execution time 5 (less than remaining time 9 of p1). So P1 is preempted and P2 is scheduled. At time 2, process P3 arrives having execution time 8 which is greater than remaining time of P2.
- So P2 will complete the execution. After P2, process P3 will execute as its execution time is less than P1 (9) and P4 (15). Process P1 has remaining execution time 9. So it is scheduled next. Finally P4 will complete the execution.

Turnaround time for P1 = (23 – 0) = 23
Turnaround time for P2 = (6 – 1) = 5
Turnaround time for P3 = (14 – 2) = 12
Turnaround time for P4 = (38 – 3) = 35
Average Turnaround time = (23 + 5 + 12 + 35) /4
= 18.75

Waiting time for P1 = (14 – 1) = 13
Waiting time for P2 = (1 – 1) = 0
Waiting time for P3 = (6 – 2) = 4
Waiting time for P4 = (23 – 3) = 20
Average Waiting time = (13 + 0 + 4 + 20) / 4
= 18.75

Syllabus Topic
Scheduling Algorithm - Priority Scheduling

→ 2.13.2(C) Priority Scheduling

Q. Explain priority scheduling algorithm.

- In priority scheduling, each process has a priority which is a integer value assigned to it.
- Smallest integer is considered as highest priority and largest integer is considered as lowest priority. Always highest priority process gets the CPU.
- In some system, largest number is treated as a highest priority and it depends on implementation.
- If priorities are internally defined then some measurable quantity such as time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst are used to compute the priorities.
- External priorities are assigned on the basis of factors such as importance of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political, factors.
- All these factors are not related to the operating system.
- Preemptive and non preemptive SJF is a priority scheduling where priority is the shortest execution time of job. In this algorithm, low priority processes may never execute. This is called **starvation**.
- Solution to this starvation problem is **aging**. In aging as time progresses, increase the priority of the process so that lowest priority processes gets converted to highest priority gradually.
- Consider the following example for priority scheduling. The Gantt chart shows the result.

Process	Burst time	Priority
P1	12	4
P2	10	5
P3	5	2
P4	4	1
P5	3	3

P4	P3	P5	P1	P2
0	4	9	12	24 34

Turnaround time for P1 = (24 – 0) = 24
Turnaround time for P2 = (34 – 0) = 34
Turnaround time for P3 = (9 – 0) = 9
Turnaround time for P4 = (4 – 0) = 4
Turnaround time for P5 = (12 – 0) = 12
Average Turnaround time = (24 + 34 + 9 + 4 + 12)/5 = 16.6

Waiting time for P1 = 12
Waiting time for P2 = 24
Waiting time for P3 = 4
Waiting time for P4 = 0
Waiting time for P5 = 9
Average Waiting time = (12 + 24 + 4 + 0 + 9) / 5 = 9.8

Syllabus Topic
Scheduling Algorithm - Round Robin

→ 2.13.2(D) Round Robin Scheduling

Q. With the help of example, explain Round Robin Scheduling algorithm.

- Round Robin Scheduling is designed especially for time-sharing systems where many processes get CPU on time sharing basis. In this algorithm, a small unit of time called time quantum is defined.
- CPU is allocated to each process for this time quantum period of time. To implement this scheduling, ready queue treated as FIFO queue.
- The new processes go to the tail of queue and each time CPU chooses the process from head of queue.
- When time quantum expires, context switch occurs and CPU switches to other process which is scheduled next.
- The time quantum is fixed and then processes are scheduled such that no process get CPU time more than one time quantum.
- If process is executing and request for I/O the process goes in waiting (blocked) state.
- After the completion of I/O, process again gets added at the tail of ready queue. The time quantum should not be very small or very large.
- If the time quantum is very large, the algorithm will behave just like FCFS.
- A smaller time quantum increases context switches leading to performance degradation (less throughput) as context switches elapses more time.

- Consider the following example. If we use a time quantum of 4 milliseconds then the Gantt chart shows the result.

Process	Burst time
P1	11
P2	7
P3	15
P4	4

P1	P2	P3	P4	P1	P2	P3	P1	P3	P3

0 4 8 12 16 20 23 27 30 34 37

Turnaround time for P1 = (30 – 0) = 30
Turnaround time for P2 = (23 – 0) = 23
Turnaround time for P3 = (37 – 0) = 37
Turnaround time for P4 = (16 – 0) = 16
Average Turnaround time = (30 + 23 + 37 + 16) / 4
= 26.5
Waiting time for P1 = 0 + (16 – 4) + (27 – 20) = 19
Waiting time for P2 = 4 + (20 – 8) = 16
Waiting time for P3 = 8 + (23 – 12) + (30 – 27)
= 22
Waiting time for P4 = 12
Average Waiting time = (19 + 4 + 22 + 12)/4 = 14.25

Syllabus Topic : Scheduling Algorithm
Multilevel Queue Scheduling

→ 2.13.2(E) Multilevel Queue Scheduling

Q. Explain multilevel queue scheduling algorithm.

- Sometimes it is necessary to categorize the processes into different groups.
- For example, separation is made between interactive processes and batch processes.
- The response-time need of these two processes can be dissimilar. So these processes can have different scheduling requirement.
- Also, interactive processes may have higher priority over batch processes.
- In multilevel queue scheduling algorithm, there are multiple ready queues.
- The allocation of process to the particular queue depends on property of that process such as memory size, priority of the process, or its type.
- Scheduling algorithm for different queues can be different.
- The round robin scheduling algorithm can be used for interactive processes queue and first come first serve scheduling algorithm can be used for batch processes queue.
- In addition to scheduling for each queue, there must be scheduling between the different queues.
- The fixed-priority preemptive scheduling is used for scheduling between the different queues.
- Let us consider the five queues and its operation with multilevel queue scheduling algorithm.
- Following five queues are written below in the order of priorities assigned to them. Number 1 indicates the highest priority and 5 is lowest priority.
  1. System processes queue
  2. Interactive processes queue
  3. Interactive editing processes queue
  4. Batch processes queue
  5. Student processes queue
- Every higher priority queue has complete priority over lower-priority queues.
- Unless and until higher priority queues become empty, processes in lowest priority queue cannot run.
- For example, processes in interactive editing process queue could not execute unless and until all the processes in the queues for system processes, interactive processes finish the execution and these queues becomes empty.
- If process in lower priority queue is executing and at the same time other process belonging to higher priority queue arrives then currently executing process in lower priority queue should be preempted.
- If a batch process entered the ready queue while a student process was executing, the student process would be preempted and batch process would be scheduled for execution.
- In this algorithm, a fixed amount of CPU time is given to each queue and within this time different processes from this queue are scheduled.

- For example, the interactive process queue can be given 70 percent of the CPU time for round robin scheduling among its processes, whereas the batch queue get 30 percent of the CPU to give to its processes on an FCFS basis.

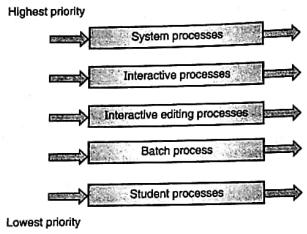


Fig. 2.13.1: Multilevel queue scheduling

➜ 2.13.2(F) Multilevel Feedback-Queue Scheduling

Q. Explain multilevel feedback queue scheduling algorithm.

- In multilevel queue scheduling processes are not permitted to transfer from one queue to the other.
- Also queue assigned to the processes are permanent and cannot be changed, leading to the low scheduling cost, but it is not flexible.
- The multilevel feedback-queue scheduling algorithm processes can move from one queue to other.
- The CPU-bound processes use more CPU time and hence it will be transferred to a lower-priority queue. I/O-bound processes use less CPU time.
- Keeping CPU bound processes in low priority queue automatically leads to keeping the I/O bound and interactive processes in the higher-priority queues. There is a chance of starvation because of processes waiting for longer period of time in a lower-priority queue.
- It is avoided by aging; by transferring these waiting processes in a higher-priority queue.
- Consider the example of a multilevel feedback-queue scheduler with four queues, queue-0, queue-1, queue-2 and queue-3. Initially the scheduler starts executing all processes in queue-0.
- Once all the processes in queue-0 finish the execution, and queue-0 becomes empty, scheduler will consider the processes in queue-1 for execution.
- After queue-1 becomes empty then scheduler will consider the processes in queue-2 for execution
- In the same way, processes in queue-3 will only be executed if queue-0, queue-1 and queue-2 are empty.
- A process arriving for queue-1 will preempt a process in queue-2. In the same way process that arrives for queue-0 will preempt a process in queue-1. Similarly if process arrives for queue-2 preempts process in queue-3.A process from the ready queue is placed in queue-0.
- Multilevel feedback queue scheduling with 3 queues are shown in Fig. 2.13.2.
- If the process does not finish the execution within allocated time quantum of 8 in queue-0, it is moved to the tail of queue-1.
- The process at the head of queue-1 is given a quantum of 16 milliseconds if and only queue-0 is empty.
- If it does not finish, it is preempted and is placed into queue-2. Processes in queue-2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- If the process are with a CPU burst of 8 milliseconds or less then this scheduling algorithm gives highest priority to it.
- This process will immediately get the CPU, finish its CPU burst, and go off to its next I/O burst.
- Processes that require above 8 but below 24 milliseconds are also served speedily, although with lower priority than shorter processes.
- Long processes automatically go down to queue 2 and are served in FCFS order with any CPU cycles left over from queues 0 and 1.

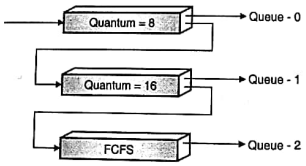


Fig. 2.13.2 : Multilevel feedback queues

Syllabus Topic
Introduction to Thread Scheduling

2.14 Introduction to Thread Scheduling

Threads can be implemented in two ways : User Level and Kernel Level.

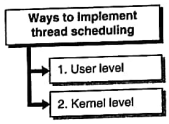


Fig. C2.9 : Ways to implement thread scheduling

➜ **1. User level**

- In **user level** implementation, kernel is unaware of the thread. In this case, thread package entirely put in user space. Java language supports threading package.
- User can implement the multithreaded application in java language. Kernel treats this application as a single threaded application.
- In a user level implementation, all of the work of thread management is done by the thread package. For these threads scheduling is carried out on a per-process basis.

➜ **2. Kernel level**

- **Kernel level** threads are implemented in operating system's kernel. The thread management is carried out by kernel.
- All these thread management activities are carried out in kernel space. So thread context and process context switching becomes same.
- Application can be written as multithreaded and threads of the application are supported as threads in single process. Kernel threads are generally requires more time to create and manage than the user threads. Here scheduling is on thread basis.
- On uniprocessor system, scheduling is done on single CPU. Hence only work involved is which process should be given next to CPU.
- In multiprocessor system, question is of which thread next and on which CPU. Threads may belongs to single application or may be from different applications which are not related to each other (independent).

Syllabus Topic : Multiprocessor Scheduling

2.15 Multiprocessor Scheduling

Q. Explain multiprocessor scheduling.

When multiple CPUs are present then scheduling involve three interrelated issues. These are:

- How to assign processes to processors?
- The use of multiprogramming on individual processors
- The actual dispatching of a process

2.15.1 Assignment of Processes to Processors

- If all the processors are similar in the sense that, there is no special advantage given to any processor with respect to memory access or I/O devices then assignment is straightforward.
- Processors can be assigned to processors on demand. Assignment can be static or dynamic.
- There is dedicated short term queue for each processor and process finish execution on that processor only.
- This approach causes less overhead in the scheduling function, as the processor assignment is done once and for all.
- The drawback of static assignment is that processor can be idle if no processes available in its queue and processor with more processes in queue can be overloaded.
- This can be prevented by maintaining global queue for all processors so that process can be assigned to any available processor.
- Hence, till process finishes the execution, it executes on many processors at different time.
- In case of tightly coupled shared memory architecture, all the processors have the context information of all processes and thus the cost of scheduling a process will be independent of the identity of the processor on which it is scheduled.
- In case of dynamic load balancing threads are moved from one processor's queue to a queue of another processor.
- Linux uses this approach. In master/slave architecture, key kernel functions of the operating system run on a particular processor for all the time and other processors may only execute user programs.

- The scheduling of job is carried out by master. Slave sends request to master for any service needed and wait for it for service to be carried out.
- This approach requires slight improvement to a uniprocessor multiprogramming operating system.
- The drawbacks to this approach are: A failure of the master leads to failure of the whole system, and the master may lead to performance bottleneck.
- In peer architecture any processor can be used for executing the kernel and processor itself carry out scheduling from the pool of available processes.
- With this approach, OS complexity increases and it must guarantee that two processors do not select the same process.
- The OS also should ensure that the processes should not be lost from the queue.
- The claims by different processes for the resources should be fulfilled by operating system.
- To resolve these issues, instead of assigning single processor to kernel, a subset of processors can be dedicated to kernel processing.
- In other approach, the requirements of kernel processes and other processes on the basis of priority and execution history is considered.

2.15.2 The Use of Multiprogramming on Individual Processors

- If one process dedicated to one processor till it finishes the execution then there is no question of multiprogramming the processor.
- In this case, processor may remain idle till I/O completes. In order to achieve high performance and processor utilization, it is important to switch the processor among multiple processes.
- If many processors are present then an average execution performance of application is important.
- Although multiple threads are present in application, all should be ready for execution simultaneously.

2.15.3 The Actual Dispatching of a Process

- In multiprogramming in uniprocesssor system, priorities and different approaches of scheduling can improve the performance.
- These approaches give better performance than FCFS as discussed.
- In case of multiprocessor scheduling, a simple approach can give more performance with lesser overhead with compare to complex approaches.

2.16 Process Scheduling

- If more than one processor is present then particular scheduling approach is less important.
- A simple FCFS approach may give better performance with less overhead.

2.17 Thread Scheduling

Q. What are general approaches used for scheduling of threads on multiprocessor system? Explain.

Following are the four general approaches used for scheduling of threads on multiprocessor system.

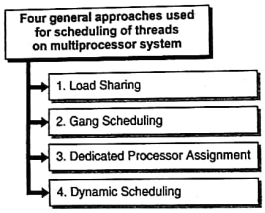


Fig. C2.10 : Approaches used for scheduling of threads

➜ **1. Load Sharing**

All the ready threads are present in single global queue and each processor selects thread from this queue when it is idle.

➜ **2. Gang Scheduling**

A set of related threads is scheduled to execute on a set of processors all at once, on a one-to-one basis.

➜ **3. Dedicated Processor Assignment**

In this approach, during execution of program the number of processors assigned is equal to number of threads available in program. All the processors returns to pool of processors after program completes the execution.

➜ **4. Dynamic Scheduling**

The number of threads in process may changes during the execution of that process.

➜ 2.17.1 Load Sharing

☞ **Advantages of load sharing**

- In this approach, load is equally distributed among all the processors so that no one remains idle.
- There is no need of centralized scheduler. The scheduling routine of the operating system executes on available processor to choose the next thread.
- The maintained global queue can be accessed by using schemes used in uniprocessor scheduling such as priority based schemes or schemes based on execution history.

☞ **Load sharing approaches**

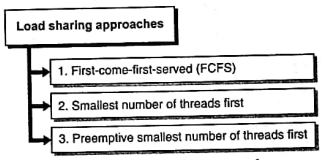


Fig. C2.11 : Load sharing approaches

➜ **1. First-come-first-served (FCFS)**

- The threads of the newly arrived job are placed consecutively at the end of shared queue.
- The idle processor selects next ready thread and executes it until it finishes execution or gets blocked.

➜ **2. Smallest number of threads first**

- If the jobs are having smallest number of unscheduled threads then highest priority is given to it.
- The shared ready queue is like priority queue and equal priority jobs are ordered according to jobs arrived first. Similar to FCFS, thread executes until it finishes execution or gets blocked.

➜ **3. Preemptive smallest number of threads first**

- Job having smallest number of unscheduled threads gets highest priority.
- If an arriving job is having smaller number of threads with compare to executing job then it preempts the threads of scheduled job.

☞ **Disadvantages of load sharing**

- A single shared ready queue may occupy the large memory portion and if many processors simultaneously access it then it may become bottleneck.
- Preempted threads may not get same processor to resume execution. The caching becomes less efficient if processors have cache memory.
- It is unlikely to get access to all the threads of the program at the same time.
- If more coordination is required between thread then process switch may lead to poor performance.

➜ 2.17.2 Gang Scheduling

☞ **Advantages of gang scheduling**

- As synchronization, blocking can be minimized and less process switching can be achieved by running the related processes in parallel there id more increase in performance.
- Scheduling overhead can be minimized as a single decision affects a number of processors and processes at one time.

Gang scheduling is better for the application whose any part is not running and other is ready for execution.

It is implemented in many multiprocessor operating systems. In this scheduling process switches are minimized as related threads run in parallel. Hence, performance is good.

As threads that require coordination run in parallel, they can access file without additional overhead and resource allocation can be done with less overhead.

In this scheduling processor allocation is required. If N processors and M applications with N or fewer threads are present then each application could be given $1/M$ of the available time on the N processors, using time slicing.

➜ 2.17.3 Dedicated Processor Assignment

- In this approach, during execution of program the number of processors assigned is equal to number of threads available in program.
- All the processors returns to pool of processors after program complete the execution.
- In this approach, if thread of an application is blocked waiting for I/O or for synchronization with other thread then processor remains idle. It can be addressed as following.
  - o If tens or hundreds of processors present in system then processor utilization does not matter for performance.
  - o As process switching is avoided, it results in better improvement in performance.

→ 2.17.4 Dynamic Scheduling

- In some applications, the number of threads changes during execution. The operating system then can adjust the load to improve performance.
- Both OS and application can carry out scheduling task.
- The operating system partitions the processors to allocate to the jobs.
- The runnable task of each job is mapped to threads and is given to the processors for execution at present in its partition.
- A proper decision, about which subset to run, in addition to which thread to suspend when a process is preempted, is left to the individual applications (maybe through a set of run-time library routines).
- This approach may be appropriate for some applications and not for all.
- Applications can be implemented to take advantage of this feature of operating system.
- Operating system is only responsible to processor allocation and performs following tasks when job demands one or more processors.
  - Allocate the idle processor to satisfy the request.
  - If job is new arrival and need processor then take the processor from currently allocated job to which more than one processor is allocated and allocate to newly arrived job.
  - In case if jobs request cannot be satisfied then it remains outstanding until processor becomes available or the job withdraws the request.
- After release of one or more processors
- Search the queue of requests that are not satisfied for processors. Allocate a single processor to each job in the list that are waiting new arrivals and currently has no processors. Then search the list again, allocating the remaining processors on an FCFS basis.

2.18 Examples on Uniprocessor Scheduling Algorithms

Example 2.18.1 **MU - Dec. 2014, 10 Marks**

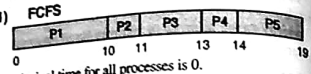
Consider the following set of processes, with the length of the CPU burst given in milliseconds:

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

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0. Draw four Gantt charts illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF nonpreemptive, priority (a smaller priority number implies a higher priority (a smaller priority number implies a higher priority) and RR (quantum = 1) and also calculate turnaround time and average waiting time.

Solution :

(1) FCFS

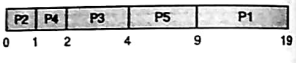


Arrival time for all processes is 0.

Process	Priority	Finish time	TAT	Waiting time
P1	3	10	10	0
P2	1	11	11	10
P3	3	13	13	11
P4	4	14	14	13
P5	2	19	19	14
		Average	13.4	9.6

(2) SJF (Nonpreemptive)

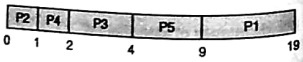
Arrival time for all processes is 0.



Process	Priority	Finish time	TAT	Waiting time
P1	3	19	19	9
P2	1	1	1	0
P3	3	4	4	2
P4	4	2	2	1
P5	2	9	9	4
		Average	7	3.2

(3) SJF (Preemptive)

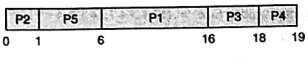
Since all processes arrive at same time 0, the solution is same as nonpreemptive SJF.



Process	Priority	Finish time	TAT	Waiting time
P1	3	19	19	9
P2	1	1	1	0

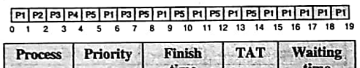
Process	Priority	Finish time	TAT	Waiting time
P3	3	4	4	2
P4	4	2	2	1
P5	2	9	9	4
		Average	7	3.2

(4) Priority Scheduling (Nonpreemptive)



Process	Priority	Finish time	TAT	Waiting time
P1	3	16	16	6
P2	1	1	1	0
P3	3	18	18	16
P4	4	19	19	18
P5	2	6	6	1
		Average	13.4	8.2

(5) Round Robin (time quantum = 1)



Process	Priority	Finish time	TAT	Waiting time
P1	3	19	19	9
P2	1	2	2	1
P3	3	7	7	5
P4	4	4	4	3
P5	2	14	14	9
		Average	9.2	5.4

Example 2.18.2

Calculate average waiting time and average turnaround time for the following situation. Assume quantum of 2 ms.

Process	Burst Time	Priority	Arrival time
P1	5	1	1
P2	7	3	5
P3	6	2	0

(i) Preemptive SJF

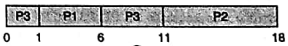
(ii) Preemptive priority

(iii) RR

Solution :

(i) Preemptive SJF

Following is the Gantt chart



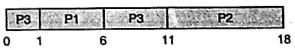
Process	Priority	TAT	Waiting time
P1	1	(6 – 1) = 5	(1 – 1) = 0
P2	3	(18 – 5) = 13	(11 – 5) = 6
P3	2	(11 – 0) = 11	(6 – 1) = 5
	Average	9.6	3.66

Average TAT = (5 + 13 + 11)/3 = 9.6

Average waiting time = (0 + 6 + 5) /3 = 3.66

(ii) Preemptive priority

Following is the Gantt chart



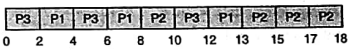
Process	Priority	TAT	Waiting time
P1	1	(6 – 1) = 5	(1 – 1) = 0
P2	3	(18 – 5) = 13	(11 – 5) = 6
P3	2	(11 – 0) = 11	(6 – 1) = 5
	Average	9.66	3.66

Average TAT = (5 + 13 + 11)/3 = 9.6

Average waiting time = (0 + 6 + 5)/3 = 3.66

(iii) RR

Following is the Gantt chart.



Process	Priority	TAT	Waiting time
P1	1	(13 – 1) = 12	(2 – 1) + (6 – 4) + (12 – 8) = 7
P2	3	(18 – 5) = 13	(8 – 5) + (13 – 10) = 6
P3	2	(12 – 0) = 12	(4 – 2) + (10 – 6) = 6
	Average	12.33	6.33

Average TAT = (12 + 13 + 12)/3 = 12.33

Average waiting time = (7 + 6 + 6)/3 = 6.33

Example 2.18.3

Consider the four processes P1, P2, P3 and P4 with length of CPU burst time. Find out average waiting time and average turnaround time for the following algorithm.

(i) FCFS

(ii) RR (slice = 4 ms)

(iii) SJF

Process	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Solution :

(i) FCFS

Following is the Gantt chart

Process	TAT	Waiting time
P1	(8 – 0) = 8	0
P2	(12 – 1) = 11	(8 – 1) = 7
P3	(21 – 2) = 19	(12 – 2) = 10
P4	(26 – 3) = 23	(21 – 3) = 18
Average	15.25	8.75

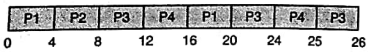
Average TAT = (8 + 11 + 19 + 23)/4

= 15.25 ms.

Average waiting time = (0 + 7 + 10 + 18)/4 = 8.75 ms.

(ii) RR (slice = 4 ms)

Following is the Gantt chart.



Process	TAT	Waiting time
P1	20	(16 – 4) = 12
P2	(8 – 1) = 7	(4 – 1) = 3
P3	(26 – 2) = 24	(8 – 2) + (20 – 12) + (25 – 24) = 15
P4	(25 – 3) = 22	(12 – 3) + (24 – 16) = 17
Average	18.25	11.75

Average TAT = (20 + 7 + 24 + 22)/4 = 18.25 ms.

Average waiting time = (12 + 3 + 15 + 17)/4 = 11.75 ms.

(iii) Preemptive SJF

Following is the Gantt chart.

Process	TAT	Waiting time
P1	(17 – 0) = 17	(10 – 1) = 9
P2	(5 – 1) = 4	(1 – 1) = 0
P3	(26 – 2) = 24	(17 – 2) = 15
P4	(10 – 3) = 7	(5 – 3) = 2
Average	13	6.5

Average TAT = (17 + 4 + 24 + 7)/4 = 13 ms.

Average waiting time = (9 + 0 + 15 + 2)/4 = 6.5 ms.

Example 2.18.4

Suppose that the following process arrive for execution at time indicate

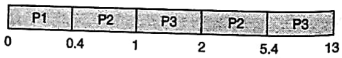
Process	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

Calculate average waiting time and average turnaround time using SRTF and SJF.

Solution :

(i) Preemptive SJF is SRTF (Shortest Remaining Time Next)

Following is the Gantt chart.



Process	TAT	Waiting time
P1	(13 – 0) = 13	(5.4 – 0.4) = 5
P2	(5.4 – 0.4) = 5	(2 – 1) = 1
P3	(2 – 1) = 1	(1 – 1) = 0
Average	6.33	2

Average TAT = (13 + 5 + 1)/3 = 6.33 ms.

Average waiting time = (5 + 1 + 0)/3 = 2 ms.

(ii) Non - preemptive SJF

Following is the Gantt chart.

Process	TAT	Waiting time
P1	(8 – 0) = 8	0
P2	(13 – 0.4) = 12.6	(9 – 0.4) = 8.6
P3	(9 – 1) = 8	(8 – 1) = 7
Average	9.53	5.2

Average TAT = (8 + 12.6 + 8)/3 = 9.53 ms.

Average waiting time = (0 + 8.6 + 7)/3 = 5.2 ms.

Example 2.18.5

Consider the following set of processes having their CPU burst time (in millisecond).

Process	CPU Burst time	Arrival time
P1	10	0
P2	5	1
P3	2	2

Calculate average waiting time using following CPU scheduling algorithms.

(1) FCFS (2) SJF

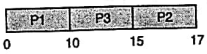
(3) Priority scheduling having priority range from 1 to 3, respectively for process P1 = 3, P2 = 2, P3 = 3 as given

(4) RR (slice = 2)

Solution :

(1) FCFS

Following is the Gantt chart.



Process	Waiting time
P1	0
P2	(10 – 1) = 9
P3	(15 – 2) = 13
Average	7.33

Average waiting time = (0 + 9 + 13)/3 = 7.33 ms.

(2) Preemptive SJF

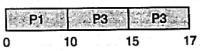
Following is the Gantt chart.

Process	Waiting time
P1	(8 – 1) = 7
P2	(4 – 2 – 1) = 1
P3	(2 – 2) = 0
Average	2.67

Average waiting time = (7 + 1 + 0)/3 = 2.67 ms.

(3) Priority scheduling having priority range from 1 to 3, respectively for process P1 = 3, P2 = 2, P3 = 3 as given.

Following is the Gantt chart.



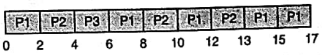
Process	Waiting time
P1	0
P2	(10 – 1) = 9
P3	(15 – 2) = 13
Average	7.33

Average waiting time = (0 + 9 + 13)/3

= 7.33 ms. (Same as FCFS)

(4) RR (slice = 2)

Following is the Gantt chart.



Process	Waiting time
P1	(6 – 2) + (10 – 8) + (13 – 12) = 7
P2	(2 – 0) + (8 – 4) + (12 – 10) = 7

Process	Waiting time
P3	(4 – 2) = 2
Average	5.33

Average waiting time = (7 + 7 + 2)/3 = 5.33 ms.

Example 2.18.6

Consider the following set of processes.

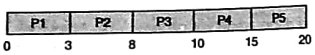
Process	CPU Burst time	Arrival time
P1	3	0
P2	5	1
P3	2	2
P4	5	3
P5	5	4

Calculate average waiting and turnaround time for each algorithm.

(1) FCFS (2) SJF (3) RR (slice = 2)

Solution :

(1) FCFS : Following is the Gantt chart.



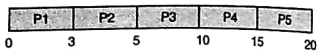
Process	TAT	Waiting time
P1	(3 – 3) = 0	0
P2	(8 – 1) = 7	(3 – 1) = 2
P3	(10 – 2) = 8	(8 – 2) = 6
P4	(15 – 3) = 12	(10 – 3) = 7
P5	(20 – 4) = 16	(15 – 4) = 11
Average	9.2	5.2

Average turnaround time = (0 + 7 + 8 + 12 + 16)/5
= 9.2 ms.

Average waiting time = (0 + 2 + 6 + 6 + 11)/5
= 5.2 ms.

(2) SJF

Following is the Gantt chart.



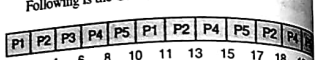
Process	TAT	Waiting time
P1	(3 – 3) = 0	0
P2	(10 – 1) = 9	(5 – 1) = 4
P3	(5 – 2) = 3	(3 – 2) = 1
P4	(15 – 3) = 12	(10 – 3) = 7
P5	(20 – 4) = 16	(15 – 4) = 11
Average	8.6	4.6

Average turnaround time = (0 + 9 + 3 + 12 + 16)/5
= 8.6 ms.

Average waiting time = (0 + 4 + 1 + 7 + 11)/5
= 4.6 ms.

(3) RR (slice = 2)

Following is the Gantt chart.



Process	TAT	Waiting time
P1	(11 – 0) = 11	(10 – 2) = 8
P2	(18 – 1) = 17	(2 – 1) + (11 – 4) + (17 – 13) =
P3	(6 – 2) = 4	(4 – 2) = 2
P4	(19 – 3) = 16	(6 – 1) + (13 – 8) + (18 – 15) =
P5	(20 – 4) = 16	(8 – 4) + (15 – 10) + (19 – 17) =
Average	12.8	8.8

Average turnaround time = (11 + 17 + 4 + 16 + 16)/5
= 12.8 ms.

Average waiting time = (8 + 12 + 2 + 11 + 11)/5
= 8.8 ms.

Example 2.18.7

Assume that you have following jobs to execute with o processor

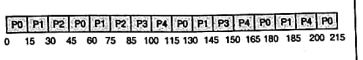
Job	CPU Burst time	Arrival time
0	75	0
1	50	10
2	25	10
3	20	80
4	45	85

Suppose system uses round robin with quantum of 15.

(i) Draw Gantt chart

(ii) Find average wait and turnaround time

Solution : (i) The Gantt chart



Process	TAT	Waiting time
P0	(215 – 0) = 215	(45 – 15) + (115 – 60) + (165 – 30) + (200 – 180) = 140
P1	(185 – 10) = 175	(15 – 10) + (60 – 30) + (130 – 75) + (180 – 145) = 125
P2	(85 – 10) = 75	(30 – 10) + (75 – 45) = 50
P3	(150 – 80) = 70	(85 – 80) + (145 – 100) = 50
P4	(200 – 85) = 115	(100 – 85) + (150 – 115) + (185 – 165) = 70
Average	130	87

(ii) Average turnaround time = (215 + 175 + 75 + 70 + 115)/5
= 130 ms.

Average waiting time = (140 + 125 + 50 + 50 + 70)/5
= 87 ms.

Example 2.18.8

Use following Scheduling algorithms to calculate ATAT and AWT for the following process

(i) FCFS

(ii) Preemptive and non-preemptive SJF

(iii) Preemptive priority.

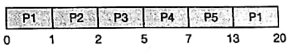
Process	Arrival time	Burst time	Priority
P1	0	8	3
P2	1	1	1
P3	2	3	2
P4	3	2	3
P5	4	6	4

Solution :

(i) FIFO

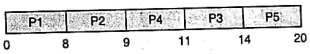
Process	Priority	Finish time	TAT	Waiting time
P1	3	8	(8 – 0) = 8	0
P2	1	9	(9 – 1) = 8	(8 – 1) = 7
P3	2	12	(12 – 2) = 10	(9 – 2) = 7
P4	3	14	(14 – 3) = 11	(12 – 3) = 9
P5	4	20	(20 – 4) = 16	(14 – 4) = 10
			Average	(53/5) = 10.6 (33/5) = 6.6

(ii) SJF(Preemtive)



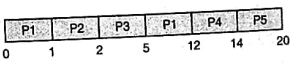
Process	Priority	Finish time	TAT	Waiting time
P1	3	20	(20 – 0) = 20	(13 – 1) = 12
P2	1	2	(2 – 1) = 1	(1 – 1) = 0
P3	2	5	(5 – 2) = 3	(2 – 2) = 0
P4	3	7	(7 – 3) = 4	(5 – 3) = 2
P5	4	13	(13 – 4) = 9	(7 – 4) = 3
			Average	(37/5) = 7.4 (17/5) = 3.4

(iii) SJF(Nonpreemtive)



Process	Priority	Finish time	TAT	Waiting time
P1	3	8	(8 – 0) = 8	0
P2	1	9	(9 – 1) = 8	(8 – 1) = 7
P3	2	12	(14 – 2) = 12	(11 – 2) = 9
P4	3	14	(11 – 3) = 8	(9 – 3) = 6
P5	4	20	(20 – 4) = 16	(14 – 4) = 10
			Average	(52/5) = 10.4 (32/5) = 6.4

(iv) Priority (Preemptive)



Process	Priority	Finish time	TAT	Waiting time
P1	3	20	(12 – 0) = 12	(5 – 1) = 4
P2	1	2	(2 – 1) = 1	(1 – 1) = 0
P3	2	5	(5 – 2) = 3	(2 – 2) = 0
P4	3	7	(14 – 3) = 11	(12 – 3) = 9
P5	4	13	(20 – 4) = 16	(14 – 4) = 10
Average			(43/5) = 8.6	(23/5) = 4.6

Example 2.18.9 MU - May 2016. 10 Marks

Find AWT, ATAT, ART and AWTAT for the following set of processes with CPU burst time in ms. Assume that all processes arrive at time 0.

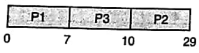
(P1-19), (P2-7), (P3-3)

i) FCFS with order P2,P3, P1

ii) Round Robin (Quantum = 2ms)

Solution :

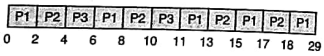
(i) FCFS : Following is Gantt chart,



Process	TAT	Waiting time
P1	(29 – 0) = 29	(10 - 0) = 10
P2	(7 - 0) = 7	0
P3	(10 – 0) = 10	(7 – 0) = 7
Average	46/3 = 15.33	17/3 = 5.66

(ii) Round Robin (RR) with quantum = 2

Following is Gantt chart,



Process	TAT	Waiting time
P1	(29 – 0) = 29	(6 - 2) + (11 - 8) +(15 - 13) +(18 - 17) = 10
P2	(18 - 0) = 18	(2 - 0) + (8 - 4) +(13 - 10) + (17 - 15) = 11
P3	(11 – 0) = 11	(4 – 0) + (10 - 6) = 8
Average	58/3 = 19.33	29/3 =9.66

Example 2.18.10 MU -Dec. 2016. 10 Marks

Assume the following processes arrive for execution at the time indicated and the length of cpu burst time given in millisecond

Job	Burst time	Priority	Arrival time
P_1	8	3	3
P_2	1	1	1
P_3	3	2	2
P_4	2	3	3
P_5	6	4	4

For the above process parameters, find average waiting times and average turnaround times for the following scheduling algorithms. First Come First Serve, Shortest Job First, non preemptive priority and Round Robin (assume quantum = 2 Units)

Solution :

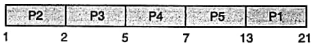
FCFS (First Solution)

Job	Priority	Arrival Time	Finish time	TAT	Waiting time
P1	3	3	13	(13-3) = 10	(5-3) = 2
P2	1	1	2	(2-1) = 1	(1-1) = 0
P3	2	2	5	(5-2) = 3	(2-2) = 0
P4	3	3	15	(15-3) = 12	(13-3) = 10
P5	4	4	21	(21-4) = 17	(15-4) = 11
Average				(43/5) =8.6	(23/5) = 4.6

FCFS (Second Solution)

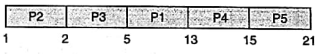
Job	Priority	Arrival Time	Finish time	TAT	Waiting time
P1	3	3	15	(15-3)=12	(7-3) =4
P2	1	1	2	(2-1)=1	(1-1) =0
P3	2	2	5	(5-2)=3	(2-2) =0
P4	3	3	7	(7-3)=4	(5-3) =2
P5	4	4	21	(21-4)=17	(15-4) =11
Average				(37/5) = 7.4	(17/5) =3.4

Non preemptive : SJF : (This solution is same for SJF preemptive)



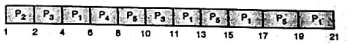
Job	Priority	Arrival Time	Finish time	TAT	Waiting time
P1	3	3	21	(21-3) =18	(13-3)= 10
P2	1	1	2	(2-1) = 1	(1-1) =0
P3	2	2	5	(5-2) = 3	(2-2) = 0
P4	3	3	7	(7-3) = 4	(5-3) = 2
P5	4	4	13	(13-4) = 9	(7-4) = 3
Average				(35/5) = 7	(15/5) =3

Non preemptive Priority : (Same as first solution of FCFS)



Job	Priority	Arrival Time	Finish time	TAT	Waiting time
P1	3	3	13	(13-3) = 10	(5-3) = 2
P2	1	1	2	(2-1) = 1	(1-1) = 0
P3	2	2	5	(5-2) = 3	(2-2) = 0
P4	3	3	15	(15-3) = 12	(13-3) = 10
P5	4	4	21	(21-4) = 17	(15-4) = 11
Average				(43/5) = 8.6	(23/5) = 4.6

Round Robin - quantum - 2 units



Job	Priority	Arrival Time	Finish time	TAT	Waiting time
P1	3	3	21	(21-)=18	(4-3)+(11-6)+(15-13)+(19-15) =12
P2	1	1	2	(2-1)=1	(1-1) =0
P3	2	2	11	(11-2)=9	(2-2)+(10-4) =6
P4	3	3	8	(8-3)=5	(6-3) =3
P5	4	4	19	(19-4)=15	(8-4)) + (13-10)+(17-15) =9
Average				(48/5) = 9.6	(30/5) = 6

Example 2.18.11

Consider following set of processes, with length of CPU burst given in milliseconds as follows

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

(i) Draw the Gantt chart for FCFS, SJF, Preemptive priority and RR (quantum 2)

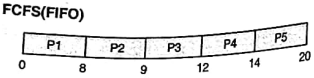
(ii) What is the turnaround time of each process for above algorithm?

(iii) What is waiting time of each process for each of the above algorithms?

(iv) Which algorithm results in minimum average waiting time?

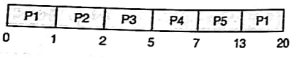
Solution :

FCFS(FIFO)



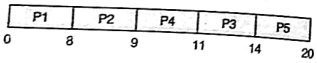
Process	Priority	Finish time	TAT	Waiting time
P1	3	8	(8 – 0) = 8	0
P2	1	9	(9 – 1) = 8	(8 – 1) = 7
P3	2	12	(12 – 2) = 10	(9 – 2) = 7
P4	3	14	(14 – 3) = 11	(12 – 3) = 9
P5	4	20	(20 – 4) = 16	(14 – 4) = 10
Average			(53/5) = 10.6	(33/5) = 6.6

SJF(Preemtive)



Process	Priority	Finish time	TAT	Waiting time
P1	3	20	(20 – 0) = 20	(13 – 1) = 12
P2	1	2	(2 – 1) = 1	(1 – 1) = 0
P3	2	5	(5 – 2) = 3	(2 – 2) = 0
P4	3	7	(7 – 3) = 4	(5 – 3) = 2
P5	4	13	(13 – 4) = 9	(7 – 4) = 3
Average			(37/5) = 7.4	(17/5) = 3.4

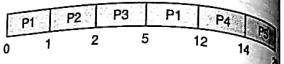
SJF(Nonpreemtive)



Process	Priority	Finish time	TAT	Waiting time
P1	3	8	(8 – 0) = 8	0
P2	1	9	(9 – 1) = 8	(8 – 1) = 7
P3	2	12	(14 – 2) = 12	(11 – 2) = 9
P4	3	14	(11 – 3) = 8	(9 – 3) = 6

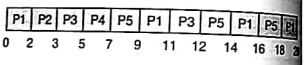
Process	Priority	Finish time	TAT	Waiting time
P5	4	20	(20 – 4) = 16	(14 – 4) = 10
Average			(52/5) = 10.4	(32/5) = 6.4

Priority (Preemptive)



Process	Priority	Finish time	TAT	Waiting time
P1	3	20	(12 – 0) = 12	(5 – 1) = 4
P2	1	2	(2 – 1) = 1	(1 – 1) = 0
P3	2	5	(5 – 2) = 3	(2 – 2) = 0
P4	3	7	(14 – 3) = 11	(12 – 3) = 9
P5	4	13	(20 – 4) = 16	(14 – 4) = 10
Average			(43/5) = 8.6	(23/5) = 6.6

RR (Quantum 2)



Process	Priority	Finish time	TAT	Waiting time
P1	3	20	(20 – 0) = 20	0+(9-2)+(14-11) +(18-16) = 12
P2	1	3	(3 – 1) = 2	(2-1) = 1
P3	2	12	(12– 2) = 10	(3-2)+(11-5) = 7
P4	3	7	(7 – 3) = 4	(5 –3) = 2
P5	4	18	(18 – 4) = 14	(7–4)+(12-9)+(16-13) = 8
Average			(50/5) = 10	(30/5) = 6

Preemptive SJF gives minimum Average waiting time.

Example 2.18.12 **MU - Nov 2015, 10 Marks**

Consider following set of processes with their CPU burst time.

Process	Burst time	Arrival time
P1	10	1
P2	4	2
P3	5	3
P4	3	4

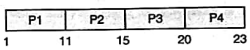
(i) Draw Gantt chart FCFS, SJF preemptive and round robin (quantum = 3).

Calculate average waiting time and average turnaround time.

(ii) Explain which scheduling policy adopted by Linux?

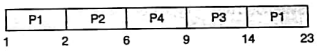
Solution . :

(i) FCFS Scheduling



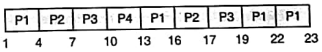
Process	TAT	Waiting time
P1	(11– 1)=10	0
P2	(15 – 2) = 13	(11 – 2) = 9
P3	(20 – 3) = 17	(15 – 3) = 12
P4	(23– 4) = 19	(20– 4) = 16
Average	59/4=14.75	37/4=9.25

(ii) SJF Preemptive Scheduling



Process	TAT	Waiting time
P1	(23– 1)=22	(14 – 2) = 12
P2	(6 – 2) = 4	0
P3	(14– 3) = 11	(9 – 3) = 6
P4	(9– 4) = 5	(6– 4) = 2
Average	42/4 =10. 5	20/4 = 5

(iii) RR Scheduling with quantum = 3



Process	TAT	Waiting time
P1	(23– 1)=22	(13 – 4) + (19 – 16) = 12
P2	(17 – 2) = 15	(4 – 2) + (16 – 7) = 11
P3	(19– 3) = 16	(7 – 3) + (17 – 10) = 11
P4	(13– 4) = 9	(10– 4) = 6
Average	62/4 =15. 5	40/4 = 10

Syllabus Topic : Linux Scheduling

2.19 Linux Scheduling

→ (Dec. 16)

Q. Write note on Scheduling in Linux system. **MU - Dec. 2016, 5 Marks**

- All the versions of Linux below version 2.5 was supporting a variant of the conventional UNIX scheduling algorithm. With this traditional UNIX scheduler, two types of trouble was :
  1. It does not give sufficient support for SMP systems.
  2. If number of tasks on the system grows, scaling was not supported.
- In version 2.5, Linux kernel have a scheduling algorithm that runs in O(1) time even with more number of tasks on the system.
- Enhanced symmetric multiprocessing together with processor affinity and load balancing is supported by the newly included scheduler. It also offer equality and support for interactive tasks.
- The Linux scheduler schedule the processes on the basis of priority assigned to them and it is a preemptive. following are the two categories of priorities defined for processes.
  - o Priorities from 0 to 99 and it is real time range..
  - o Priorities from 100 to 140 and called as nice value.
- Mapping of above two ranges are done in a global priority scheme. The lower integer values denote higher priorities and higher integer values denote the lower priorities.

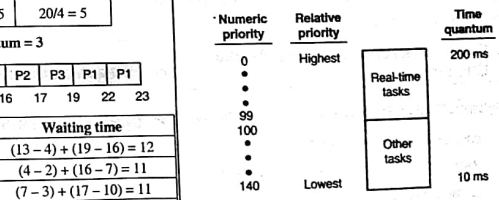


Fig. 2.19.1 : Relationship between priorities and time slice length

- To the higher-priority tasks longer time quantum is allocated and to the lower-priority tasks shorter time quantum is assigned.
- The relation between priorities and time-slice length is shown in Fig. 2.19.1.
- If the task is runnable then it is eligible for execution on the CPU if and only if it has time left over in its time slice.
- If a task has used up its time slice, it is assumed expired and is not eligible for execution again until all other tasks have also used up their time quanta.
- The kernel keeps a record of all runnable tasks in a data structure. As SMP is supported by kernel, each processor preserves its own run queue and schedules itself independently.
- Every run queue holds two priority arrays: the active array contains all tasks with time remaining in their time slices, and the expired array contains all expired tasks.
- Each of these priority arrays contains a list of tasks indexed according to priority.
- The scheduler selects the task with the highest priority from the active array for execution on the CPU. In multiprocessor systems, all processors are scheduling the highest-priority task from its own run queue structure.
- When all tasks have worn out their time slices (when active array is empty), the two priority arrays are exchanged; the expired array becomes the active array, and vice versa.
- Static priorities are assigned to real-time tasks. Remaining other tasks has dynamic priorities that are based on their *nice* values plus or minus the value 5.
- The interactivity of a task decides whether the value 5 will be added to or subtracted from the *nice* value.

2.20 Exam Pack (University and Review Questions)

Syllabus Topic : Process - Concept of Process

Q. What do you mean by process? *(Refer section 2.1.1)* (4 Marks) **(Dec. 2014)**

Q. Explain the concept of process. *(Refer section 2.1.1)*

Q. What is context switch ? *(Refer section 2.2)*

Syllabus Topic : Process States

Q. Draw process state transition diagram. *(Refer section 2.3)* (6 Marks) **(Dec 2014)**

Q. Draw and explain five state process model. *(Refer section 2.3)* (5 Marks) **(June 2015, Nov. 2015)**

Q. Draw and explain process state transition diagram. *(Refer section 2.3)*

Syllabus Topic : Process Control Block (PCB)

Q. Explain role of process control block. *(Refer section 2.5)* (5 Marks) **(Dec. 2014)**

Q. Write note on : Process Control Block. *(Refer section 2.5)* (5 Marks) **(Dec. 2016)**

Syllabus Topic : Operations on Processes

Q. What are the operations performed on process? Explain. *(Refer section 2.6)*

Syllabus Topic : Threads - Definition

Q. Differentiate the Process vs Thread *(Refer section 2.8)* (5 Marks) **(May 2016)**

Syllabus Topic : Types of Threads

Q. Explain different types of threads. *(Refer section 2.9)*

Q. Explain user level thread with advantages and disadvantages. *(Refer section 2.9.1)*

Q. Explain kernel level thread with advantages and disadvantages. *(Refer section 2.9.2)*

Syllabus Topic : Concept of Multithreading

Q. Explain multithreading models. *(Refer section 2.10)*

Syllabus Topic : Multicore Processors and Threads

Q. Explain Multicore Processors and Threads. *(Refer section 2.11)*

Syllabus Topic : Scheduling

Q. With help of neat diagram explain the functions of different types of schedulers. *(Refer section 2.12.1)*

Q. Compare the functions of different types of schedulers. *(Refer section 2.12.1(D))*

Q. Differentiate between long-term, short-term and medium-term scheduler. *(Refer section 2.12.1(D))*

Syllabus Topic : Types of Scheduling Preemptive and Non-preemptive

Q. Differentiate the Preemptive vs Non-Preemptive Scheduling. *(Refer section 2.12.2)* (10 Marks) **(May 2016)**

Syllabus Topic : Uniprocessor Scheduling

Q. Discuss various scheduling criteria. *(Refer section 2.13.1)* (5 Marks) **(May 2016)**

Q. What are the criteria for evaluation of scheduling algorithm performance? *(Refer section 2.13.1)*

Syllabus Topic : Scheduling Algorithms

Q. Explain different types of scheduling algorithms. *(Refer section 2.13.2)*

Syllabus Topic : Scheduling Algorithm - Priority Scheduling

Q. Explain priority scheduling algorithm. *(Refer section 2.13.2(C))*

Syllabus Topic : Scheduling Algorithm - Round Robin

Q. With the help of example, explain Round Robin Scheduling algorithm. *(Refer section 2.13.2(D))*

Syllabus Topic : Scheduling Algorithm Multilevel Queue Scheduling

Q. Explain multilevel queue scheduling algorithm. *(Refer section 2.13.2(E))*

Q. Explain multilevel feedback queue scheduling algorithm. *(Refer section 2.13.2(F))*

Syllabus Topic : Multiprocessor Scheduling

Q. Explain multiprocessor scheduling. *(Refer section 2.15)*

Q. What are general approaches used for scheduling of threads on multiprocessor system? Explain. *(Refer section 2.17)*

Example 2.18.1 (10 Marks) **(Dec. 2014)**

Example 2.18.9 (10 Marks) **(May 2016)**

Example 2.18.10 (10 Marks) **(Dec. 2016)**

Example 2.18.12 (10 Marks) **(Nov. 2015)**

Syllabus Topic : Linux Scheduling

Q. Write note on Scheduling in Linux system. *(Refer section 2.19)* (5 Marks) **(Dec. 2016)**

□□□

Module 3

CHAPTER 3

Synchronization and Deadlock

Syllabus Topics

Concurrency : Principles of Concurrency, Inter-Process Communication, Process/Thread Synchronization
Mutual Exclusion : Requirements, Hardware Support, Operating System Support (Semaphores and Mutex), Programming Language Support (Monitors), Classical synchronization problems: Readers/Writers Problem, Producer and Consumer problem.
Principles of Deadlock : Conditions and Resource Allocation Graphs, Deadlock Prevention, Deadlock Avoidance: Banker"s Algorithm for Single and Multiple Resources, Deadlock Detection and Recovery, Dining Philosophers Problem.

Syllabus Topic
Concurrency - Principles of Concurrency

3.1 Concurrency

3.1.1 Principles of Concurrency

→ (May 16)

Q. Explain the process synchronization in brief.
MU - May 2016. 5 Marks

- In a single-processor multiprogramming system, processes are not executed concurrently. In order to get the appearance of concurrent execution, a fixed time slot is allocated to each process.
- After utilization of this time slot, CPU gets allocated to other process. Such switching of CPU back and forth between processes is called as context switch.
- At a time single process gets executed so parallel processing cannot be accomplished.
- Also there is a definite amount of overhead drawn in switching back and forth between processes.
- Apart from above limitations, interleaved execution offers major benefits in processing efficiency and in program structuring.
- In a multiple processor system, interleaving and overlapping the execution of multiple processes is achievable.
- Both interleaving and overlapping correspond to basically diverse modes of execution and present diverse problems.
- In reality, both interleaving and overlapping can be treated as illustration of concurrent processing and both the techniques address the similar problems.
- The comparative speed of execution of processes depends on activities and behavior of other processes, how the operating system handles interrupts, and the scheduling policies of the operating system.

The difficulties that arise

1. Global resources sharing. For example, suppose two processes use the same global variable simultaneously and both carry out read and write operations on that variable, then various reads and writes execution ordering is serious.
2. Optimal management of the resources is not easy for the operating system.
3. Programming error tracing is not easy as results are typically non- deterministic and not reproducible.

- Only a single user is supported by single processor multiprogramming system at a time. While working on one application, user can switch to other application in this system.
- The keyboard for input purpose and screen for output purpose used by each application is same. The reason is, each application needs to use the procedure echo.

- As echo procedure is shared by each application, memory can be saved by keeping single copy of procedure in memory portion which is global for all sharing applications.

Example

```
void echo()
{
    chinput = getchar();
    choutput = chinput;
    putchar(choutput);
}
```

- In order to have efficient and close interaction among processes, sharing of primary memory among processes is useful. Consider the following sequence :
  1. Initially process P1 calls the echo procedure. The getchar() returns its value and stores it in input variable chinput. Process P1 gets interrupted straight away after returned value gets stored in chinput. At this instant, the most recently entered character, m, is stored in variable chinput.
  2. After P1 gets interrupted, there is turn of process P2 and it call up the echo procedure. Process P2 inputs and displays the single character n on the monitor.
  3. Now again, process P1 resumes its execution and the value m gets overwritten in the variable chinput and as a result it gets lost. The variable chinput holds value n, which is transferred to variable choutput and displayed.
  4. In this way, the first character m is lost and the second character n is displayed two times. All this happens due to shared global variable, chinput. If after updating the shared global variable, one process is interrupted, another process may modify the variable before the previously interrupted process can use its value. In above example both P1 and P2 are allowed to use shared global variable chinput.
- However, if only one process at a time is allowed to be in procedure, above discussed sequence would result in the following :
  (i) Initially process P1 calls the echo procedure. The getchar() returns its value and stores it in input variable chinput. Process P1 gets interrupted straight away after returned value gets stored in chinput. At this instant, the most recently entered character, m, is stored in variable chinput.
  (ii) After P1 gets interrupted, there is turn of process P2 and it call up the echo procedure. At this situation, process P1 is in suspended state and it is still inside the echo procedure. The process P2 is not allowed to enter inside the echo() procedure. Therefore, P2 is suspended until P1 comes out of the echo procedure.
  (iii) Later on, process P1 is resumes and finishes the execution of echo procedure. The output displayed is the character m.
  (iv) When P1 comes out of echo() procedure, P2 becomes active. Now process P2 can successfully call the echo() procedure.
- Therefore it is necessary to use shared global variable by only one process at a time.

Syllabus Topic : Interprocess Communication

3.2 Interprocess Communication

→ (May 16)

Q. Explain the Inter-process communication in brief.
MU - May 2016. 5 Marks

- Synchronization and communication are two basic requirements should be satisfied when processes communicate with each other.
- Synchronization of processes is required to achieve the mutual exclusion.
- Independent processes do not communicate with each other but cooperating processes may need to exchange information. Cooperative processes either communicates through shared memory or message passing.
- Cooperating processes require an Interprocess Communication (IPC) mechanism that will allow them to exchange data and information.
- There are two fundamental models of interprocess communication

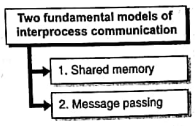


Fig. C3.1 : Two fundamental models of interprocess communication

➔ **1. Shared memory**

- In the shared-memory model, a region of memory that is shared by cooperating processes is established.
- Processes can then exchange information by reading and writing data to the shared region.
- In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.

➔ **2. Message Passing**

- Message passing provides both functions.
- Message passing has the further benefit that it lends itself to implementation in distributed systems as well as in shared-memory multiprocessor and uniprocessor systems.
- Following are the two primitives used in message passing :
  - o send (destination, message)
  - o receive (source, message)
- This is the minimum two operations required for processes to send and receive the messages.
- A process sends data in the form of a message to another process indicated by a destination.
- A process receives data by executing the receive primitive, indicating the source and the message.
- Communication by sending and receiving messages require synchronization. The receiver cannot receive a message until it has been sent by another process.
- The sending process is blocked until the message is received, or it is not after the send primitive executed by process.
- Similarly, when a process issues a receive primitive, there are two possibilities :
  - o Previously sent message is received and execution continues.
  - o If there is no waiting message, then either (a) the process is blocked until a message arrives, or (b) the process continues to execute, abandoning the attempt to receive.
- Thus, both the sender and receiver can be blocking or nonblocking. Three combinations are common, although any particular system will usually have only one or two combinations implemented:
  - o **The blocking send and blocking receive :** Until the message is handed over, the sender and receiver both gets blocked.
  - o **The nonblocking send and blocking receive :** After sending the message, the sender continues its work but receiver remains blocked until message is arrived to it. This permits a process to send one or more messages to a multiple destinations as quickly as possible. Here receiver is in need of message so that it can resume the execution. So it gets blocked until message arrives.
  - o **Nonblocking send, nonblocking receive :** Neither party is required to wait.
  - o **The nonblocking send and nonblocking receive :** Both sender and receiver will not wait and both will continue the work.
- Message passing system should give guarantee that messages will be correctly received by receiver. Receiver sends acknowledgement to sender after receiving the message.
- If acknowledgement not received in defined time then sender resend the message. It also offers authentication service.

Syllabus Topic
Process/Thread Synchronization

3.3 Process/Thread Synchronization

3.3.1 Critical Section Problem

➔ **(Dec. 14, June 15)**

Q. Explain critical section problem. **MU - Dec. 2014, 5 Marks**

Q. Explain critical section problem with its different solutions. **MU - June 2015, 10 Marks**

- As discussed previously, it is necessary to find out some means to disallow more than one process from reading and writing the shared data at the same time.
- The portion of the program where the shared memory is accessed is referred as the *Critical Section*.
- In order to avoid race conditions and faulty results, one must be able to recognize the codes in *Critical Sections* in each thread.
- The typical properties of the code that comprises *Critical Section* are as follows :

  - o These codes reference one or more variables in a "read-update-write" way. At the same time, any of those variables maybe changed by another thread.
  - o These codes modify one or more variables that can be referenced in "read-update-write" fashion by another thread.
  - o Any part of data structure used by the codes can be modified by another thread.
  - o Codes modify any part of a data structure that is currently in use by another thread.
- When one process is currently executing shared modifiable data in its critical section, no other process is to be permitted to execute in its critical section.
- Hence, the execution of critical sections by the processes is mutually exclusive in time.
- In a code called as critical section, only one process executes at a time.
- In a critical section problem, an algorithm needs to be designed which permits at most one process into the critical section at a time, with no deadlock.
- The critical section's problem solution must obey mutual exclusion, progress and bounded waiting. Any process directly cannot enter in critical section.
- First process has to obtain permission for its entry in its critical section. The segment of code which implements this appeal of process is the entry section. When process comes out of the critical section after completing its execution, there it has to execute exit section.
- The rest of the code is the remainder section.
- Any solution to the critical-section problem must satisfy the following three necessary conditions :

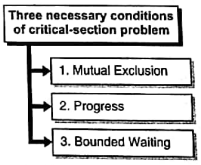


Fig. C3.2 : Necessary conditions for critical section problem

➔ **1. Mutual exclusion**

At a time, single process should be executing in critical section (CS). If currently Process P_i is executing in its CS then other processes should not execute in their critical sections.

➔ **2. Progress**

If process's CS is free, means it is not executing in its CS. In this situation suppose some processes desire to go into their critical sections for execution. In order to take the decision about who will now enter next in CS, only processes that are not executing in their remainder sections are allowed to take part to make this decision. This decision should be taken in some definite time and should not delayed indefinitely.

➔ **3. Bounded waiting**

A limit should be set on the number of times that other processes are permitted to go into their critical sections after a process has made a request to go into its critical section and before that request is approved.

- Semaphore and monitor are the solutions to achieve mutual exclusion.
- A synchronization variable taking positive integer values is called semaphore. Binary semaphore only has two values 0 or 1.
- Hardware does not supply the semaphore. Semaphore offer the solution to critical section problem. If semaphore variable takes value greater than 1 then it is called as counting semaphore.
- Like semaphore, a monitor also solves critical section problem. It is a software component which contains one or more procedures, an initialization sequence and local data. Following are the components of monitors :
  - o Shared data declaration
  - o Shared data initialization
  - o Operations on shared data
  - o Synchronization statement

3.3.2 Race Condition

Q. What is race condition? Explain with example.

- A race condition takes place when more than one process write data items so that the final result depends on the order of execution of instructions in the multiple processes.
- Consider two processes, P_1 and P_2, which share the global variable "x". At the time of execution, process P_1 updates "x" to the value 1 and at the time of execution of another process, P_2 updates "x" to the value 2.

- Thus, the two tasks are in a race to write variable "x". In this example the process that modifies x value lastly decides the final value of "x".

Role of operating system

1. The operating system should be capable of keeping track of the different processes.
2. Allocation and reclamation of various resources should be done by operating system.
3. Each process's data and physical resources should be protected by operating system against unintentional interfering by other processes.
4. The operations performed by the process and the output that it generates should not depend on the speed of execution compare to the speed of other concurrent processes.

Process Interaction can be defined as :

- o The processes not aware of each other
- o The processes in a straight way or indirectly aware of each other

- The conflict occurs between the concurrently executing processes, if these processes compete for the use of the same resource.
- More than one process may require the same resource while they are executing.
- The existence of the particular process is not known to other process. The competing processes do not pass the information to each other.

Syllabus Topic : Mutual Exclusion

3.4 Mutual Exclusion

→ (May 16)

Q. What is mutual exclusion ? Explain its significance. MU - May 2016. 5 Marks

- At the same time as one process executes the shared variable, all remaining processes wishing to accomplish so at the same instant should be kept waiting; when that process has over executing the shared variable, one of the processes waiting to perform so should be permitted to carry on.
- In this manner, each process executing the shared data (variables) keep outs all others from doing so at the same time. Only one process should be allowed to execute in critical section. This is called Mutual Exclusion. Race condition is then avoided due to mutual exclusion.
- The mutual exclusion is put into effect only if processes access shared changeable data – If during execution, the operations of the processes do not conflict with each other, they should be permitted to proceed in parallel.

Syllabus Topic : Mutual Exclusion - Requirements

3.4.1 Requirement of Mutual Exclusion

- It is obligatory to enforce a mutual exclusion: If many processes want to execute in critical section then only one process should be allowed to execute in that critical section among all processes that have critical sections for the same resource or shared object.
- If the process halts in its remainder section (other than critical section) then it is allowed to do so without interfering with other processes.
- If process is waiting to enter in critical section then it should not be delayed for an indefinite period: that is it should not lead to deadlock or starvation.
- If the critical section is empty (any process not executing in critical section), then if any process that requests entry to its critical section must be allowed to enter without delay.
- Assumptions about relative process speeds or number of processors are not made.
- Any process will not remain inside its critical section for indefinite time. It should stay there for a finite time only.

3.4.2 Mutual Exclusion Conditions

- A race condition can be avoided if we disallow to have two processes in their critical section at the same time.
- Following 4 conditions should hold to obtain a excellent solution for the critical section problem (mutual exclusion) :
  1. Two processes should not be inside their critical sections at the same time.
  2. Comparative speeds of processes or number of CPUs in the systems are not assumed.
  3. Process outside its critical section should not block other processes.
  4. Any process should not wait for longer amount of time arbitarily to enter its critical section.

Syllabus Topic : Hardware Support

3.5 Hardware Support

- The simple hardware instructions that are available on many systems can be used successfully in solving the critical section problem.
- Hardware features can make any programming task easier and get better system efficiency.
- If we restrict the interrupts from taking place while a shared variable was being modified, in uniprocessor environment the critical-section problem could be solved.
- Many systems have special instructions called Test and Set Lock called as TSL instruction. It is having format: "TSL ACC, X". In this instruction ACC is accumulator register and X is symbolic name of memory location. X holds character to be used as flag.
- TSL is indivisible instruction means that it cannot be interrupted in between. After instruction gets executed, following action take place.
  - o Content of X is copied to ACC
  - o Content of X becomes N
- During execution of above two steps, TSL instruction is not interrupted. If we assume value of X=N initially.
- The meaning of N is critical region is not free and F means it is free. Consider the following steps to enter in critical region.
  1. TSL ACC, X (Content of X is copied to ACC and Content of X becomes N)
  2. CMP ACC, "F (See if critical region is free)
  3. BU 1 (If critical region not free then loop back.)
  4. Return (Return to caller and enter)
- Following are the steps for exiting from critical region :
  1. MOV X, "F" (F gets copied to X)
  2. Return (Return to caller)
- Consider the two processes *Pi* and *Pj*.
- Assume initially scheduler schedules *Pi* and X = "F". So step 1 makes ACC= F and X= N. Here Pi after executing step 4, prepare to enter CR (critical region).
- *Pj* gets scheduled due to context switch before *Pi* enters the CR. *Pj* executes step 1 and X = N. In step 2, *Pj* fails the comparison and loop backs.
- It cannot execute step 3. So *Pj* cannot enter in CR as *Pi* is already in its CR.
- Consider *Pi* is again scheduled and executing in its CR. Even though context switch occurs and *Pj* gets scheduled at this moment it cannot enter in CR as ACC and X values are N (busy waiting).
- After completion of execution in CR, *Pi* executes step 5 to exit the CR ans X = "F'.
- If *Pj* scheduled after this, it execute step 1 and ACC becomes F as X was F in above step. Now due to step 1 by *Pj* ACC = "F" and X= "F".
- Step 2 is executed by *Pj* and succeeds and it enters the CR.
- This solution satisfies all conditions mutual exclusion, progress and bounded waiting.
- Since special hardware is required cannot be generalized to all machines. Also due to busy waiting, it is not efficient solution.

Syllabus Topic : Operating System Support (Semaphores and Mutex)

3.6 Operating System Support

3.6.1 Semaphores and Mutex

→ (June 15, Nov. 15)

Q. Give software approaches for mutual exclusion. MU - June 2015, Nov. 2015. 5 Marks

- E.W. Dijkstra (1965) abstracted the key notion of mutual exclusion in his concepts of semaphores.
- The solutions of the critical section problem represented in the section are not easy to generalize to more complex problems.
- To avoid this complicatedness, we can use a synchronization tool call a semaphore.
- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait and signal. These operations were firstly termed P (for wait) and V (for signal).

Definition

Q. What is semaphore ?

- A semaphore S is integer variable whose value can be accessed and changed only by two operations wait

(P or sleep or down) and Signal (V or wakeup or up). Wait and Signal are atomic operations.

- Binary Semaphores does not assume all the integer values.
- It assumes only two values 0 and 1. On the contrary, counting semaphores (general semaphores) can assume only nonnegative values.
- The wait operation on semaphores S, written as wait(S) or P(S), operates as follows :

```
wait(S): IF S > 0
         THEN S: = S - 1
         ELSE (wait on S)
```

The signal operation on semaphore S, written as signal(S) or V(S), operates as follows :

```
signal(S): IF (one or more process are waiting on S)
           THEN (let one of these processes proceed)
           ELSE S: = S +1
```

- The two operations, wait and signal are done as single indivisible atomic operation. It means, once a semaphore operation has initiated, no other process can access the semaphore until operation has finished. Mutual exclusion on the semaphore S is enforced within **wait(S)** and **signal(S)**.
- If many processes attempt a wait(S) at the same time, only one process will be permitted to proceed. The other processes will be kept waiting in queue. The implementation of wait and signal promises that processes will not undergo indefinite delay. Semaphores solve the lost-wakeup problem.

Disadvantages of Semaphore

Q. List the disadvantages of Semaphore.

1. Semaphores are fundamentally shared global variables.
2. From any location in a program it can be accessed during course of execution .
3. A lack of command over it or assurance of proper usage.
4. A lack of proper linking between the semaphore and the data to which the semaphore controls access.
5. They provide two functions, mutual exclusion and scheduling constraints

- **Mutex** is the concept related to binary semaphore. A key difference between the two is that the process that locks the mutex (sets the value to zero) must be the one to unlock it (sets the value to 1).
- In contrast, it is possible for one process to lock a binary semaphore and for another to unlock it.

3.7 Peterson Solution

Q. Explain Peterson solution to mutual exclusion.

- Peterson's solution involves only two processes. It gives a good means of algorithmic explanation of work out the critical-section problem and exemplifies some of the difficulties concerned in designing software that addresses the requirements of mutual exclusion, progress, and bounded waiting.
- The two processes *Pi and Pj* perform alternate execution between their critical sections and remainder sections (remaining code of the process).
- Peterson's solution requires the two processes to share two data items :

```
int x;
boolean y[2];
```

- The variable x decides who will enter in its critical section. if x==i, then process *Pi* is permissible to execute in its critical section.
- The y array is used to specify whether process *is ready* to enter its critical section.
- For example, if y[i] is true, this value indicates that *Pi* is ready to enter its critical section.
- In order to enter the critical section, process *Pi* first sets y[i] equal to true and then sets variable x to the value j. *Pi* sets x==j because, if the other (*Pj*) process desires to enter the critical section, it can do so.
- If both processes attempt to enter simultaneously, x will be set to both i and j at approximately the same time.
- Only one of these assignments will last; the other will occur but will be overwritten straight away.

```
do {
    y [i] = TRUE;
    x= j;
    while (y[j] && x== j);

  critical section

    y[i] = FALSE;

  remainder section

    } while (TRUE);
```


- The final value of x decides which of the two processes is permitted to enter its critical section first. We can prove that solution is correct by showing:
  - Mutual exclusion condition is satisfied.
  - The progress requirement is satisfied.
  - The bounded-waiting requirement also achieved.

Mutual exclusion

- Every *Pi* go into its critical section if and only if either y[j] == false or x==i. Also note that, if both *Pi* and *Pj* runs in critical sections simultaneously, then y[0] == y [1] ==true.
- It implies that *Pi* and *Pj* could not have successfully executed their while statements at about the same time, since the value of variable x can be either i or j but cannot be both.
- Hence, one of the processes say, *Pi* must have successfully executed the while statement, whereas *Pj;* had *to* execute at least one additional statement ("x== j").
- However, at that time, y [j] == true and x == j, and this condition will continue as long as *Pi* is in its critical section.
- Therefore we can conclude that mutual exclusion is preserved.

Progress and bounded waiting

- Process *Pi* can be prohibited from entering the critical section only if it gets stuck in the while loop with the condition y [j] ==true and x== j; this loop is the only one possible.
- If *Pj* is not ready to go into the critical section, then y [j] ==false, and *Pi* can go into its critical section.
- If *Pj* has set y[j] to true and is also running in its while statement, then either variable x == i or x == j.
- Depending on the value of x either *Pi* or *Pj* will enter in the critical section.
- If x == i, then *Pi* will enter the critical section. If x== j, then *Pj* will enter the critical section. However, once *Pj* exits its critical section, it will reset y[j] to false, allowing *Pi* to enter its critical section.
- If *Pj* resets y [j] to true, it must also set x to i.
- Thus, since *Pi* does not change the value of the variable x while executing the while statement, *Pi* will enter the critical section (progress) after at most one entry by *Pj* (bounded waiting).

Syllabus Topic
Programming Language Support (Monitors)

3.8 Programming Language Support (Monitors)

Q. How it is used to achieve mutual exclusion? Explain.

- If semaphores are used incorrectly, it can produce timing errors that are hard to detect. These errors occur only if some particular execution sequences results and these sequences do not always occur.
- In order to handle such errors, researchers have developed high-level language constructs called as monitor.
- A monitor is a set of procedures, variables, and data structures that are all grouped together in a particular type of module or package.
- Processes may call the procedures in a monitor if required, but direct access to the monitor's internal data structures from procedures declared outside the monitor is restricted to the processes.
- Following is the example of the monitor.

```
monitor example
integer i ;
condition c ;
procedure producer ( );
end ;
procedure consumer ( );
end;
end monitor;
```

- Monitors can achieve the mutual exclusion: only one process can be active in a monitor at a time.
- As monitors are a programming language construct, the compiler manages the calls to monitor procedures differently from other procedure calls.
- Normally, when a process calls a monitor procedure, if any other process is currently executing within the

monitor, it gets checked. If so, the calling process will be blocked until the other process has left the monitor.

- If no other process is using the monitor, the calling process may enter.
- The characteristics of a monitor are the following :
  - Only the monitor's procedures can access the local data variables. External procedures cannot access it.
  - A process enters the monitor by calling one of its procedures.
  - Only one process may be running in the monitor at a time; any other process that has called the monitor is blocked, waiting for the monitor to become available.
- Conditional variables are contained within monitors. These conditional variables can be accessed only within the monitors and used to achieve the synchronization.
- Functions cwait () and csignal() carry out the operation on conditional variable and these are treated as special data type in monitors.
  1. **cwait (c) :** On condition c, the execution of the calling process gets poised. After this other process can now access the monitor.
  2. **csignal (c) :** By executing cwait (c), process was blocked. Now this blocked process resumes its execution. Only one process will be selected to resume the execution in case if many such processes were blocked. No action will be taken If there is no such process.
- Monitor wait and signal operations are dissimilar from those for the semaphore. If a process in a monitor signals and no task is waiting on the condition variable, the signal is lost.
- When one process is executing in monitor, processes that trying to enter the monitor join a queue of processes blocked waiting for monitor availability.
- Within the monitor process may provisionally block itself on condition x by issuing cwait (x); After blocking, it then waits in a queue of processes to enter again the monitor when the condition alters, and resume execution at the point in its program following the cwait (x) call.
- If the execution of process going on in the monitor and during execution, it notices alter in condition variable x, it issues csignal (x), which signals the related condition queue that the condition has altered.
- It is possible for producer to put in characters to the buffer only by way of using procedure append within the monitor; the producer does not have straight access to buffer.
- This procedure initially makes sure the condition not *full* to conclude if there is space existing in the buffer. If space is not available then the process executing the monitor is blocked on that condition.

Syllabus Topic : Classical Synchronization Problems - Readers/Writers Problem

3.9 Classical Synchronization Problems

3.9.1 Readers/Writers Problem

→ (June 15)

Q. Explain how Readers/writers problem can be solved with semaphores? **MU - June 2015, 10 Marks**

- While defining the reader/writers problem, it is assumed that, many processes only read the file (readers) and many write to the file (writers).
- File is shared among a many number of processes. The conditions that must be satisfied are as follows :
  - Simultaneously reading of the file is allowed to many readers.
  - Writing to the file is allowed to only one writer at the same time.
  - Readers are not allowed to read the file while writer are writing to the file.
- In this solution, the first reader accesses the file by performing a down operation on the semaphore file. Other readers only increment a counter, read count. When readers finish the reading, counter is decremented.
- When last one ends by performing an up on the semaphore, permitting a blocked writer, if there is one, to write. Suppose that while a reader is reading file, another reader comes along.
- Since having two readers at the same time is not a trouble, the second and later readers can also be allowed if they come down.

- After this assumes that a writer wants to perform write on the file. The writer cannot be allowed to write the file, since writers must have restricted access, so the writer is suspended.
- The writer will suspended until no reader is reading the file. If a new reader arrives continuously with very short interval and perform reading, the writer will never obtain the access of the file.
- To stop this circumstance, the program is written in a different way: when a reader comes and at the same time a writer is waiting, the reader is suspended instead of being allowed reading immediately.
- Now writer will wait for readers that were reading and about to finish but does not have to wait for readers that came along after it.
- The drawback of this solution is that it achieves less concurrency and thus lower performance. Following is the solution given.

```
typedef int semaphore;
/* controls access to 'readcount' */
semaphore mutex = 1;
semaphore file = 1;/* controls access to file */
int readcount = 0;
void reader(void)
{
    while (TRUE) { /* repeat evermore */
    down(&mutex);
/* get exclusive access to 'readcountc' */
    readcountc = readcountc + 1;
/* one reader more now */
    if (readcount == 1) down(&file);
/* if this is the first reader... */
    up{&mutex};
/* release exclusive access to 'readcount' */
    read_file() /*read the file */
    down(&mutex);
/* get exclusive access to 'readcount' */
readcount = readcount - 1;
/* one reader fewer now */
if (readcount == 0) up(&db);
/* if this is the last reader... */
up(&mutex);
/* release exclusive access to 'readcount' */
use_data_read();/* noncritical region */
    }
}
void writer(void)
{
    while (TRUE) {/* repeat evermore */
    think_up_data(); /* noncritical region */
    down(&file);/* get exclusive access */
    write_file(); /* write the file */
    up(&file); /* release exclusive access */
    }
}
```

Syllabus Topic
Producer and Consumer Problem

3.9.2 Producer and Consumer Problem

→ (Dec. 16)

Q. Explain an algorithm for producer-consumer problem. **MU - Dec. 2016, 10 Marks**

- In producer-consumer problem assumes that buffer is bounded buffer. This means that there is a finite numbers of slots are available in a buffer. While producing the items, if the buffer is full the producer process should be suspended.
- In the same way, while consuming the items from buffer, if it becomes empty, consumer should be suspended. It is also necessary to ensure that only one process at a time manipulates a buffer to avoid race conditions or lost updates.
- Sleep-wake up system calls is used in this situation to avoid race condition. Producer process should go to sleep when buffer is full and consumer process should wake up when producer will put data in buffer
- In the same way, consumer goes to sleep until the producer puts some data in buffer and wakes up to consume this data.
- Here two processes, producer and consumer share a common, fixed-size (bounded) buffer. The producer puts data into the buffer and the consumer takes data out.
- Difficulty arises when the producer wants to put a new data in the buffer, but there is no space in buffer. That is, it is already full.
- Way out to this problem is that, producer goes to sleep and to be awakened when the consumer has removed data.

- It may also happen that, consumer wants to take out data from the buffer but buffer is already empty.
- Way out to this problem is that, consumer goes to sleep until the producer puts some data in buffer and wakes consumer up.

Code for the producer process

The code for the producer process can be modified as follows :

```
while (true)
{
/* produce an item in nextProduced variable */
while (counter == ARRAY_SIZE); /* do nothing */
array[in] = nextProduced;
in = (in + 1) % ARRAY-SIZE;
counter++;}
```

Code for the consumer process

The code for the consumer process can be modified as follows :

```
while (true)
{
while (counter == 0)
/* do nothing */
nextConsumed = array [out];
out = (out + 1) % ARRAY_SIZE;
counter--;
/* consume the item in nextConsumed variable*/}
```

- Producer and consumer functions are accurate if considered separately, they may not work correctly when executed concurrently.
- Race condition also occurs in this approach just like we came across in previous approaches.
- As constraint is not imposed on accessing the 'counter' variable, the race condition is occurred.

3.9.2(A) Producer Consumer Problem Using Semaphore

- In previous section, we have seen to producer-consumer problem. The Solution to producer-consumer problem uses three semaphores namely, full, empty and mutex.
- The semaphore 'full' is used for counting the number of slots in the buffer that are full.
- The 'empty' for counting the number of slots that are empty and semaphore 'mutex' to make sure that the producer and consumer do not access modifiable shared section of the buffer simultaneously.

Initialization

- Set full buffer slots to 0.
  i.e. semaphore Full = 0.
- Set empty buffer slots to N.
  i.e. semaphore empty = N.
- For control access to critical section set mutex to 1
  i.e. semaphore mutex = 1.

```
Producer ( )
WHILE (true)
    produce-Item ( );
    P (empty);
    P (mutex);
    enter-Item ( )
    V (mutex)
    V (full);
Consumer ( )
WHILE (true)
    P (full)
    P (mutex);
    remove-Item ( );
    V (mutex);
    V (empty);
    consume-Item (Item)
```

Busy Waiting → (Nov. 15, Dec. 16)

Q. What do you mean by busy waiting? What is wrong with it? **MU - Nov. 2015, Dec 2016. 5 Marks**

- Busy waiting is the limitation of semaphore definition. If critical section is not free (process is already executing in it) then other process trying to enter in critical section should loop continuously in entry code.
- This busy waiting (continual looping) is the problem as far as multiprogramming environment is considered.
- In this case a single CPU is shared among many processes. This busy waiting waste CPU cycles which would have been used by other processed effectively.

Syllabus Topic : Principles of Deadlock

3.10 Principles of Deadlock

3.10.1 Deadlock Problem

→ (June 15, Nov. 15, May 16, Dec. 16)

Q. What is deadlock ? **MU - June 2015, Nov. 2015, May 2016, Dec. 2016. 2 Marks**

- We know that processes needs different resources in order to complete the execution.
- So in a multiprogramming environment, many processes may compete for a multiple number of resources.
- In a system, resources are finite. So with finite number of resources, it is not possible to fulfill the resource request of all processes.
- When a process requests a resource and if the resource is not available at that time, the process enters a wait state. In a multiprogramming environment it may happen with many processes.
- There is chance that waiting processes will remain in same state and will never again change state.
- It is because the resources they have requested are held by other waiting processes. When such type of situation occurs then it is called as deadlock.
- If there are 5 same type of resources are available in system then we say that there are 5 instances of this resource type.
- Suppose the process request the instance of resource. If any instance of the resource type is allocated to it, then request will be fulfilled.
- If it is not possible to fulfill the request, then the instances are not identical, and the resource type classes have not been defined appropriately.
- As a rule, first process must request a resource and then use it. After the use of this resource, it should release it. A process may request any number of resources which are necessary to complete its execution.

Following are the sequence when a process may utilize a resource :

1. **Request :** If it is not possible to fulfill the request instantly, then the requesting process must wait until it can get the resource.
2. **Use :** The process use the resource to accomplish the task.
3. **Release :** The process free the resource after use

Syllabus Topic
Principles of Deadlock : Conditions

3.10.2 Conditions

In deadlock, processes never finish execution as resources are held up by other processes. Due to this unavailability new processes are prevented from starting the execution.

Necessary Conditions

→ (June 15, Nov. 15, May 16, Dec. 16)

Q. Explain necessary and sufficient condition for deadlock to occur. **MU - June 2015, Nov. 2015, May 2016, Dec. 2016. 4 Marks**

If the following four conditions hold simultaneously in a system then a deadlock situation can occur in the system. These are shown in Fig. C3.3.

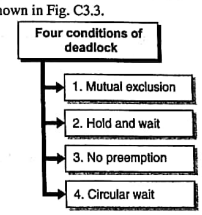


Fig. C3.3 : Necessary conditions for deadlock

→ **1. Mutual exclusion**

This condition ensure that, resource should be used by single process at a time and it remains with using process in non-sharable mode. If same resource is needed by another process then the requesting process must be postponed to use this resource. Requesting process will get the resource after released by holding process.

→ **2. Hold and wait**

The process holds minimum one resource and waits to obtain remaining needed resources that are at present being held by other processes.

→ **3. No preemption**

Preemption of the resource will not be done; Process holding the resource, releases it on its own after the completion of the its chosen task.

4. Circular wait

In this condition, a collection of waiting processes say{P0, P1, ..., Pn } exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ..., Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

Syllabus Topic
Resource Allocation Graphs

3.10.3 Resource Allocation Graphs

→ (Dec. 16)

Q. Write note on Resource Allocation Graph.
MU - Dec. 2016. 5 Marks

- A directed graph called a system resource-allocation graph can explain the deadlock precisely.
- In this graph the set of vertices V is partitioned into two different types of nodes. These are,
  - o The set consisting of all the active processes in the system,
    say P = { P1, P2, ... Pn}
  - o The set consisting of all resource types in the system,

  say R ={ R1, R2, ..., Rn}
- In this graph, If there is a edge with direction from process P to instance of resource type R then it means that P has made request for an instance of R and now waiting for that resource. If there is edge with direction from resource type R to process P then it means that an instance of resource type R has been allocated to process P.
- So, edge from process P to resource R is request edge and edge from resource R to process P is assignment edge. Circle nodes in graph represent processes and square represents resources.
- If resource allocation graph contains cycle then deadlock may exist in the system. If it does not contain cycle, then processes are not deadlocked.
- If system contains single instance of all the resource types, then all processes involved in cycle is deadlocked.
- So in this case, a cycle in resource allocation graph is both necessary and sufficient condition for existence of deadlock. Dot inside square represents single instance of that resource.
- Cycle in following graph shows the deadlock situation when resource instances are single. P1 has requested the resource held by P2 and P2 request the resource held by P1. P3 has requested resource R3.

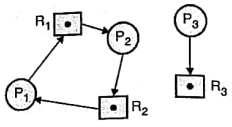


Fig. 3.10.1: Resource Allocation Graph

- If more than one instances of each resource type is present in resource allocation graph then a cycle indicates that a deadlock has occurred. In this case, a cycle is a necessary condition for the existence of deadlock but it is not sufficient for the existence of deadlock.
- The reason behind cycle not being the sufficient condition is that, the process can release the held resource although cycle exists.
- This released resource then can be obtained by requesting process.
- Consequently cycle can be broken. Hence, if cycle is not present in resource allocation graph then system is free from deadlock state. In case if cycle exists, then system may or may not be in a deadlocked state.

Syllabus Topic : Deadlock Prevention

3.11 Deadlock Prevention

→ (Dec. 14, June 15, Nov. 15)

Q. What is difference between deadlock prevention and avoidance algorithms.
MU - Dec. 2014, Nov. 2015. 5 Marks

Q. Explain deadlock prevention.
MU - June 2015. 2 Marks

- As described above, for a deadlock to occur, each of the four necessary-conditions: mutual exclusion, hold and wait, no preemption and circular wait should hold simultaneously in the system.
- By ensuring that at least on one these conditions cannot hold, we can prevent the occurrence of a deadlock.
- So deadlock prevention can be achieved by denying the holding of at least one of the following four conditions.

1. Mutual exclusion	2. Hold and wait
3. No preemption	4. Circular wait

The approach is explained as below.

1. Mutual exclusion

- Non-sharable resources cannot be shared by many processes simultaneously.
- If resources are non-sharable then mutual-exclusion condition must hold for them.
- For example, It is not possible to use printer by many processes at the same time. Printer is non-sharable but read only file is sharable resource and do not require mutually exclusive access.
- Therefore sharable resource such as read only file cannot be involved in a deadlock.
- We cannot prevent deadlock by denying mutual exclusion as some resources are inherently non-sharable such as printer.

2. Hold and wait

- Here we have to see that hold and wait will never occur. There should be an assurance that, a process should requests a resource only if it does not hold any other resource. Following two protocols can be used.
  - o Allocate all the resources needed by process before it starts the execution. This protocol will keep the allocated resources unused. The reason is that, even though some of the resources are needed by process in the last stage of execution, it will hold it from start of execution.
  - o Second protocol allows the process to request the resources only when it has released the previously allocated resources. Again the disadvantage of first protocol appears here. Suppose the important resource for which many processes compete is needed only at the start and end of the execution by this process.
  - o If initially process release resource and at the end request for the same, then it may not possible to get the same resource immediately. So we must request all resources at the beginning.
- Apart from the limitation of the above two protocols, other limitation can be starvation.
- In this, processes will keep significant and popular resources with them forcing to wait other processes indefinitely which require these resources.

3. No Preemption

We have to ensure that no preemption condition does not hold. Following two protocols can be used.

- If a process at present keep some resources with it and requests another resource that cannot be immediately allocated to it, then all resources at present being held should be preempted.
- The list of the resources for which process is waiting is maintained. The preempted resources from the process are included in this list.
- The process now restart the execution provided that it can get back its old preempted resources, in addition to the new ones that it is requesting.
- If the needed resources by the process are available, then allocate to the process requesting for it. On the other hand if they are not available, then check whether it is allocated to the process which is further waiting for additional resources. If it is the case then withdraw the resources from waiting process and allot them to requesting process.
- The process which request the resources should wait in case if resources are neither available nor held by waiting process.
- In this situation, if other process make a request for the resource held by waiting process, then it should be preempted from waiting process.

4. Circular wait

- To prevent the occurring of circular-wait condition inflict a total ordering of all resource types. In this case each process should request the resources in an increasing order of assigned numbers to the resources.
- Consider the collection of resource types R = {R1, R2, ..., Rn}. To this each resource type, a unique integer number is assigned.
- As resources are now recognized by their numbers, it is easy to compare them. It is also easy to know if one resource precedes the other in our ordering.
- Assignment of numbers to resources is a one-to-one function F: R → N, where N is the set of natural numbers. In order to prevent the deadlock, following two protocols can be used.
  - o Each process requests resources in an increasing order of assigned numbers to the resources.

- Whenever process request instance of resource type Rj, it should release any resource Ri such that F(Ri) > = F(Rj).

– If these two protocols used, circular wait condition cannot hold. If process Pi waits for resource Ri which is held by process Pi+1 we get F(Ri) < F(Ri+1) for all i.

– It implies that F(R0) < F(R1) <F(R2) < ..... F(Rn) < F(R0). By transitive relation, we get contradictory statement F(R0) < F(R0). circular wait does not hold.

Syllabus Topic : Deadlock Avoidance

3.12 Deadlock Avoidance

→ (Dec. 14, June 15, Nov. 15)

Q. What is difference between deadlock prevention and avoidance algorithms
MU - Dec. 2014, Nov. 2015, 5 Marks

Q. Explain deadlock avoidance, prevention and detection.
MU - June 2015, 2 Marks

– Deadlock avoidance requires that the system has some information available up front. Initially every process declares the maximum number of resources need which it may require.

– An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested.

– If the sequence in which process will request the resource and will release it known additionally, then we can decide for each request whether or not the process should wait.

– To avoid the circular wait condition completely, a deadlock-avoidance algorithm inspects the resource-allocation state in dynamic way.

– Following are the factors which define and describe resource allocation state.

- The number of available resources in the system.
- The number of allocated resources in the system.
- The maximum demands of the processes.

– Dijkstra's Banker's algorithm is used for deadlock avoidance. The algorithm requires prior knowledge of number of resources each process needs.

– After that, the OS act like a sincere small-town banker. OS will allocate the resources to processes only it has sufficient number of resources available to fulfill possible demand. This is considered as a safe state.

Safe State

– If the system is able to allocate resources to each process up to its maximum need in some order and still avoid a deadlock then the system is in safe state.

– If the safe sequence of all processes present in system, the system is considered to be in safe state. Sequence < P1, P2, ...Pn > is considered as a safe sequence for the present allocation state if, for each process Px, the resources which Px can still request can be fulfilled by,

- The at present available resources in the system plus
- The resources held by all of the processes Py's, where y < x.

– If process Pi cannot get all the needed resources as they are not available, it should wait until Pj complete the execution and frees the resources.

– If the system is in a safe state, there can be no deadlock. If the system is in an unsafe state, there is the possibility of deadlock.

Example

Consider a system with 10 disk drives and 3 processes (P0, P1, and P2)

Process	Maximum needs	Current needs
P0	8	4
P1	4	2
P2	7	2

– Initially 10 disk drives are available in the system. The sequence <P1, P0, P2> is safe sequence. Process P0 requires 8 disk drives, process P1 requires 4 disk drives and P2 may require up to 7 disk drives.

– At time t1, process P0 is holding 4 disk drives, process P1 is holding 2 disk drives and process P2 is holding 2 disk drives. (Thus in the system 2 disk drives are free.)

– At time to, the system is in safe state. The sequence <P1, P0, P2> satisfies safety condition. P1 can immediately be allocated its disk drives and returns them after completion. Now total 4 disk drives are available.

– P0 gets all the disk drives. After completion, it returns them and now total 8 disk drives are available in the system.

– Finally P2 can be allocated all the needed disk drives and returns them after completion. (The system will have now all 10 disk drives available). Fig. 3.12.1 shows safe, unsafe and deadlock state spaces.

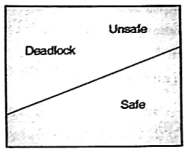


Fig. 3.12.1 : Safe, unsafe, and deadlock state spaces

– At time t1, if any process requests the additional resource say disk drives in our example and if after its allocation other process could not be allocated needed resource type due to unavailability, then system may go in unsafe state.

3.12.1 Deadlock Avoidance Algorithms

→ (May 16, Dec. 16)

Q. Suggest techniques to avoid deadlock.
MU - May 2016, Dec. 2016, 1 Mark

Following algorithms are used in the avoidance of deadlock :

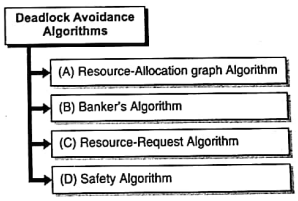


Fig. C3.4 : Deadlock avoidance algorithms

→ 3.12.1(A) Resource-Allocation Graph Algorithm

– If system has resource allocation system with only one instance of each resource type, then only this algorithm is applicable.

– Claim edge (dotted edge) in resource allocation graph is like a future request edge.

– When a process requests a resource, the claim edge is converted to a request edge.

– When a process releases a resource, the assignment edge is converted to a claim edge.

– The request can be granted by converting request edge to assignment edge. If assignment does not convert into cycle in resource allocation graph then system is in safe state.

– After assignment of resource, if cycle detection algorithm detects cycle, then system is in unsafe state. The unsafe state in resource allocation graph is shown in Fig. 3.12.2.

Cycle detection : $O(n^2)$

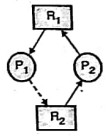


Fig. 3.12.2 : Resource-allocation graph for deadlock avoidance

Syllabus Topic : Banker's Algorithm for Single and Multiple Resources

→ 3.12.1(B) Banker's Algorithm

→ (June 15, Dec. 16)

Q. Explain data structures used in bankar's algorithm.
MU - June 2015, 5 Marks

Q. Explain the banker's algorithm in detail.
MU - Dec. 2016, 10 Marks

– It is applicable to the resource allocation system with multiple instances of each resource type.

– It is less efficient than resource-allocation graph algorithm.

– Newly entered process should declare maximum number of instances of each resource type which it may require.

– The request should not be more than total number of resources in the system.

– System checks if allocation of requested resources will leave the system in safe state. If it will the requested resources are allocated.

– If system determines that resources cannot be allocated as it will go in unsafe state, the requesting process should wait until other process free the resources.

- Following data structures are used in Banker's algorithm :
  - **A[m] :** Array A of size m shows the number of available resources.
  - **M[n][m] :** Two dimensional array M shows maximum requirement of the resources by each process. M[i][j] = k indicates process Pi can request at the most k instances of resource type Rj.
  - **C[n][m] :** Two dimensional array C shows current allocation status of resources to each process. C[i][j] = k indicates process Pi allocated k instances of resource type Rj.
  - **N[n][m] :** Two dimensional array N shows the remaining possible need of each process. (i.e., N[i][j] = M[i][j] - C[i][j]). N[i][j] = k indicates process Pi may need additional k instances of resource type Rj to accomplish the execution.

➜ 3.12.1(C) Resource-Request Algorithm

Q. Explain resource-request algorithm.

- This algorithm decides whether requests of the resources can be safely granted so as to system will remain in safe state.
- Let R[n][m] be the two dimensional array, describe the current outstanding requests of all processes. R[i][j] = k indicates process Pi want k instances of resource type Rj. to accomplish the execution.

1. If R[i][j] <= N[i][j], to step 2; otherwise, raise an error condition as process is requesting more resources than it needs.
2. If R[i][j] > A[j], then the process should wait as needed resources are not available.
3. Otherwise, go to step 4
4. Allocate the requested resources to Pi. The state is altered as

$$A[j] = A[j] - R[i][j]$$
$$C[i][j] = C[i][j] + R[i][j]$$
$$N[i][j] = N[i][j] - R[i][j]$$

- Once the resources are allocated, check to see if the system state is safe. If unsafe, the process must wait and the old resource-allocated state is restored.

➜ 3.12.1(D) Safety Algorithm

This algorithm find out whether system is in safe state or not.

1. Let W be an integer array of length m, initialized to array A (Available number of resources). Let F be a boolean array of length n, initialized to false (determines if process finished or not).
2. Find i such that both :
   F[i] == false
   N[i] <= W
   If no such i exists, go to step 4
3. W = W + C[i];
   F[i] = true;
   Go to step 2
4. If F[i] == true for all i, then the system is in a safe state, otherwise unsafe.

Run-time complexity : $O(m \times n^2)$

- To demonstrate the working of banker's algorithm, consider the following snapshot of the system at time to processes are numbered P0 to P4 (total 5 processes).
- Resources are P, Q, and R (total 3 resources). Resource type P has 12 instances, resource type Q has 9 instances, and resource type R has 6 instances.

Process	C(Allocation)	M(Maximum requirement)	N (Remaining need)	A(Available resources)
	P, Q, R	P, Q, R	P, Q, R	P, Q, R
P0	3, 0, 1	5, 2, 1	2, 2, 0	1, 6, 2
P1	0, 2, 1	3, 2, 2	3, 0, 1	
P2	2, 0, 0	3, 2, 2	1, 2, 2	
P3	2, 1, 0	7, 3, 1	5, 2, 1	
P4	4, 0, 2	5, 0, 3	1, 0, 1	

The system currently with sequence <P2, P1, P0, P4, P3> is in safe state. The sequence <P2, P1, P0, P4, P3> satisfies the safety criteria as shown.

1. Process P2

N <= A (1, 2, 2) <= (1, 6, 2) is true

A = A – N (1, 6, 2) – (1, 2, 2) = (0, 4, 0)

A = A + C (0, 4, 0) + (3, 2, 2) = (3, 6, 2) are available in the system after P2 finishes.

2. Process P1

N <= A (3, 0, 1) <= (3, 6, 2) is true

A = A – N (3, 6, 2) – (3, 0, 1) = (0, 6, 1)

A = A + C (0, 6, 1) + (3, 2, 2) = (3, 8, 3) are available in the system after P1 finishes.

3. Process P0

N <= A (2, 2, 0) <= (3, 8, 3) is true

A = A – N (3, 8, 3) – (2, 2, 0) = (1, 6, 3)

A = A + C (1, 6, 3) + (5, 2, 1) = (6, 8, 4) are available in the system after P0 finishes.

4. Process P4

N <= A (1, 0, 1) <= (6, 8, 4) is true

A = A – N(6, 8, 4) – (1, 0, 1) = (5, 8, 3)

A = A + C(5, 8, 3) + (5, 0, 3) = (10, 8, 6) are available in the system after P0 finishes.

5. Process P3

N <= A (5, 2, 1) <= (10, 8, 6) is true

A = A – N(10, 8, 6) - (5, 2, 1) = (5, 6, 5)

A = A + C(5, 6, 5) + (7, 3, 1) = (12, 9, 6) are available in the system after P0 finishes.

- So the required resources can be allocated to the processes in the sequence <P2, P1, P0, P4, P3> and it is a safe sequence. Suppose at time t1, P2 request one additional instance of resource type P and two instances of resource type Q, so Request R2 = (1, 2, 0) and it is not greater than available resources A.
- So step 4 of the resource request algorithm indicates that request can be allocated.
- After allocation of these resources, now safety algorithm is executed to determine whether system remains in safe state. If any such sequence exists, then system will be in safe state.

Example 3.12.1 MU - Dec. 2014, 10 Marks

Consider the following snapshot of a system :

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

With reference to banker's algorithm

(a) Find need matrix

(b) is the system in a safe state?

(c) If a request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately ?

Solution :

(a) Need matrix

	Need			
	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

(b) System is in safe state because many safe sequences exist such as (P0; P2; P1; P3; P4). As available is (1, 5, 2, 0), either process *P0* or *P3* can complete the execution. Once process *P3* finishes, it releases its resources which permit all other existing processes to execute.

(c) Yes, request can be granted immediately. After granting this, *Available* will become (1, 1, 0, 0).One of the safe sequence is (*P0*, *P2*, *P3*, *P1*, and *P4*). Allocation for P1 becomes (1, 4, 2, 0) and Need for P1 becomes (0, 3, 3, 0).

Example 3.12.2

Consider the following snapshot of the system.

Process	Allocation A B C	Maximum Requirement A B C
P0	0 0 1	1 1 1
P1	2 0 0	3 2 3
P2	1 3 2	4 3 1
P3	1 0 1	0 0 1
P4	0 0 1	3 2 1

Let ABC be the resources with instances of A is 7, B is 3 and C has 6 instances.

(i) What is the content of matrix need ?

(ii) Is the system in a safe state? Prove it.

Solution :

(i) Need = Max – Allocation

A	B	C
1	1	0
1	2	3

A	B	C
3	0	1
–1	0	0
3	2	0

(ii) A = 7 , B = 3 and C = 6 are total instances of resources.

By adding the Columns of Allocation matrix

We get, A = 4, B = 3 and C = 5.

Now Available number of resources
= Total – Allocation

Resource A available = 7 – 4 = 3

Resource B available = 3 – 3 = 0

Resource C available = 6 – 5 = 1

By using above available number of resources and applying bankers algorithm, sequence

<P2, P3, P4, P0, P1> is safe sequence.

Hence, system is in safe state.

Example 3.12.3

Consider the system with 5 processes, P0 to P4 and resource type A, B, C. Answer the following questions.

Process	Allocation A B C	Request A B C	Available A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

Let ABC be the resources with instances of A is 7, B is 3 and C has 6 instances.

(i) Is the system in a safe state?

(ii) If at time t1, P2 makes additional request for an instance of type C, Check whether system is in deadlock or Not at time t_1.

Solution :

(i) Request [P0]<= Available (0 0 0 <= 0 0 0)

Hence request of P0 is satisfied. P0 free the resources (0 1 0) and Available becomes (0 1 0)

After P0, Let the P2 request for resources.

Request [P0]<= Available (0 0 0 <= 0 1 0)

Hence request of P2 is satisfied. P2 free the resources (3 0 3) and Available becomes (3 1 3).

Available = Available + Allocation [P2].

In this way we can find that P3, P1 and finally P4 gets all their requested resources and safe sequence exist < P0, P2, P3, P1, P4>. Therefore system is in safe state.

(ii) If at time t_1, P2 makes additional request for an instance of type C then Request [P0]< = Available (0 0 1 < = 0 1 0) is false and resource C cannot be granted to P2. System will be in deadlock state.

Example 3.12.4

It is proposed to use bankers algorithm for handling deadlock. Total number of resources available for allocation is 7, 7 and 10 respectively. The current resource allocation state is shown as below.

Process	Allocation R1 R2 R3	Max R1 R2 R3	Available R1 R2 R3
P1	2 2 3	3 6 8	7 7 10
P2	2 0 3	4 3 3	
P3	1 2 4	3 4 4	

(i) Is the current allocation in a safe state?

(ii) Would the following request be granted in current state?

* P1 request (1, 1, 0)

Solution :

(i) Consider sequence <P1, P2, P3>

Total Allocated resources are (5, 4, 10). It is obtained by adding Allocation column.

Request[P1] = Max[P1] – Allocation[P1] = (1,4,5)

Now available in system are (7, 7, 10)

The request of P2 satisfied as available in system now (7,7,10) and then of P3 is satisfied, therefore system is in safe state.

(ii) Request of P1 is satisfied as (1,1,0) < = (7, 7, 10)

Example 3.12.5

Consider following snapshot of the system

Using Banker's algorithm answer the following questions-

(i) How many resources of type A B C D are there?

(ii) What are the content of need matrix ?

(iii) Find if system is in safe state? If it is, find the safe sequence.

Process	Max A B C D	Allocation A B C D	Available A B C D
P0	6 0 1 2	4 0 0 1	3 2 1 1
P1	1 7 5 0	1 1 0 0	

Process	Max A B C D	Allocation A B C D	Available A B C D
P2	2 3 5 6	1 2 5 4	
P3	1 6 5 3	0 6 3 3	
P4	1 6 5 6	0 2 1 2	

Solution :

(i) A-9 ; B-13 ; C-10 ; D-11

(ii) Need[i, j] = Max[i,j] – Allocation[i,j] so content of Need matrix is

	A	B	C	D
P0	2	0	1	1
P1	0	6	5	0
P2	1	1	0	2
P3	1	0	2	0
P4	1	4	4	4

(iii) The system is in a safe state as the processes can be finished in the sequence P0, P2, P4, P1 and P3.

Example 3.12.6 MU - June 2015. 10 Marks

Consider the following snapshot of the system

	Allocation A B C	Max A B C	Available A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Answer following questions using banker's algorithm

(a) What is the content of need matrix?

(b) Is the system in a safe state?

(c) If a request from process P1 arrives for (1, 0, 2), can the request be granted immediately ?

Solution . :

(a) Need = Max- Allocation

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

(b) System is in safe state because safe sequence exist and it is (P1; P3; P4; P2; P0).

(c) Yes, request can be granted immediately. The reason is Request <= Available that is (1, 0, 2) <= (3, 3, 2)

Example 3.12.7 MU - Nov 15. 10 Marks

Consider the snap of the system

Process	Allocation A B C D	Max A B C D	Available A B C D
P0	0 2 1 2	0 3 2 2	2 5 3 2
P1	1 1 0 2	2 7 5 2	
P2	2 2 5 4	2 3 7 6	
P3	0 3 1 2	1 6 4 2	
P4	2 4 1 4	3 6 5 8	

Answer the following questions using bankar's algorithm.

(i) What is the content of Matrix *need?*

(ii) Is the system in safe state?

(iii) If a request from process P1 arrives for (1,3,2,1) can the request be granted immediately?

Solution : Ans. (i)

Need[i, j] = Max[i,j] – Allocation[i,j]

so content of Need matrix is

	A	B	C	D
P0	0	1	1	0
P1	1	6	5	0
P2	0	1	2	2
P3	1	3	3	0
P4	1	2	4	4

(ii) Yes, System is safe and safe sequence (P0, P2, P1, P4, P3) exist.

(iii) As (3, 3, 4, 8) remains available in system, P1 can be granted (1,3,2,1) immediately

Syllabus Topic : Deadlock Detection

3.13 Deadlock Detection

If the system does not make use of deadlock prevention or deadlock detection algorithm, then system will observe the deadlock situation. In this situation, it is necessary that, system should have :

- Algorithm which examines the state of the system to decide whether a deadlock has occurred or not.
- Algorithm to recover from deadlock if it is occurred.
- Both the algorithms requires the overhead including run-time cost of maintaining necessary information and executing the detection algorithm and potential losses which are natural in recovering from deadlock.

Single instance of each resource type

- A deadlock detection algorithm makes use of a wait-for graph which is resource allocation graph. Wait for graph is used by algorithm if all resources have only a single instance. The wait for graph is obtained from the resource-allocation graph by eliminating the nodes of type resource and collapsing the suitable edges. This is shown in resource allocation graph Fig. 3.13.1. It shows that, process P1 is requesting resource R2 which is held by process P2.

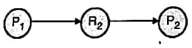


Fig. 3.13.1 : Resource allocation graph

- Above situation in resource allocation graph is represented as shown in wait-for graph Fig. 3.13.2.

Fig. 3.13.2 : Wait for graph of above resource allocation graph

- In this way system maintains the wait-for graph and if it contains the cycle, the deadlock exists in the system. By invoking cycle detection algorithm on wait-for graph, the deadlock can be detected. If n is number of vertices in wait-for graph, n^2 operations are required to detect the cycle.

Many instances of each resource type

- The wait-for graph discussed above is not applicable to multiple instances of a resource type.
- In this case, we have to use detection algorithm similar to Banker's, which simply investigates every possible allocation sequence for the processes which remain to be completed. The data structures used are as follows.
  - **A[m] :** Array A of size m shows the number of available resources.
  - **C[n][m] :** Two dimensional array C shows current allocation status of resources to each process.
  - **R[n][m] :** be the two dimensional array, describing the current outstanding requests of all processes. R[i][j] = k indicates process Pi want k instances of resource type Rj to accomplish the execution.

Algorithm

Q. Write note on "Deadlock Detection Algorithms".

1. Let W be an integer array of length m, initialized to array A (Available number of resources).

   Let F be a boolean array of length n, initialized to false (determines if process finished or not).

   For all i, if C[i] != 0, then F[i] = false; therwise F[i] = true.
2. Find an i such that both

   F[i] == false // Pi is currently *not* involved in a deadlock

   R[i] <= W

   If no such i exists, go to step 4
3. // reclaim the resources of process Pi

   W = W + C[i]; F[i] = true; Go to step 2
4. If F[i] == false for some i,

   then the system is in a deadlocked state. Moreover, if F[i] == false, then process Pi is deadlocked.

Run-time complexity : $O(m \times n^2)$

Invocation of detection algorithm depends on :

- Frequency of occurring the deadlock. If it is occurring frequently then algorithm will be called frequently.
- The number of processes that will be influenced by deadlock when it occurs.
- If the request of any process cannot be fulfilled then there are chances of deadlocks to occur.
- The detection algorithm is invoked in following two cases.
  - **Extreme :** Invoke the algorithm every time a request is denied
  - **Alternative :** Invoke the algorithm at less frequent time intervals :
    - Once per hour
    - Whenever CPU utilization < 40%

Disadvantage

- Cannot determine exactly which process 'caused' the deadlock.

Syllabus Topic : Deadlock Recovery

3.14 Deadlock Recovery

Q. Explain deadlock recovery in detail.

- It needs to recover from deadlock when it is detected. Several options exist after detection algorithm detects that a deadlock exists in the system. One possibility is to inform operator and let them decide how to deal with it manually.
- The second option is to allow the system to recover from the deadlock on its own (automatically). There are two solutions exist to break a deadlock. One solution is just to abort one or more processes so that circular wait will be broken. The second option includes preemption of some of the resources from one or more of the processes which are involved in deadlock.

Process Termination (Kill a process)

When deadlock occurs, the operating system decides to kill one or more processes. This will reclaim the resources acquired by that killed processes. The deadlock detection algorithm detects whether deadlock exist or not after killing the process. Following two methods are used for killing the process :

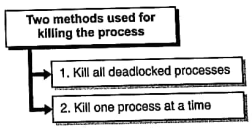


Fig. C3.5 : Methods used for killing the process

➜ 1. Kill all deadlocked processes

This method is simple and effective to eliminate the deadlock and clearly will break the deadlock cycle, but at a huge cost. If all these killed processes have been performed the computation for longer period of time then the partially computed result would be wasted. Recomputation will be done and is very costly.

➜ 2. Kill one process at a time

In this method, one process is killed at a time and detection algorithm is invoked to check whether deadlock is still exist or not in the system. Invocation of the detection algorithm after killing each process is considerable overhead. We must re-run algorithm after each kill.

- Killing a process is not easy. We should kill only those processes whose killing will incur minimum cost.

Different parameters that need to be consider to ensure the minimum cost

- Priority of the process.
- How much computation process has finished and how much more time does it need to finish the execution?
- The number of the resources process has used.
- The type of resources the process has used.
- The number of the resources the process will need to complete the execution.
- The number of processes needs to be killed.
- Whether the process is interactive or batch.

Resource Preemption

Incrementally preempt the resources from the process and reallocate resources to other processes until the circular wait is broken.

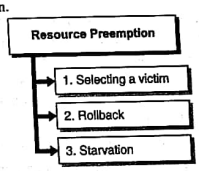


Fig. C3.6 : Incrementally preempt the resources

➜ 1. Selecting a victim

Which process and which resources should be selected for preemption to minimize the cost? The process which has finished almost all computation and only few amount of computation is remained should not be selected as a victim.

➜ 2. Rollback

After preempting the resources from the particular process, it cannot continue the execution as needed resources are preempted. It is required to rollback the process to safe state and restarts the execution from that state. As safe state is difficult to define, total rollback will be the simple solution.

➜ 3. Starvation

It may happen that, same process will be selected many times to preempt the resources.

Syllabus Topic : Dining Philosopher Problem

3.15 Dining Philosopher Problem

→ (Dec. 14, Nov. 15)

Q. Explain dining philosopher problem and solution to it
MU - Dec. 2014, Nov 2015, 10 Marks

Q. Explain Dining philosopher problem using semaphores.

- The problem is stated as : The work of five philosophers is thinking and eating. A table is laid for them and all agreed that only food that contributed to their thinking efforts was spaghetti. Each philosopher requires two forks to eat spaghetti.
- The provision for eating is on a round table with a big serving pot of spaghetti, five plates, and five forks. One plate is given to one philosopher. A philosopher sits at assigned place at the table. He has to use two forks on either side of the plate. He takes and eats some spaghetti.
- Algorithm should be designed which will allow the philosophers to eat. The algorithm must satisfy mutual exclusion (two philosophers should not use the same fork at the same time) while avoiding deadlock and starvation. This problem exemplifies basic problems in deadlock and starvation.
- The dining philosopher's problem can be seen as example of problems related to coordination of shared resources, which may occur when an application includes concurrent threads of execution. In the same way, this problem is a typical test case for assessing approaches to synchronization.

Solution using semaphores

- Following is the solution using semaphores. Each philosopher picks up first the fork on the left side and then the fork on the right side.
- After the philosopher is finished eating, the two forks are replaced on the table.
  - This solution, unfortunately, leads to deadlock: If all of the philosophers are hungry simultaneously, all will pick up the fork on their left, and right side fork will not be there. In this unseemly position, all philosophers starve.
  - Additional five forks (a more clean solution!) will solve the problem or use of just one fork to eat the spaghetti can be a solution. Following program shows the solution.

```
/* program dining philosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
    while (true) {
    think();
    wait (fork[i]);
    wait (fork [(i+1) mod 5]);
    eat();
    signal(fork [(i+1) mod 5]);
    signal(fork[i]);}
}
void main()
{
    parbegin (philosopher (0), philosopher (1),
    philosopher (2), philosopher (3),
    philosopher (4));
}
```

- In another strategy consider the helper who only allows four philosophers at a time into the dining room.
- With at most four seated philosophers, at least one philosopher will have access to two forks. Following is the solution using semaphore.

```
/* program dining philosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
    while (true) {
    think();
    wait (room);
    wait (fork[i]);
    wait (fork [(i+1) mod 5]);
    eat();
    signal (fork [(i+1) mod 5]);
    signal (fork[i]);
    signal (room); }
}
void main()
{
    parbegin (philosopher (0), philosopher (1),
    philosopher (2), philosopher (3),
    philosopher (4));
}
```


3.16 Exam Pack (University and Review Questions)

Syllabus Topic : Concurrency - Principles of Concurrency

Q. Explain the process synchronization in brief. *(Refer section 3.1.1)* (5 Marks) **(May 2016)**

Syllabus Topic : Interprocess Communication

Q. Explain the Inter-process communication in brief. *(Refer section 3.2)* (5 Marks) **(May 2016)**

Syllabus Topic : Process/Thread Synchronization

Q. Explain critical section problem. *(Refer section 3.3.1)* (5 Marks) **(Dec. 2014)**

Q. Explain critical section problem with its different solutions. *(Refer section 3.3.1)* (10 Marks) **(June 2015)**

Q. What is race condition? Explain with example. *(Refer section 3.3.2)*

Syllabus Topic : Mutual Exclusion

Q. What is mutual exclusion ? Explain its significance. *(Refer section 3.4)* (5 Marks) **(May 2016)**

Syllabus Topic : Operating System Support (Semaphores and Mutex)

Q. Give software approaches for mutual exclusion. *(Refer section 3.6.1)* (5 Marks) **(June 2015, Nov. 2015)**

Q. What is semaphore ? *(Refer section 3.6.1)*

Q. List the disadvantages of Semaphore. *(Refer section 3.6.1)*

Q. Explain Peterson solution to mutual exclusion. *(Refer section 3.7)*

Syllabus Topic : Programming Language Support (Monitors)

Q. How it is used to achieve mutual exclusion? Explain. *(Refer section 3.8)*

Syllabus Topic : Classical Synchronization Problems - Readers/Writers Problem

Q. Explain how Readers/writers problem can be solved with semaphores? *(Refer section 3.9.1)* (10 Marks) **(June 2015)**

Syllabus Topic : Producer and Consumer Problem

Q. Explain an algorithm for producer-consumer problem. *(Refer section 3.9.2)* (10 Marks) **(Dec. 2016)**

Q. What do you mean by busy waiting? What is wrong with it ? *(Refer section 3.9.2(A))* (5 Marks) **(Nov. 2015, Dec 2016)**

Syllabus Topic : Principles of Deadlock

Q. What is deadlock ? *(Refer section 3.10.1)* (2 Marks) **(June 2015, Nov. 2015, May 2016, Dec. 2016)**

Syllabus Topic : Principles of Deadlock : Conditions

Q. Explain necessary and sufficient condition for deadlock to occur. *(Refer section 3.10.2)* (5 Marks) **(June 2015, Nov. 2015, May 2016, Dec. 2016)**

Syllabus Topic : Resource Allocation Graphs

Q. Write note on Resource Allocation Graph. *(Refer section 3.10.3)* (5 Marks) **(Dec. 2016)**

Syllabus Topic : Deadlock Prevention

Q. What is difference between deadlock prevention and avoidance algorithms. *(Refer section 3.11)* (5 Marks) **(Dec. 2014, Nov. 2015)**

Q. Explain deadlock prevention. *(Refer section 3.11)* (2 Marks) **(June 2015)**

Syllabus Topic : Deadlock Avoidance

Q. What is difference between deadlock prevention and avoidance algorithms *(Refer section 3.12)* (5 Marks) **(Dec. 2014, Nov. 2015)**

Q. Explain deadlock avoidance, prevention and detection. *(Refer section 3.12)* (2 Marks) **(June 2015)**

Q. Suggest techniques to avoid deadlock. *(Refer section 3.12.1)* (1 Mark) **(May 2016, Dec. 2016)**

Syllabus Topic : Banker's Algorithm for Single and Multiple Resources

Q. Explain data structures used in bankar's algorithm. *(Refer section 3.12.1(B))* (5 Marks) **(June 2015)**

Q. Explain the banker's algorithm in detail. *(Refer section 3.12.1(B))* (10 Marks) **(Dec. 2016)**

Q. Explain resource-request algorithm. *(Refer section 3.12.1(C))* (5 Marks)

Example 3.12.1 (10 Marks) **(Dec. 2014)**

Example 3.12.6 (10 Marks) **(June 2015)**

Example 3.12.7 (10 Marks) **(Nov. 2015)**

Syllabus Topic : Deadlock Detection

Q. Write note on "Deadlock Detection Algorithm" *(Refer section 3.13)*

Syllabus Topic : Deadlock Recovery

Q. Explain deadlock recovery in detail. *(Refer section 3.14)*

Syllabus Topic : Dining Philosopher Problem

Q. Explain dining philosopher problem and solution to it *(Refer section 3.15)* (10 Marks) **(Dec. 2014, Nov. 2015)**

Q. Explain Dining philosopher problem using semaphores. *(Refer section 3.15)*

□□□

Module 4

CHAPTER 4

Memory Management

Syllabus Topics

Memory Management : Memory Management Requirements, Memory Partitioning: Fixed Partitioning, Dynamic Partitioning, Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit, Next Fit, Buddy System, Relocation. Paging, Segmentation.

Virtual Memory : Hardware and Control Structures, Demand Paging, Structure of Page Tables, Copy on Write, Page Replacement Strategies: FIFO, Optimal, LRU, LFU Approximation, Counting Based. Allocation of frames, Thrashing

Background

- Memory is an important resource of the computer system that needs to be managed by the operating system.
- To execute the program, user needs to keep the program in main memory.
- The main memory is volatile. Therefore a user needs to store his program in some secondary storage which is non volatile.
- Every process needs main memory since a process code, stack, heap (dynamically-allocated structures), and data (variables) must all reside in memory.
- The management of main memory is required to support for multiprogramming. Many executables processes exist in main memory at any given time.
- Different processes in main memory have different address space.
- Memory manager is the module of the operating system responsible for managing memory. Programs after completion of execution move out of main memory or processes suspended for completion of I/O can be swapped out on secondary storage to free the main memory for other processes.
- New processes are required to be loaded in main memory. If available main memory is not sufficient to hold all processes swapping between main memory and secondary memory is done.
- Memory managers move processes back and forth between memory and disk during execution.
- So it is required that operating system should have some strategy for the management of memory.

Syllabus Topic
Memory Management Requirements

4.1 Memory Management Requirements

Q. What are different requirements for memory management?

Following are the requirements for memory management.

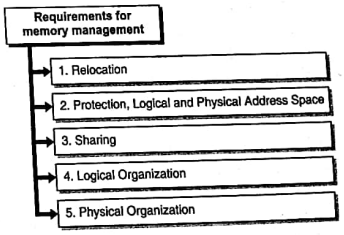


Fig. C4.1 : Requirements for memory management

Module 4

CHAPTER 4

Memory Management

Syllabus Topics

Memory Management : Memory Management Requirements, Memory Partitioning: Fixed Partitioning, Dynamic Partitioning, Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit, Next Fit, Buddy System, Relocation. Paging, Segmentation.

Virtual Memory : Hardware and Control Structures, Demand Paging, Structure of Page Tables, Copy on Write, Page Replacement Strategies: FIFO, Optimal, LRU, LFU Approximation, Counting Based. Allocation of frames, Thrashing

Background

- Memory is an important resource of the computer system that needs to be managed by the operating system.
- To execute the program, user needs to keep the program in main memory.
- The main memory is volatile. Therefore a user needs to store his program in some secondary storage which is non volatile.
- Every process needs main memory since a process code, stack, heap (dynamically-allocated structures), and data (variables) must all reside in memory.
- The management of main memory is required to support for multiprogramming. Many executables processes exist in main memory at any given time.
- Different processes in main memory have different address space.
- Memory manager is the module of the operating system responsible for managing memory. Programs after completion of execution move out of main memory or processes suspended for completion of I/O can be swapped out on secondary storage to free the main memory for other processes.
- New processes are required to be loaded in main memory. If available main memory is not sufficient to hold all processes swapping between main memory and secondary memory is done.
- Memory managers move processes back and forth between memory and disk during execution.
- So it is required that operating system should have some strategy for the management of memory.

Syllabus Topic
Memory Management Requirements

4.1 Memory Management Requirements

Q. What are different requirements for memory management?

Following are the requirements for memory management.

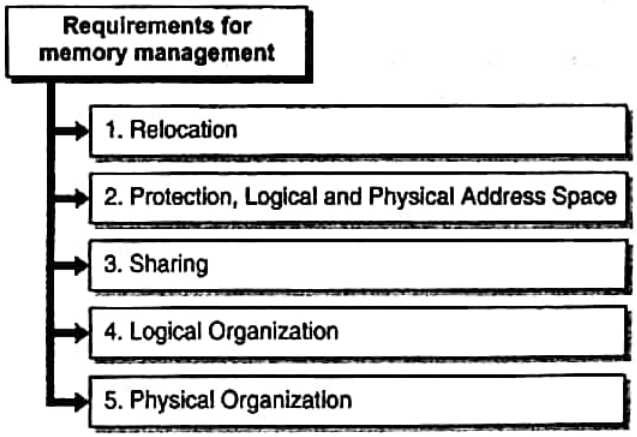


Fig. C4.1 : Requirements for memory management

4.1.1 Relocation

Q. Write short note on Relocation.

- There can be many numbers of processes simultaneously in main memory when multiprogramming approach is used.
- Programmers remain unaware about which program will be in memory during execution of his or her program.
- In order to improve the processor utilization, processes gets swapped in and out of main memory. This swapping of processes is to get more number of ready processes available for processor for execution.
- The swapped process from main memory to disk can be swapped again from disk to main memory
- Next time process may not get previous portion of main memory. It needs to be relocated to new memory area.
- The operating system manages the memory and it keeps track on address from which program begins the execution.
- The processor hardware and operating system software is responsible to translate the memory references located in the code of the program into actual physical memory addresses, reflecting the current location of the program in main memory.

4.1.2 Protection, Logical and Physical Address Space

Q. With the help of example, clearly differentiate between logical and physical address space.

- An address generated by CPU is called logical address or virtual address and an address generated by Memory Management Unit (MMU) is called physical address.
- If size of program written by user is 1000 K and loaded in main memory from address 5000 to 6000.
- The physical address space will become 5000 to 6000 and logical address space is 0 to 1000.
- The logical address space is collection of all logical addresses generated by a program and a physical address space is collection of all physical addresses corresponding to these logical addresses.
- If the binding of instructions and data to memory addresses are at compile time or load time, the logical and physical addresses are same.
- If the same binding occurs at run time then logical and physical addresses are different.
- The relocation register (base register) contains address 5000. Physical address (5200) can be calculated as follows. Consider logical address 200 of the same program.
- Physical address (5200) = logical address (offset) (200) + content of base register (5000)
- Mapping from virtual to physical addresses is done by the memory management unit (MMU) at run time.
- The user program deals with logical addresses. The memory-mapping hardware converts logical addresses into physical addressed logical addresses.
- Memory protection is done by using two registers, usually a base and a limit, as shown in Fig. 4.1.1.
- The base register holds the smallest legal physical memory address (in our example 5000).
- The limit register gives the size of the range of physical addresses. For example, if the base register holds 5000 and the limit register is 1000, then the program can legally access all addresses from 5000 through 6000.

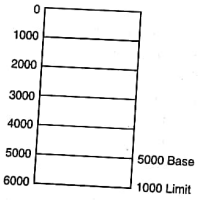


Fig. 4.1.1: Memory Protection with base and limit registers

4.1.3 Sharing

- Protection mechanism should allow many processes to access the same part of memory.
- All the processes executing same program should be allowed to use single copy of program.
- Coordinating processes should be allowed to use same data structures and controlled access to it should be ensured.

4.1.4 Logical Organization

- The main memory is a linear, or one-dimensional, address space, comprising the sequence of bytes or words.
- Physically secondary memory organization is same like main memory.
- The programs written may be organized into modules. Some modules cannot be modified (read only, execute only) and some of which have data that can be modified.
- If the operating system and hardware is able to handle user programs and data in the form of modules then many benefits can be realized:
  - Each module independently can be implemented and compiled, with all references from one module to another resolved by the system at run time.
  - The different degrees of protection (read only, execute only) can be assigned to different modules with small additional cost.
  - The modules can be shared among processes.
- Segmentation technique satisfies above requirements.

4.1.5 Physical Organization

- Two levels of organization with main memory and secondary memory is available in system.
- Main memory (RAM) offers fast access than secondary memory with high cost and it is volatile. Secondary memory provides permanent storage.
- The information exchange between main memory and secondary memory should be handled by system and not by programmer.

Syllabus Topic : Memory Partitioning

4.2 Memory Partitioning

4.2.1 Monoprogramming

- In early computer systems and early microcomputer operating systems, mono-programming concept was used.
- Only one program was residing in main memory at given point of time.
- Operating system was residing in some portion of memory and other portion was fully devoted to the single executing process.
- This system was simple to design but was not supporting multiprogramming. In this approach no relocation was needed as program could be always loaded at the same location.

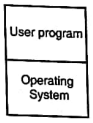


Fig. 4.2.1 : Monoprogramming

4.2.2 Multiprogramming

- Multiprogramming is required to support multiple processes simultaneously.
- Since multiple processes are resident in memory at the same time, it increases processor utilization if the processes are I/O bound, CPU does not remain idle.
- CPU utilization increases as it executes multiple processes on time sharing basis.
- Multiprogramming gives illusion of running multiple processes at once and provides users with interactive response to processes.

Syllabus Topic
Memory Partitioning - Fixed Partitioning

4.2.3 Multiprogramming with Fixed and Variable Partitions (Contiguous Allocation)

Q. Explain memory management with Fixed and Variable Partitions.
Q. What is internal fragmentation and external fragmentation? Explain with example.

- In this scheme memory is divided into a number of fixed-sized partitions. Each partition may contain exactly one process.
- The number of partitions and its size would be defined by the system administrator when the system starts up.
- The degree of multiprogramming will be high if the number of partitions created is more.
- Always a selected process from the input queue is placed into the free partition.
- When the executing process finishes the execution and terminates, the partition becomes free for another

process. The operating system maintain a table to keeps track on free and occupied partitions by processes.

- Partitions can be of fixed size or of variable size. Fixed sized partitions are relatively simple to implement. The problems with the fixed size partitions are :
  - If program size is larger than the partition size, program cannot be stored in partition.
  - The second problem is **internal fragmentation**. If partition size is larger than program size, some space of the partition will be unused within that partition. Fig. 4.2.2 shows fixed size partitions.

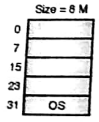


Fig. 4.2.2 : Fixed size partitions

  - As a name suggest, in a variable-sized partition, partitions of different sizes are available.
  - Memory is partitioned in partitions of different sizes. The best-fit strategy is used to allocate the partitions to the processes.
  - In best-fit, that partition is chosen to fit the process so that less amount of memory will be wasted. It means smallest partition big enough to accommodate the process is chosen.
- A queue is organized for each size of the partition where processes wait for that partition.
- Partitions are allocated to the processes with best-fit policy. Best fit policy ensures to allocate the partition that is big enough for the process to reside.
- Therefore variable partitions minimize internal fragmentation. However, such an allocation may reduce the performance as processes would queue up in best fit queue even though other partitions are free.
- For example, it may have many processes queued up in a queue of partition of size 8M, and no processes are queued up in a queue for the partition size 16M.
- In both fixed and variable size partitions it may happen that, partition can remain empty even though processes are waiting in other partitions queue leading to **external fragmentation**. Fig. 4.2.3 shows variable size partitions.

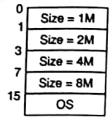


Fig. 4.2.3 : Variable size partitions

☞ **Difference between Internal and external Fragmentation**

Sr. No.	Parameters	Internal Fragmentation	External Fragmentation
1.	Definition	Internal fragmentation takes place in fixed partitioning technique	External fragmentation takes place in variable partitioning technique
2.	Partition Size	If partition size is larger than program size, some space of the partition will be unused within that partition called as internal fragmentation	In both fixed and variable size partitions it may happen that, partition can remain empty even though processes are waiting in other partitions queue leading to external fragmentation.
3.	Memory allocation	Contiguous memory allocation reduce the internal fragmentation	Paging and segmentation technique is the solution over external fragmentation problem.

Syllabus Topic
Memory Partitioning - Dynamic Partition

4.2.4 Dynamic Partition Technique

- Dynamic partition technique can reduce the **internal and external** fragmentations.

- In this technique basically a variable partitioning with a variable number of partitions determined dynamically. Initially, all memory is available for user processes, and is considered as one large block, of available memory, a hole.

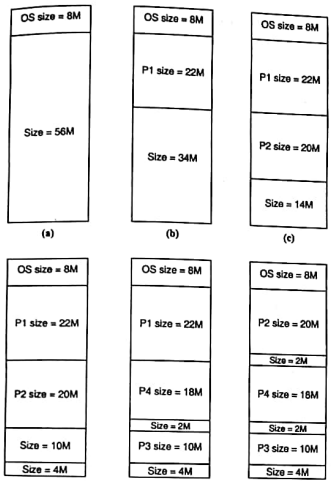


Fig. 4.2.4 : Creation of scattered holes in dynamic partition technique

- When a process arrives and needs memory, it searches for a hole large enough to fit for this process.
- This technique ensures the efficient use of main memory. There is no internal fragmentation.
- However, if the many small holes are scattered, it can't be allocated to one large size process. Consider the following example of Fig. 4.2.4.
- Initially 64 MB main memory is available. 8 MB is allocated to operating system as shown in Fig. 4.2.4(a). For user processes 56 MB memory is available and considered as one large single hole.
- Processes P1, P2 and P3 are loaded and to these processes sufficient memory space can be allocated, creating the hole of size 4 MB at the end of memory (Figs. 4.2.4(b), (c) and (d))
- At some point of time none of the allocated processes are ready.
- Operating system swaps out P2 and in its place new process P4 of size 18 MB is swapped in Fig. 4.2.4(e). The hole of 2 MB is created.
- Again after some time since none of the processes in main memory is ready and only P2 is in ready suspend state, is available.
- To swap in P2, there is no sufficient space is available. So operating system swaps P1 out and swaps P2 back in as shown in Fig. 4.2.4(f).
- The example shows that, there are many scattered small size holes.
- These holes cannot be allocated to the process of larger size than these holes. This is called **external fragmentation.**

4.2.5 Compaction

Q. What is compaction? Explain with example.

- Solution to above discussed problem is compaction. **Compaction** is a technique to covert many small size scattered holes into one large size continuous hole.
- It is relocation of the various processes so as to accumulate all the free space in one place.
- Due to compaction, there is efficient use of processor. Fig. 4.2.5 shows the memory after compaction.
- Fig. 4.2.5(a) shows the scattered small size holes. If processes P4 and P3 moved up, we get one large size hole at the end of memory as shown in Fig. 4.2.5(b).
- If processes P2, P4 and P3 moved downward, we get one large size hole of 8 MB at the start of user memory area as shown in Fig. 4.2.5(c).
- However in this, 48 MB processes need to be moved compared to 28 MB shown in Fig. 4.2.5(b). Fig. 4.2.5(d) shows the 28 MB shifting of processes.
- Compaction should ensure the moving of processes having minimum total size. Compaction increases the degree of multiprogramming.
- It is because processes can be loaded in the memory which is available by combining many scattered holes.

- If relocation is done at compile time (static), compaction is not possible. It is possible only if relocation is dynamic and done at execution time.

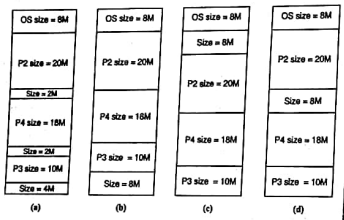


Fig. 4.2.5 : Compaction

4.3 Dynamic Loading

- Dynamic loading ensure the better memory space utilization. Unless and until called, routine is not loaded in main memory in dynamic loading.
- All routines remain stored on disk in a relocatable load format.
- The main program is loaded into main memory and is executed.
- The advantage of dynamic loading is that, memory space is utilized only for the routines which are currently need to be executed and other unused routines reside on disk.
- If user designs the program which uses many library routines then operating system will itself support for dynamic loading.
- No special support require from operating system.

4.4 Overlays

- In the early days, as large programs were too large to fit into their partitions, programmers created overlays to solve this problem.
- Rather than loading an entire program into memory at once, we can divide it into those modules that are always required and those that are required only at certain times during program execution.
- Overlays involve moving data and program segments in and out of memory which helps for reusing the area in main memory.
- Overlays were physical divisions of the program that were established by the programmer.
- To complete the execution, it is required that, entire program and data of a process must be in physical memory.
- It is required to have process in main memory for execution and hence physical memory should accommodate the process.
- The process size can be larger than the amount of memory allocated to it.
- The overlays allows to have needed modules and data in main memory at any given time.
- The data and modules which are not required for execution at that time are swapped out of memory.
- Next time, the needed modules are loaded into space which is freed now by swapping the modules not required for execution.
- An overlay handler exists in another area of memory and is responsible for swapping overlay pages or overlay segments.
- Overlays are implemented by user; no special support needed from operating system, programming design of overlay structure is complex.
- Consider the example of two pass assembler. As we know that pass 2 is executed after pass 1, it is not necessary to keep both modules of pass 1 and pass 2 in memory simultaneously.
- During pass 1 symbol table is constructed and pass 2 generates machine-language code.
- Symbol table and common routines are required in both passes.
- Consider the memory requirement of pass 1 module, pass 2 module, symbol table and common routines.

Pass 1 Module	100 K
Pass 2 Module	90 K
Symbol Table	30 K
Common Routines	30 K
Total	**250 K**

- Total 250 K memory is needed to run the assembler. If only 200 K memory is available then overlays can be defined as overlay A and overlay B.

- Overlay A contains pass 1 module, symbol table and common routines. Overlay B contains pass 2 module, symbol table and common routines.
- Overlay driver of 20 K is loaded in memory. Initially overlay A is loaded in memory which requires 160 K of memory.
- Once overlay A finishes execution, overlay driver loads overlay B in pace of overlay A by overwriting overlay B. Overlay B requires 160 K of memory.

4.5 Swapping

Q. Explain Swapping in detail.

- Systems in which it is possible to move entire processes between disk and main memory during execution are called swapping systems.
- In a time sharing operating system, system's memory is allocated to multiple processes.
- In order to have sufficient memory free, it will be required to swap data between memory and secondary storage (disk).
- If we use round robin CPU scheduling algorithm then after expire of time quantum memory manager will swaps out the finished process and swaps in the another process.
- It increases degree of multiprogramming ensures effective management of memory among many processes.
- If scheduling is priority based, then after arrival of high priority process low priority process is swapped out on disk.
- After completion of high priority process again a low priority process gets swapped in memory by memory manager.
- Swapping needs a backing store to store a swapped data. The backing store is usually a fast disk and having capacity enough to store copies of all memory images for all users.
- This backing store must offer direct access to these memory images.
- The operating system maintains a ready queue for the processes which are ready to execute.
- These processes have their memory images on the backing store or in memory.
- The CPU scheduler calls the dispatcher program, when it decides to run the process. If dispatcher does not find next process in the queue in main memory, it swaps in the desired process.
- If memory region is not free to bring the process from backing store, dispatcher swaps out current process from the memory.
- It then reloads registers and transfers control to the selected process. The context-switch time in such a swapping system is quite high

Syllabus Topic : Memory Allocation Strategies

4.6 Memory Allocation Strategies

Q. Explain different memory allocation strategies.
Q. Discuss partition selection algorithm in brief.

If process requests the free hole, then question is which free hole should be allocated to this process. Following are the strategies used to allocate the free hole to the requesting process.

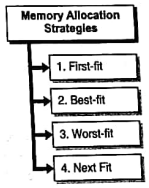


Fig. C4.2 : Memory allocation strategies

Syllabus Topic : First Fit

→ 1. First-fit

In first-fit strategy of memory allocation, the very first available hole that is big enough to occupy an incoming process get allocated to that process. There are two options available to start the search. Searching can either initiate from the location where previously search was stopped or at the start of the set of holes. Searching end when free hole that is big enough to hold the process is located.

Syllabus Topic : Best-Fit

➔ **2. Best-fit**

In best-fit strategy of memory allocation, the smallest hole that is big enough to occupy an incoming process get allocated to that process. If the list of holes is not kept in sorted order of size then the complete list needs to be searched. This approach leads to create the smallest leftover hole.

Syllabus Topic : Worst Fit

➔ **3. Worst-fit**

In worst-fit strategy of memory allocation, always the largest hole gets allocated to incoming process. Again, If the list of holes is not kept in sorted order of size then the complete list needs to be searched. This approach leads to create the largest leftover hole which can be used again to allocate the next incoming process.

Syllabus Topic : Next Fit

➔ **4. Next-fit**

Starts the searching of memory from the position of the previous placement, and selects the next available block that is large enough.

Example 4.6.1

Given memory partition 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 the processes of 212 K, 417 K, 112 K and 426 K in order? Which algorithm makes the most efficient use of memory?

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 4.6.2

Given memory partition of 150k, 500k, 200k, 300k and 550k (in order), how would each of the first fit, best fit and worst fit algorithm place the processes of 220k, 430k, 110k , 425k (in order). Which algorithm makes the most efficient use of memory ?

Solution :

First-fit

- 220 K is put in 500 K partition.
- 430 K is put in 550 K partition.
- 110 K is put in 150 k partition.
- 425K must wait.

Best-fit

- 220 K is put in 300 K partition.
- 430 K is put in 500 K partition.
- 110 K is put in 150 K partition.
- 425 K is put in 550 K partition.

Worst-fit

- 220 K is put in 550 K partition.
- 430 K is put in 500 K partition.
- 110 K is put in 330 K partition (550 K – 220 K).
- 425 K must wait.

In this example, Best-fit turns out to be the best.

Syllabus Topic : Buddy System

4.7 Buddy System

Q. Explain Buddy system.

Fixed partitioning technique leads to inefficient use of space as number of active processes can be in limited number. Dynamic partitioning technique suffers through compaction. Solution to these problems is buddy system.

- In this system, memory blocks are of size 2^K words, $L \leq K \leq U$, where 2^L is the least size block that is allocated and 2^U is the largest size block that is allocated; usually 2^U is the size of the total memory existing for allocation.
- Initially whole memory is treated as a single block having size 2^U. For request of space size r which is $2^{U-1} < r \leq 2^U$ the whole block is given or else block is divided in two equal buddies of size 2^{U-1}.
- In case of holding the condition $2^{U-2} < r \leq 2^{U-1}$, request is fulfilled by anyone buddies out of two.
- If request is not fulfilled then one of the buddy is divided again in two partitions.
- This process goes on until the smallest block greater than or equal to r is produced and allocated to the request.
- The buddy system keeps all the time a list of unallocated blocks of each size 2^i.
- A hole or unallocated block may be taken out from the $(i + 1)$ list by partitioning it in half to generate two buddies of size 2^i in the i list.
- Whenever a pair of buddies on the i list both become unallocated, they are taken out from that list and combined into a single block on the $(i + 1)$.

Syllabus Topic : Relocation

4.8 Relocation

Consider the partitions of memory as follows.

First partition: 100K (Operating system)

Second partition: 100K

Third partition: 200K

Fourth partition: 300K

- Consider very first instruction is a call to a function at absolute address 100 within the binary file created by the linker.
- If loader loads the program in first partition at address 100K, then instruction will find OS at absolute address 100.
- If instead of first partition, loader loads the program in second partition, the instruction will jump to address 100K + 100.
- For third partition instruction will jump to address 200K+100 and so on. This problem is called as relocation problem.
- The solution to this problem is to in fact modify the instructions as the loading of program is done in memory.
- If programs are loaded in first partition then 100K is added to each address, programs loaded into second have 200K added to addresses, and so on.
- To carry out relocation at the time of loading like this, the binary program must have linker included list or bitmap indicating which addresses are to be relocated and which are opcodes, constants, or other items not needed to relocate.

Syllabus Topic : Paging

4.9 Paging

➔ **(June 15)**

Q. What is paging? **MU - June 2015. 5 Marks**

- Paging is memory management technique which allows physical address space of the process to be noncontiguous.
- In some sense, paging mechanism is similar to reading of a book. When we read a book we only see and need to open the current page to read.
- All the other pages are not visible to us and remains closed. In the same manner, we can say that even when we may have a large program available, the processor only needs a small set of instructions to execute at any time.
- In fact, all these instructions which the processor needs to execute are within a small proximity of each other.
- This is like a page which contains all the statements which we are currently reading.
- In this way paging allows to keep just the parts of a process that we're using in memory and the rest on the disk.

4.9.1 Basic Operation

Q. What is Page Table? Explain the conversion of Virtual Address to Physical Address in Paging with example.

- The problem of external fragmentation is avoided by using paging.
- Paging is implemented by integrating the paging hardware and operating system.

- In this technique, logical memory is divided into blocks of the same size called pages.
- The physical memory is divided into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes, also larger sizes possible in practice). The page size and frame size is equal.
- Initially all pages remains on secondary storage (backing store). When a process is needed to be executed, its pages are loaded into any available memory frames from the secondary storage.

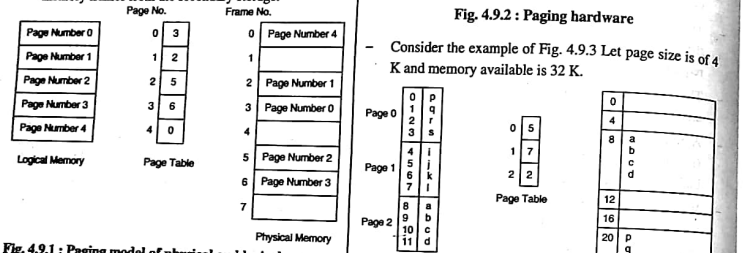


Fig. 4.9.1 : Paging model of physical and logical memory

- Fig. 4.9.1 shows the paging model of physical and logical memory.
- Following basic operations is done in paging :
  - CPU generates logical address and it is divided into two parts: a page number (p) and a page offset (d).
  - The page number is used as an index into a page table.
  - The base address of each page in physical memory is maintained by page table.
  - The combination of base address with page offset defines the physical memory address that is sent to the memory unit.
  - The physical location of the data in memory is therefore at offset *d* in page frame f.
- Because of the paging the user program sights the memory as one single contiguous space, it gives the illusion that memory contains only one program.
- But the user program is spread throughout main memory (physical memory). The logical addresses are translated into physical addresses.

Fig. 4.9.2 shows the operation of the paging hardware.

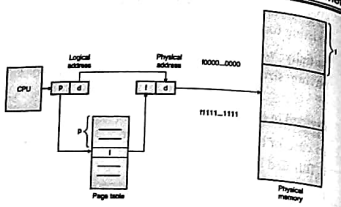


Fig. 4.9.2 : Paging hardware

- Consider the example of Fig. 4.9.3 Let page size is of 4 K and memory available is 32 K.

Fig. 4.9.3 : Paging example for a 32 K memory with 4 K pages

- Page 0 is in frame number 5. Page 1 is in frame number 7 and Page 0 is in frame number 5. Logical address 4 is in page 1 and offset is 0. Page 1 is mapped to frame 7. So physical address 28 = ((7 × 4) + offset 0).
- Logical address 10 is in page 2 and offset is 2. Page 2 is mapped to frame 2.

  So physical address 10 = ((2 × 4) + offset 2).

4.9.2 Memory Protection and Sharing

- Page table contains the frame number. So for the reference of the page, page table is verified to get correct frame number.
- Protection bit is associated with each page to denote whether page is read-write or read only.
- If try is made to write read only page, a hardware trap goes to operating system.

- A legal page remains in logical address space of the process and in page table this page is marked with valid (v) bit.
- Otherwise page is illegal and marked with invalid (i) bit.
- Paging also provides advantages of sharing of common code.
- The reentrant code cannot be modified. If many users share this code, only one copy needs to be kept in main memory.
- The page table of different user points to same physical memory (frames). In this case data used by different users can be different so data pages are mapped to different frames.
- By simply having page table entries for different processes point to the page frame, the operating system can create regions of memory that are shared by two or more processes.

4.9.3 Translation Look aside Buffer

- In paging, in order to access a byte first page-table entry needed to be accessed and then byte is accessed.
- Therefore, two memory accesses are needed to be performed.
- It results in slowing down memory access by a factor of 2. This delay would be unbearable under most situations.
  - The above problem can be resolved by using translation look-aside buffer which is fast searching hardware cache. The TLB is associative, high-speed memory.
  - The every entry in TLB is divided in two parts one is key and other is value.
  - The associative memory is presented with an item is compared with all keys at the same time. If there is match then the corresponding value field is returned.
  - Although the hardware is expensive, the searching mechanism supported in this way is very fast. Normally, TLB holds small number of entries, often ranging the numbers between 64 and 1,024.

4.9.4 Effect of Page Size on Performance

- Page size is important factor to decide the performance.
- If page size is small then the any program will be divided in many numbers of pages leading to have a large page table.
- On some machine, it is required to load the page table in hardware register every time when context occurs.
- Hence more time will be required to load the larger size page table due to small size of the pages. More number of transfers to and from disk will be required due to more number of page faults.
- Hence more time will be elapsed in seek and rotational delay. If page size is large then last page will remain somewhat empty leading to internal fragmentation.
- Due to large size of pages, most of the unused part of program can remain in memory.
  - The size of the memory divided by the page size is equal to the number of frames. If the size of page is increased then the number of frame is decreases.
  - Having a fewer frames will increase the number of page faults because of the lower freedom in replacement choice.
  - Large pages would also waste space by Internal Fragmentation.
  - On the other hand, a larger page-size would draw in more memory per fault; so the number of fault may decrease if there is limited contention.
  - Larger pages also reduce the number of TLB misses.

Hardware Support for Paging

→ (Dec. 16)

Q. Explain the hardware support for paging MU - Dec. 2016. 10 Marks

- Each operating system has its own methods for storing page tables. Most allocate a page table for each process.
- A pointer to the page table is stored with the other register values (like the instruction counter) in the process control block.
- When the dispatcher is told to start a process, it must reload the user registers and define the correct hardware page-table values from the stored user page table.
- The hardware implementation of the page table can be done in several ways.
- In the simplest case, the page table is implemented as a set of dedicated registers. These registers should be built with very high-speed logic to make the paging-address translation efficient.
- Every access to memory must go through the paging map, so efficiency is a major consideration.
- The CPU dispatcher reloads these registers, just as it reloads the other registers. Instructions to load or modify the page-table registers are, of course,

privileged, so that only the OS can change the memory map.

- The use of registers for the page table is satisfactory if the page table is reasonably small (for example, 256 entries).
- Most contemporary computers, however, allow the page table to be very large (for example, 1 million entries).
- For these machines, the use of fast registers to implement the page table is not feasible.
- Rather, the page table is kept in main memory, and a page-table base register (PTBR) points to the page table.
- Changing page tables requires changing only this one register, substantially reducing context-switch time.
- The problem with this approach is the time required to access a user memory location.
- If we want to access location *I*, we must first index into the page table, using the value in the PTBR offset by the page number

Example 4.9.1

For a paged system TLB hit ratio is 0.9. Let the RAM access time t be 20 ns. and TLB access time T be 100 ns. Find out

(1) Effective memory access with TLB

(2) Effective memory access without TLB

(3) Reduction in effective access time.

Solution :

TLB hit ratio, RAM access time t and TLB access time T is given in problem.

(1) Effective memory access with TLB (Ea)

$Ea = ((TLB\ hit\ ratio) \times (T + t)) + (1 - TLB\ hit\ ratio) \times (2T + t)$

$= (0.9 \times (100 + 20)) + (1 - 0.9) \times ((2 \times 100) + 20)$

$= 130$ ns.

(2) Effective memory access without TLB (Ewt)

$Ewt = 2T = 200$ ns

(3) Reduction in effective access time

$= (Ewt - Ea) \times (T/Ewt)$

$= (200 - 130) \times (100/200)$

$= 35\ \%$

Syllabus Topic : Segmentation

4.10 Segmentation

Q. Explain segmentation with paging.

Q. What is segmentation? Explain it with example.

- In segmentation, a user program and its associated data can be subdivided into a number of segments.
- Different segments of the programs can have different length.
- Although there is a maximum segment length, the length depends on purpose of the segment in the program.
- Segments are arbitrarily-sized contiguous ranges of the address space.
- They are a tool for structuring the application. As such, the programmer or compiler may decide to arrange the address space using multiple segments.
- For example, this may be used to create some data segments that are shared with other processes, while other segments are not.
- Compiler can create the different segment for global variables, code, thread stack, library etc.
- In segmentation, a program may occupy more than one partition, and these partitions need not be contiguous.
- Internal fragmentation is eliminated in segmentation but, like dynamic partitioning, it suffers from external fragmentation. In segmentation :
  - Logical address it is divided into two parts: a segment number (s) and offset (d) into that segment.
  - Each entry of segment table contains segment base and segment limit.
  - Segment base indicate starting physical address of the segment in main memory and segment base denotes length of the segment.
  - Segment number is used as index to the segment table.
  - Offset (d) always between 0 and segment limit otherwise error will occur indicating attempting access beyond the end of the segment.
  - Fig. 4.10.1 indicates 4 segments of the program having different sizes. Segment table contains base and limit of each segment.

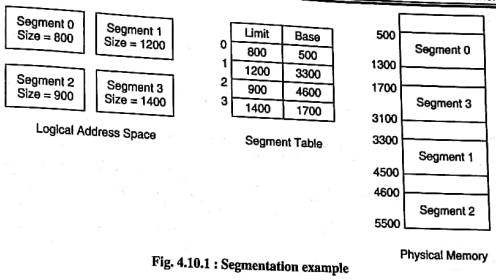


Fig. 4.10.1 : Segmentation example

Segment 3 is of 1400 bytes long and its base is 1700. Reference to byte 89 of segment 3 is calculated as 1700 + 89 = 1789.

Example 4.10.1

On a system using simple segmentation, compute the physical address for each of following the logical address if address generates a segment fault then indicate so.

Segment	Base	Length
0	330	124
1	876	125
2	111	99
3	498	302

(i) 0, 99 (ii) 2, 78 (iii) 1, 268 (iv) 3, 222 (v) 0, 111

Solution :

Length given is limit of the segment.

(i) **0, 99 :** Segment 0 is present in segment table. Now we have to check (offset < limit) or not. If offset is less than limit then physical address = base + offset.

99<124 therefore physical address = 330 + 99 = 428

(ii) **2,78 :** Segment 2 is present in segment table. 78<99 therefore physical address = 111 + 78 = 189

(iii) **1,268 :** Segment 1is present in segment table. 268<125 (false) therefore segment fault and trap to OS . Not possible to calculate physical address

(iv) **3,222 :** Segment 3 is present in segment table. 222<302 therefore physical address = 498 + 222 = 720

(v) **0, 111 :** Segment 0 is present in segment table. 111<124 therefore physical address = 330 + 111 = 441

Example 4.10.2

Consider the following segment table

Segment	Base	Length
0	219	600
1	2300	14
3	1327	580
4	1952	96

What are physical addresses of following logical addresses.

(i) 0, 430 (ii) 1, 10 (iii) 2, 500 (iv) 3, 4000 (v) 4, 112

Solution :

(i) **0, 430 :** Segment 0 is present in segment table. Now we have to check (offset < limit) or not. If offset is less than limit then physical address = base + offset.

430<600 therefore physical address = 214 + 430 = 649

(ii) **1, 10 :** Segment 1 is present in segment table. 10<14. Therefore physical address = 2300 + 10 = 2310

(iii) **2, 500 :** Segment 2 is present in segment table. 500<580 (false) therefore segment fault and trap to OS. Not possible to calculate physical address

(iv) **3, 400 :** Segment 3 is present in segment table. 400<580 therefore physical address = 1329 + 400 = 1729

(v) **4, 112 :** Segment 4 is present in segment table. 112<96 (false) therefore segment fault and trap to OS. Not possible to calculate physical address.

4.10.1 Difference between Paging and Segmentation

→ (Dec. 16)

Q. Explain the difference between paging and Segmentation. MU - Dec. 2016, 5 Marks

Sr. No.	Parameters	Paging	Segmentation
1.	Logical address	CPU generates logical address and it is divided into two parts: a page number (p) and a page offset (d).	Logical address it is divided into two parts segment number (s) and offset (d) into the segment.
2.	Table index	The page number is used as an index into a page table.	Segment number is used as index to the segment table.
3.	Base Address	The base address of each page in physical memory is maintained by page table.	Segment base indicate starting physical address of the segment in main memory and segment base denotes length of the segment.
4.	Physical Address	The combination of base address with page offset defines the physical memory address that is sent to the memory unit.	Segment base indicate starting physical address of the segment in main memory and segment base denotes length of the segment
5.	Fragmentation	Suffer through internal fragmentation	Suffer through external fragmentation

4.11 Segmentation with Paging

- In segmentation with paging, advantages of both paging and segmentation are implemented together to make it more effective.
- In paging page size is same and protection and sharing also provided in segmentation.
- In this technique, each of the segment is divided in multiple pages and there is also page table maintained for each of the segment.
- Segment offset part of the logical address (containing offset and segment number) is further split in page number and page offset.
- Every segment table entry includes segment base, segment limit and address of page table of that particular segment.
- As shown in Fig. 4.11.1, logical address has three components: segment number (SN), page number (PN) and page offset.
- Memory management unit uses segment number as an index into segment table in order to locate the address of page table.
- Then PN belonging to logical address is attached to high order end of address of page table and it is used as an index to page table to locate entry in page table.
- Frame number in page table entry is used to calculate physical address. This frame number is attached to high order end of page offset to obtain physical address.

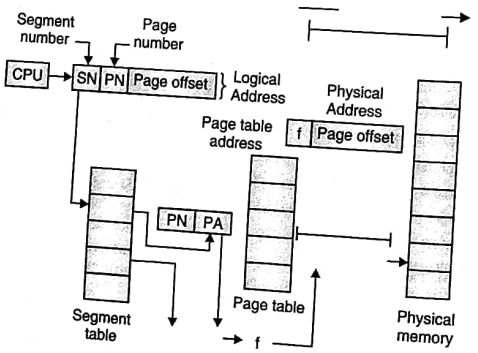


Fig. 4.11.1: Segmentation with paging

Syllabus Topic : Virtual Memory

4.12 Virtual Memory

→ (May 16)

Q. Write short note on Virtual Memory.
MU - May 2016. 5 Marks

- There are many cases where entire program is not needed in main memory at a time.
- In many cases even though entire program is needed, it may not all be needed at the same time.
- Application programs always get the feeling of availability of contiguous working address space due to the idea of virtual memory.
- Actually, this working memory can be physically fragmented and may even overflow on to disk storage.
- This technique makes programming of large applications easier and use real physical memory more efficiently than those without virtual memory.
  - Although executing process is not entirely present in memory, it can complete its execution due to virtual memory technique.
  - Virtual memory permits execution of larger size programs although smaller size physical memory is available.
  - It means larger size programs than available physical memory can complete the execution.
  - Virtual memory concept separate the user logical memory from actual physical memory.
  - This separation offers very large virtual memory to programmers although actual physical memory is having very small size.

Syllabus Topic
Hardware and Control Structures - Demand Paging

4.13 Hardware and Control Structures

4.13.1 Demand Paging

→ (Dec. 14, Dec. 16)

Q. Write short note on Demand Paging.
MU - Dec. 2014, Dec. 2016. 5 Marks

Q. What is demand paging? Explain its advantages and disadvantages.

- A demand paging is paging system with swapping where pages are brought in main memory from secondary storage on demand.
- When it is needed to execute a process, memory manager swap it into memory.
- Instead of swapping the entire process into memory, demand paging allows to swap in only those pages which are required for execution at that time.
- There will be failure in accessing the page, if it is not presently mapped to a frame. This will cause a page fault which is a special type of interrupt.
- To handle this interrupt, disk operation is initiated by OS handler to read the required page in memory.
- After this, mapping of page to free frame is carried out. During all this operation the process remains blocked.
- As soon as page becomes available, the blocked process during disk operation is now unblocked, and placed on the ready queue.
- When scheduler schedules this process, it restarts its execution with the instruction that caused the page fault.
- The execution will continue until all the instructions are in memory. This approach of fetching pages as they are needed for execution is called demand paging.
- Before swapping in the process in memory, the pager program makes sure about which pages will be used before the process is swapped out again.
- The complete process is not swapped in. Instead, the pager brings only those required pages into memory.
- This would restrict the swapping in of pages in memory which are not needed for execution.

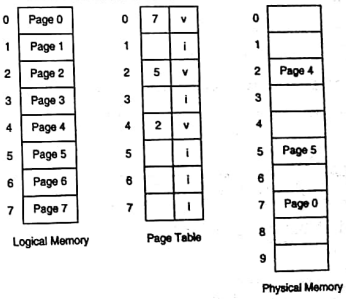


Fig. 4.13.1 : Page table with valid and invalid pages

- It saves swap time and the amount of physical memory needed.
- In demand paging, valid and invalid bits are used to differentiate between those pages that are in memory and those pages that are on the disk.
- Demand paging keep more processes in memory than the sum of their memory needs ensuring CPU utilization as high as possible.
- Fig. 4.13.1 shows page table with valid and invalid pages. Those pages which are in main memory are marked as valid. Pages on secondary storage are marked as invalid.
- Page 0, Page 2 and Page 4 are in main memory. So it is marked with valid (v) bit. Page 1 and 3 are not in main memory so marked with invalid (i) bit. Page 1, page 3, Page 5, page 6, and page 7are on secondary storage.
- Continuous allocation of memory to the pages is not necessary.
- The reason is, any page can be mapped into any available page frame.
- The page table maintain the information regarding mapping of page to page frame. Due to this, it offers the impression of having one contiguous block of memory.

Advantages of Demand Paging

- Provides large size virtual memory.
- Memory is utilized more efficiently.
- Support for degree of multiprogramming is very good.

Disadvantages of Demand Paging

- Page interrupts requires more number of table handling and processor overhead with compare to simple paged management techniques.

Syllabus Topic : Structure of Page Tables

4.14 Structure of Page Tables

→ (Dec. 14)

Q. Discuss various techniques for structuring page tables. MU - Dec. 2014. 10 Marks.

Following are the most common techniques for structuring the page table.

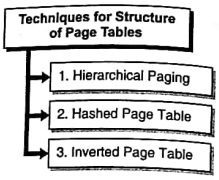
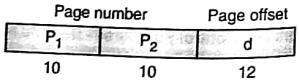


Fig. C4.3 : Techniques for structuring the page table

→ 4.14.1 Hierarchical Paging

- If the logical address space is very large such as 2^{32} or 2^{64}, the page table itself becomes extremely large.
- In this case each process may require may large physical address space for the page table alone. So it would not be advantageous to allocate the page table contiguously in main memory.
- Solution to this problem is to split the page table into smaller pieces.
- One method to achieve this is to use a two-level paging algorithm, in which the page table itself is also paged.
- Consider the system with 32 bit logical address space with page size 4 KB. A 32 bit logical address is divided in :
  - 20 bit page number and
  - 12 bit page offset
- Since paging of the page tables are done, above 20 bit page number is again subdivided in :
  - 10 bit page number and
  - 10 bit page offset
- Thus the 32 bit logical address is divided as follows :



Where,

- P_1 is an index into the outer page table and
- P_2 is the displacement within the page of the outer page table.
- The address-translation scheme for this architecture is shown in following Fig. 4.14.1. As address translation

works from the outer page table inward, this scheme is also known as a forward-mapped page table.

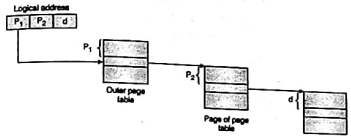


Fig. 4.14.1 : Address translation for a two-level 32-bit paging architecture

- Fig. 4.14.2 shows a two-level page-table scheme.

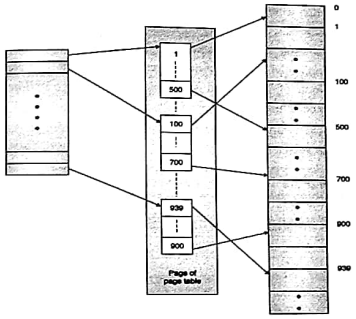


Fig. 4.14.2 : A two-level page-table scheme

→ 4.14.2 Hashed Page Table

- If the address spaces are larger than 32 bits then hashed page table is used in which hash value is used as virtual page number.
- The elements which are hashed at same location, its linked list is maintained at each entry in the hash table. It is necessary to avoid the collision. Each element consists of three fields :
  1. The virtual page number
  2. The value of the mapped page frame
  3. A pointer to the next element in the linked list.

  The working of the algorithm is as follows :
- Virtual address contains the virtual page number. This virtual page number is hashed into the hash table.
- The comparison of virtual page number and the field 1 in the first element in the linked list is performed.
- If match occurs then the corresponding page frame (field 2) is used to form the desired physical address.
- If there is no match then succeeding entries in the linked list are searched for a matching virtual page number. This scheme is shown in Fig. 4.14.3.
- A variant of this method that is good for 64-bit address spaces has been proposed.
- In this, each entry in the hash table refers to a number of pages (such as 16) rather than a single page just like hashed page tables.
- Due to this, a single page-table entry can store the mappings for many physical-page frames.
- A grouped page tables are mainly useful for address spaces, where memory references are noncontiguous and scattered throughout the address space.

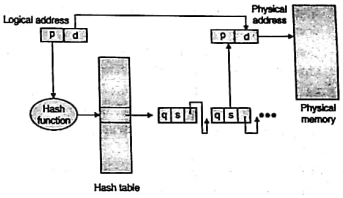


Fig. 4.14.3 : Hashed Page Table

→ 4.14.3 Inverted Page Table

- Generally the operating system converts the page reference into a physical memory address.
- Because the table is sorted by virtual address, the operating system is capable of computing where in the table the associated physical address entry is located and to use that value directly.
- The disadvantages of this scheme are that each page table may contain millions of entries.
- These tables may use large amounts of physical memory just to monitor how other physical memory is being used.
- The problem is resolved with inverted page tables.
- There is one entry for each real page (or frame) of memory in inverted page table. In this entry the virtual address of the page stored in that real memory location is present.
- Along with this virtual address, entry also contains the information about the process containing this page.

- Because of this only one page table is in the system, and it has only one entry for each page of physical memory. Fig. 4.14.4 shows the operation of an inverted page table.

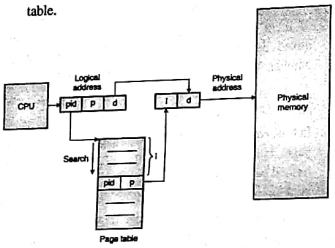


Fig. 4.14.4: Inverted Page Table

- In order to explain this approach, example of the inverted page table used in the IBM RT is demonstrated.
- Every virtual address in the system is composed of a triple: <process-id, page-number, offset>.
- As shown in Fig. 4.14.4 inverted page-table entry contains a pair <process-id, page-number>.
- The process-id carry out the work as a address-space identifier.
- At the time memory reference take place, part of the virtual address, consisting of <process-id, page-number>, is given to the memory subsystem.
- Searching of the inverted page table is performed for a match. If there is a match at entry i-then the physical address <i, offset> is produced.
- If there is no match, then an illegal address access has been tried.
- In this scheme the amount of memory needed to store each page table is saved, but the amount of time needed to search the table when a page reference occur increases.
- The entire table might need to be searched to get a match. This search would consume more time.
- To ease this problem, a hash table can be used limit the search to one-or at most few-page-table entries.

Syllabus Topic : Copy-on-Write

4.15 Copy-on-Write

- The fork() system call is used to create child process which is a duplicate of its parent. Usually, fork() system call worked by creating a copy of the parent's address space for the child.
- While doing so it also duplicates the pages belonging to the parent. On the other hand, if we assume, many child processes call upon the exec() system call straight away after creation, the copying of the parent's address space may be needless.
- Copy-on-Write allows parent and child to share the same pages. If parent or child writes to shared pages then it is marked as copy-on-write and its copy is created.
- All unmodified pages can be shared by the parent and child processes.
- Pages that cannot be modified such as pages containing executable code can be shared by the parent and child. This technique used by Windows XP, Linux, and Solaris.
- As soon as it is known that a page is going to be duplicated using copy-on-write, it is significant to make a note of the location from which the free page will be allocated.
- Several operating systems provide a pool of free pages for such requests.

Syllabus Topic : Page Replacement Strategies

4.16 Page Replacement Strategies

→ (Dec. 14)

Q. Write short note on various page replacement policies. MU - Dec. 2014, 5 Marks

- After page fault handling, to accommodate the new page, it is not possible that frames will remain always free.
- So it is needed to replace the existing page from main memory.
- For that purpose there are many page replacement algorithms.
- All these algorithms can be evaluated by running it on a particular string of memory reference. It is expected that the number of page faults should be minimum to get the performance.
- Reference string is the string of memory references. The generation of the reference string is carried out artificially.
- The reference string can also be generated by listing the address of each memory reference by tracing the system.

- The important factor to determine number of page faults is the number of page frames available.
- These numbers of page faults are for a particular reference string and for page replacement algorithm for given available number of page frames.
- The number of frames available is inversely proportional to number of page faults. It means that if more frames are available less number of faults will occur.

Syllabus Topic : FIFO

4.16.1 FIFO Algorithm

Q. Compare Optimal, LRU and FIFO page replacement algorithms with Illustration.
Q. Compare FIFO and LRU page replacement algorithm.

- The simplest page-replacement algorithm and work on the basis of first in first out (FIFO). It throws out the pages in the order in which they were brought in.
- The time is associated with each page when it was brought into main memory.
- This algorithm always chooses oldest page for replacement.
- Since replacement is FIFO, a queue can be maintained to hold all the pages in main memory.
- This algorithm doesn't care about which pages are accessed frequently and which are not. However, it is used in windows 2000.

Consider the reference string 5, 0, 2, 3, 0, 1, 3, 4, 5, 4, 2, 0, 3, 4, 3 and consider 3 available frames.

Reference string

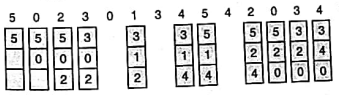


Fig. 4.16.1 : FIFO page replacement algorithm

- Since first three pages were initially not in main memory, references (5, 0, 2) causes page faults and brought into these 3 free frames. Reference to page 3 again causes the fault.
- To bring page 3 in memory as per FIFO policy, page 5 is replaced. Next reference is to page 0 and it is already in memory.
- To bring page 1 in memory again as per FIFO policy page 0 is replaced. Page 3 is already in memory. Reference to page 4 again causes the fault.
- To bring page 4 in memory as per FIFO policy, page 2 is replaced. This process continues as shown in Fig. 4.16.1.
- Whenever faults occurred, it is shown that which pages are in 3 frames.
- There are in total 11 faults occurred together. This algorithm sometimes suffers through result called as Belady's anomaly : That is page fault rate will increase even though number of allocated frames increases.

Syllabus Topic : Optimal

4.16.2 Optimal Page Replacement Algorithm

Q. Compare Optimal, LRU and FIFO page replacement algorithms with Illustration.

- Compare to all other page-replacement algorithms, this algorithm has the lowest page-fault rate.
- This algorithm replaces that page which will not be used for the longest period of time.
- This delays the unavoidable page fault as much as possible, and thus decreases the total number of page faults.
- As algorithm replace the page on the basis of future use of page, it is impossible to implement.
- The optimal algorithm is unrealizable because it is impossible to determine the number of instructions that will be executed in the future before the page will be referenced.
- This algorithm does not suffer through Belady's anomaly.

Again consider the same reference string 5, 0, 2, 3, 0, 1, 3, 4, 5, 4, 2, 0, 3, 4, 3

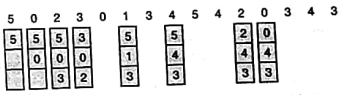


Fig. 4.16.2 : Optimal page replacement algorithm

- Since first three pages were initially not in main memory, references (5, 0, 2) causes page faults and brought into these 3 free frames.
- Reference to page 3 again causes the fault. To bring page 3 in memory as per optimal replacement policy, page 2 is replaced because it is the page which will not be used for longest period of time.
- Next reference is to page 0 and it is already in memory. To bring page 1 in memory again as per optimal replacement policy, page 0 is replaced. Next reference is to page 0 and it is already in memory.

- Reference to page 4 again causes the fault. Since page 1 will no longer be referenced, it is replaced. This process continues as shown in Fig. 4.16.2 causing in total 8 page faults.

Syllabus Topic : LRU

4.16.3 Least Recently Used (LRU) Page Replacement Algorithm

Q. Compare Optimal, LRU and FIFO page replacement algorithms with Illustration.
Q. Compare FIFO and LRU page replacement algorithm.

- The time of page's last use is associated with each page.
- When a page must be replaced, LRU chooses that page that was used farthest back in the past.
- LRU is a good approximation to the optimal algorithm.
- This algorithm looks backward in time while optimal replacement algorithm looks forward in time.
- This policy suggests that replace a page whose last usage is farthest from current time.
- This algorithm can be implemented with some hardware support and is considered to be a good solution for page replacement.
- This algorithm does not suffer through Belady's anomaly.
- We will again consider the same reference string 5, 0, 2, 3, 0, 1, 3, 4, 5, 4, 2, 0, 3, 4, 3. Following is the result of applying LRU page replacement algorithm.

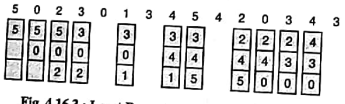


Fig. 4.16.3 : Least Recently Used page replacement algorithm

- Since first three pages were initially not in main memory, references (5, 0, 2) causes page faults and brought into these 3 free frames. Reference to page 3 again causes the fault.
- To bring page 3 in memory as per optimal replacement policy, page 5 is replaced because it is the page which is used least recently.
- Next reference is to page 0 and it is already in memory.
- To bring page 1 in memory again as per LRU replacement policy, page 2 is replaced. Next reference is to page 3 and it is already in memory. This process continues as shown in Fig. 4.16.3 causing in total 11 page faults.

Implementation of LRU

Using Stack

- Initially keep all the page numbers on the stack.
- Remove the page from stack whenever it is referenced and place it on the top of the stack.
- Any time top of stack shows latest page number that is referenced and bottom shows the page which is not used for longest period of time.

Using Counters

- In a page table add time of use field with each page.
- At each reference to page a CPU logical clock is incremented.
- Copy clock register content to time of use field.
- Time of use field will show the time of last reference to the page.
- Replace the page having smallest time value.

4.16.4 LRU-Approximation Page Replacement

- Not all the system provides hardware for LRU replacement. If hardware support not available then FIFO algorithm is used.
- The reference bit is associated with each page and stored in page table.
- If page is referenced for read or write then this bit is set. Initially this bit is set to 0 by OS. Once page is referenced then set to 1.
- This information is used for LRU-Approximation Page Replacement.

4.16.4 LRU-Approximation Page Replacement

4.16.4(A) Additional-Reference-Bits Algorithm

- In this algorithm, 8 bit byte is associated with page and maintained in table in memory.
- After regular interval, OS is interrupted which then shifts the reference bit for each page into the high-order bit of its 8-bit byte.
- Other bits are shifted right by 1 and low order bit is discarded. This 8 bit shift register shows the history of page use in last 8 time period.
- For example, following values of shift register shows the use of page as shown below.
  00000000 - Not used for last 8 time period
  11111111 - Used at least once in each period
  11000100 – This page is used more recently than one with a shift register value of 01110111.
- The least recently used page is the lowest number page when this shift register value is considered as unsigned integer.
- The number of bits used is hardware dependent. If single bit used then it is second chance replacement.

4.16.4(B) Second-Chance Algorithm

- The FIFO algorithm is the basic algorithm of second-chance replacement algorithm.
- After selecting page for replacement, its reference bit is checked. If the reference bit is 0 (old and unused page), then page is replaced.

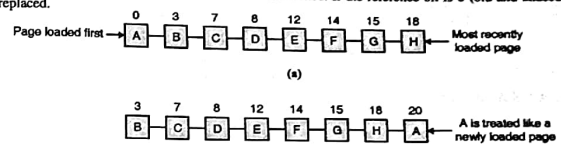


Fig. 4.16.4 : Second Chance Replacement

- If the reference bit is set to 1, a second chance is given to the page and then preceded on to select the next FIFO page. After a page gets a second chance, its reference bit is cleared.
- Now this page is considered as arrived currently and its arrival time is reset to the current time.
- Because of this, page that got a second chance will not be replaced until all other pages have been replaced (or given second chances).
- In addition, if a page is used often enough to keep its reference bit set, it will never be replaced.
- Page A was initially loaded at time 0 and given a second chance. Its arrival time is now set to current time 20.

4.16.4(C) Enhanced Second-Chance Algorithm

This algorithm considers pair of reference bit and modify bit. We get 4 combinations with two bits as:

- (0, 0) – It is the best page for replacement which is neither recently used nor modified.
- (0, 1) – It indicates page is not good for replacement as it is not referenced but modified. It need to be written out before replacement.
- (1, 0) – This page may be used again as it shows recently used but not modified.
- (1, 1) – This page is recently used and modified. This page may be used again and hence need to be written out to disk before it is replaced.

4.16.4(D) Clock Page Replacement Algorithm

- Clock page replacement algorithm maintains all the page frames on a circular list in the form of a clock.
- The clock hand points to the oldest page.
- After the occurrence of page fault, the page being pointed to by the hand is checked.
- If its reference bit is 0, the page is replaced and the new page is inserted into the clock in its place.
- After this insertion of new page, the hand is advanced one position.
- If reference bit is 1, it is cleared and the hand is advanced to the next page.
- This process is repeated until a page is found with reference bit equal to 1.

Syllabus Topic : LFU

4.16.5 Counting-Based Page Replacement

4.16.5(A) Not Frequently Used or Least Frequently Used Page Replacement Algorithm (NFU or LFU)

Q. Explain counting based approach.

- This policy assumes that, those pages which are referenced for more number of times will be needed again.
- A counter is used for each page and incremented with each memory reference.
- A page with less counter value should be replaced. Suppose that process P has not made use of page 2 and other pages have a count of usage like 3, 4 or even 5 times.
- So the basic argument is that these pages may still be needed as compared to the page at 2. So page 2 should be replaced which is having less count.

– The limitation of the algorithm is that, if initially page is used frequently and then there is no reference to this page, still it will remain in memory.

4.16.5(B) Most Frequently Used Page Replacement Algorithm (MFU)

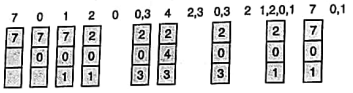
It is based on the assumption that, page having less count just arrived in memory and it is yet to be used.

4.17 Examples on Page Replacement Algorithms

Example 4.17.1

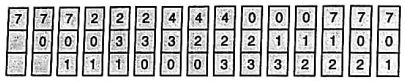
For the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0. 1. 7, 0, 1. Calculate the Page Faults applying (i) Optimal (ii) LRU and (iii) FIFO Page Replacement Algorithms for a Memory with three frames.

Solution : Optimal Algorithm



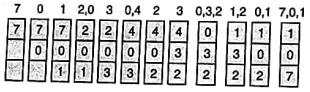
Total Page faults = 09

FIFO Algorithm : String = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0. 1. 7, 0, 1



Total Page faults = 15

LRU Algorithm : String = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0. 1. 7, 0, 1



Total Page faults = 12

Example 4.17.2

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 four frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

(1) LRU replacement (2) FIFO replacement

Solution :

(1) LRU

:

String 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6.

1 1 1 1 1 1 1 1 6 6
2 2 2 2 2 2 2 2
3 3 5 5 3 3 3
4 4 6 6 7 7 1

Page faults for LRU = 10

(2) FIFO

String 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6.

1 1 1 1 5 5 5 5 3 3 3 3 1 1
2 2 2 6 6 6 6 7 7 7 7 3
3 3 3 3 2 2 2 2 6 6 6 6
4 4 4 4 1 1 1 1 2 2 2

Page faults for FIFO = 14

Example 4.17.3

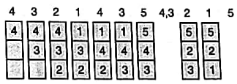
Consider following page reference string

4,3,2,1,4,3,5,4,3,2,1,5

Assume frame size = 3. How many page faults would occur for FIFO, Optimal, LRU algorithm ?

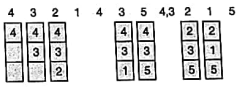
Solution :

(1) FIFO page replacement algorithm



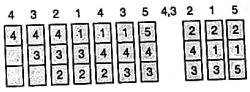
Total number of page faults = 9

(2) Optimal page replacement algorithm



Total number of page faults = 7

(3) LRU page replacement algorithm



Total number of page faults = 10

Example 4.17.4 MU - Nov. 2014. 10 Marks

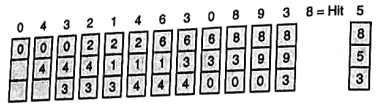
Calculate Hit and Miss using LRU, optimal, FIFO page replacement policies for the following sequence.

Page frame size is 3.0,4,3,2,1,4,6,3,0,8,9,3,8,5

Solution :

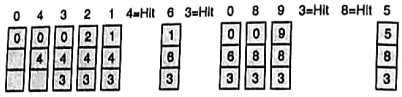
Consider frame size = 3

(1) LRU



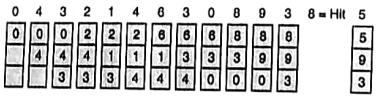
Number of hits = 01, Miss = 13

(2) Optimal



Number of hits = 04, Miss = 10

(3) FIFO



Number of hits = 01, Miss = 1

Example 4.17.5

Calculate hit and miss percentage for the following string using page replacement policies FIFO, LRU and Optimal. Compare it for the frame size 3 and 4. Page string 2,0,3,0,4, 2,3,0,3,2,7,2,0,7,5,0,7,5,7,0

Solution:

Frame Size = 3

FIFO Page Replacement

Number of Hits = 8 and Miss = 12

Frame	2	0	3	0	4	2	3	0	3	2	7	2	0	7	5	0	7	5	7	0
0	2	2	2	2	4	4	4	4	3	3	3	3	0	0	0	0	0	0	0	0
1		0	0	0	0	2	2	2	2	7	7	7	7	5	5	5	5	5	5	5
2			3	3	3	3	3	0	0	0	0	2	2	2	2	2	7	7	7	7
PF	Y	Y	Y	-	Y	Y	-	Y	Y	-	Y	Y	Y	-	Y	-	Y	-	-	-

Optimal Page Replacement

Number of Hits = 13 and Miss = 7

Frame	2	0	3	0	4	2	3	0	3	2	7	2	0	7	5	0	7	5	7	0
0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	5	5	5	5	5	5
1		0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0	0
2			3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7
PF	Y	Y	Y	-	Y	-	-	Y	-	-	Y	-	-	-	Y	-	-	-	-	-

LRU Page Replacement

Number of Hits = 10 and Miss = 10

Frame	2	0	3	0	4	2	3	0	3	2	7	2	0	7	5	0	7	5	7	0
0	2	2	2	2	4	4	4	0	0	0	7	7	7	7	7	7	7	7	7	7
1		0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0	0	0	0
2			3	3	3	2	2	2	2	2	2	2	2	5	5	5	5	5	5	5
PF	Y	Y	Y	-	Y	Y	Y	Y	-	-	Y	-	Y	-	Y	-	-	-	-	-

Frame Size = 4

FIFO Page Replacement

Number of Hits = 12 and Miss = 8

Frame	2	0	3	0	4	2	3	0	3	2	7	2	0	7	5	0	7	5	7	0
0	2	2	2	2	2	2	2	2	2	2	7	7	7	7	7	7	7	7	7	7
1		0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2
2			3	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0
3					4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
PF	Y	Y	Y	-	Y	-	-	-	-	-	Y	Y	Y	-	Y	-	-	-	-	-

Frame Size = 4

Optimal Page Replacement

Number of Hits = 14 and Miss = 6

Frame	2	0	3	0	4	2	3	0	3	2	7	2	0	7	5	0	7	5	7	0
0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2			3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7
3					4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
PF	Y	Y	Y	-	Y	-	-	-	-	-	Y	-	-	-	Y	-	-	-	-	-

Frame Size = 4

LRU Page Replacement

Number of Hits = 14 and Miss = 6

Frame	2	0	3	0	4	2	3	0	3	2	7	2	0	7	5	0	7	5	7	0
0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2			3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3					4	4	4	4	4	4	7	7	7	7	7	7	7	7	7	7
PF	Y	Y	Y	-	Y	-	-	-	-	-	Y	-	-	-	Y	-	-	-	-	-

When frame size = 4 then number of hits are more all algorithms for given page string.

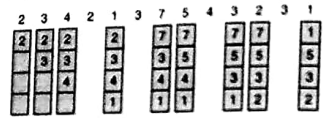
Example 4.17.6 MU - June 2015, 10 Marks

What is paging. Explain LRU, FIFO, and OPT page replacement policy for the given page sequences. Page frame size is 4.

2, 3, 4, 2, 1, 3, 7, 5, 4, 3, 2, 3, 1

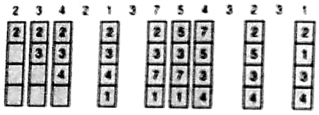
Calculate page hit and miss.

Solution : (i) FIFO



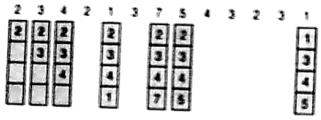
For FIFO Hit = 4 and Miss = 9

(ii) LRU



For LRU Hit = 4 and Miss = 9

(iii) Optimal



For Optimal Hit = 6 and Miss = 7

Example 4.17.7 MU - Dec. 2015 - 10 Marks

Calculate hit and miss for the following string using page replacement policies-FIFO, LRC and Optimal. Compare it for its frame size 3 & 4.

1 2 3 2 1 5 2 1 6 2 5 6 3 1 3 6 1 2 4 3

Solution :

Frame Size = 3

(i) FIFO

Frame	1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
0	1	1	1	1	1	5	5	5	5	2	2	2	2	1	1	1	1	1	4	4
1		2	2	2	2	2	2	1	1	1	5	5	5	5	5	6	6	6	6	3
2			3	3	3	3	3	3	6	6	6	6	3	3	3	3	3	2	2	2
PF	Y	Y	Y	-	-	Y	-	Y	Y	Y	Y	-	Y	Y	-	Y	-	Y	Y	Y

(ii) LRU

Frame	1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
0	1	1	1	1	1	1	1	1	1	1	5	5	5	1	1	1	1	1	1	3
1		2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	2	2	2
2			3	3	3	5	5	5	6	6	6	6	6	6	6	6	6	6	4	4
PF	Y	Y	Y	-	-	Y	-	-	Y	-	Y	-	Y	Y	-	-	-	Y	Y	Y

(iii) Optimal

Frame	1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
0	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	2	4	4
1		2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
2			3	3	3	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3
PF	Y	Y	Y	-	-	Y	-	-	Y	-	-	-	Y	Y	-	-	-	Y	Y	-

Frame Size = 4

(i) FIFO

Frame	1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
0	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	6	6	6
1		2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
2			3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2
3						5	5	5	5	5	5	5	5	5	5	5	5	5	4	4
PF	Y	Y	Y	-	-	Y	-	-	Y	-	-	-	Y	-	-	-	Y	Y	Y	

(ii) LRU

Frame	1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
0	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	4	4
1		2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
2			3	3	3	3	3	3	6	6	6	6	6	6	6	6	6	6	6	3
3						5	5	5	5	5	5	5	5	5	5	5	5	2	2	2
PF	Y	Y	Y	-	-	Y	-	-	Y	-	-	-	Y	Y	-	-	-	Y	Y	Y

(iii) Optimal

Frame	1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
0	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	6	4	4
1		2	2	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	3
3						5	5	5	5	5	5	5	1	1	1	1	1	1	1	1
PF	Y	Y	Y	-	-	Y	-	-	Y	-	-	-	Y	-	-	-	-	Y	-	

Frame Size =		Hit	Miss	Frame Size =		Hit	Miss
3	FIFO	6	14	4	FIFO	11	9
	LRU	9	11		LRU	10	10
	OPTIMAL	11	9		OPTIMAL	13	7

Example 4.17.8 MU - Dec. 2015. 10 Marks

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 three, five frames for LRU, FIFO and optimal page replacement?

Solution :

Frame size = 3

i) LRU gives 15 page faults ii) FIFO, 16 page faults iii) Optimal, 11 page faults

The "Y" in bottom rows denote the page fault

(i) LRU

Frame	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1	1	1	4	4	4	5	5	5	1	1	1	7	7	7	2	2	2	2	2
1		2	2	2	2	2	2	6	6	6	6	3	3	3	3	3	3	3	3	3
2			3	3	3	1	1	1	2	2	2	2	2	6	6	6	1	1	1	6
PF	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	-	Y	Y	Y	-	Y	Y	-	-	Y

(ii) FIFO

Frame	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1	1	1	4	4	4	4	6	6	6	6	3	3	3	3	2	2	2	2	6
1		2	2	2	2	1	1	1	2	2	2	2	7	7	7	7	1	1	1	1
2			3	3	3	3	5	5	5	1	1	1	1	6	6	6	6	6	3	3
PF	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	-	Y	Y	Y	-	Y	Y	-	Y	Y

(iii) Optimal

Frame	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	3
1		2	2	2	2	2	2	2	2	2	2	2	7	7	7	2	2	2	2	2
2			3	4	4	4	5	6	6	6	6	6	6	6	6	6	1	1	1	1
PF	Y	Y	Y	Y	-	-	Y	Y	-	-	-	Y	Y	-	-	Y	Y	-	-	Y

If frame size = 5 then

i) LRU gives 8 page faults ii) FIFO, 10 page faults iii) Optimal, 7 page faults

Bold page number shows the fault.

(i) LRU

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

(ii) FIFO

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

(iii) Optimal

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Example 4.17.9 MU - Dec 2016. 10 Marks

Calculate number of page faults and page hits for the page replacement policies FIFO, Optimal and LRU for given reference string 6, 0, 5, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 5, 2, 0, 5, 6, 0, 5 (assuming three frame size).

Solution : Frame Size = 3

FIFO Page Replacement

Number of Hits = 5 and Miss = 15

Frame	6	0	5	2	0	3	0	4	2	3	0	3	2	5	2	0	5	6	0	5
0	6	6	6	2	2	2	2	4	4	4	0	0	0	0	0	0	0	6	6	6
1		0	0	0	0	3	3	3	2	2	2	2	2	5	5	5	5	5	0	0
2			5	5	5	5	0	0	0	3	3	3	3	3	2	2	2	2	2	5
PF	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	-	-	Y	Y	-	-	Y	Y	Y

Optimal Page Replacement

Number of Hits = 12 and Miss = 8

Frame	6	0	5	2	0	3	0	4	2	3	0	3	2	5	2	0	5	6	0	5
0	6	6	6	2	2	2	2	2	2	2	2	2	2	2	2	2	2	6	6	6
1		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
2			5	5	5	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5
PF	Y	Y	Y	Y	-	-	-	Y	-	-	Y	-	-	Y	-	-	-	Y	-	-

LRU Page Replacement

Number of Hits = 8 and Miss = 12

Frame	6	0	5	2	0	3	0	4	2	3	0	3	2	5	2	0	5	6	0	5
0	6	6	6	2	2	2	2	4	4	4	0	0	0	5	5	5	5	5	5	5
1		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
2			5	5	5	3	3	3	2	2	2	2	2	2	2	2	2	6	6	6
PF	Y	Y	Y	Y	-	Y	-	Y	Y	Y	Y	-	-	Y	-	Y	-	Y	-	-

Syllabus Topic : Allocation of Frames

4.18 Allocation of Frames

Q. Explain different strategies for allocation of frames.

- Operating system maintains free frame list. Let us take an example of single-user system with 512 KB of memory consisting of pages 4 KB in size.
- This system has 128 frames each of 4 KB in size. The operating system may occupy 35 frames and remaining 93 frames are for the user process.
- If we consider the pure demand paging, all 93 frames would at the start be put on the free-frame list.
- Once user process begins execution, it would produce a sequence of page faults.
- The initial 93 page faults would all get free frames from the free-frame list.
- When the free-frame list was worn out, a page-replacement algorithm would be used to select one of the 93 in-memory pages to be replaced with the 94th, and so on.
- If the more number of frames allocated to process, the page-fault rate decreases.
- If less number of frames allocated to each process, the page-fault rate increases, slowing process execution.
- If page fault occurs prior to an executing instruction is complete, the instruction must be started again.
- As a result, we must have sufficient frames to hold all the different pages that any single instruction can reference.

Equal allocation

- Consider p number of processes and q number of frames.

- In this case every process will get p/q frames. In our example suppose 5 processes and 93 frames present. Then 93/5, that is each process will get 18 frames and 3 will remain at free-frame list.

☞ **Proportional allocation**

- In this allocation, frames are allocated to the processes as per their needs. Suppose 60 free frames available. Process P1 has size 10 KB and P2 has size 128 KB.
- And if page size is 1 KB. Then (10/(10+128)) × 60 = 4.34, say nearly 4 frames allocated to P1. P2 will get (128/(10+128)) × 60 = 55.65, that is 55 frames.

☞ **Global vs Local allocation**

- Page replacement algorithm is categorized as global replacement if it can choose frame for replacement from all frames although currently it is allocated to other process also.
- It means processes can take frames from other processes. Local replacement needs to replace frame by process only from its own set of allocated frames.
- The difficulty with a global replacement algorithm is that a process cannot control its own page-fault rate.
- The set of pages in memory for a process depends not only on the paging actions of that process but also on the paging actions of other processes. Consequently, the same process may perform fairly differently.

Syllabus Topic : Thrashing

4.19 Thrashing

→ **(June 15)**

Q. What is thrashing ? **MU - June 2015, 5 Marks**

- A process should have some minimum number of frames to support active pages which are in memory.
- It helps to reduce the number of page faults. If these numbers of frames are not available to the process then it will quickly page fault.
- To handle this page fault, it is necessary to replace the existing page from memory.
- Since all the pages of the process are active, it will also need in future. So any replaced page will cause page fault again and again.
- Since in paging, pages are transferred between main memory and disk, this has an enormous overhead.
- Because of this frequent movement of pages between main memory and disk, system throughput reduces.
- This frequent paging activity causing the reduction in system throughput called as thrashing.
- Although many processes are in memory, due to thrashing CPU utilization goes low.
- When operating system monitors this CPU utilization, it introduces new process in memory to increase the degree of multiprogramming.
- Now it is needed that the existing pages should be replaced for this new process.
- If global page replacement algorithm is used, it replaces the pages of other process and allocates frames to this newly introduced process. So other processes whose pages are replaced are also causes page faults.
- All these faulting processes go in wait state and waits for paging device. In this case again CPU utilization goes low.
- When CPU scheduler observes the low CPU utilization, it introduces new processes again in system to improve degree of multiprogramming. This also results in page faults of running processes.
- Again the waiting queue of paging device becomes long causing further drop in CPU utilization. In this way page fault rate further increases and CPU utilization decreases.
- There is no actual work getting done and processes spend time only in paging.
- This thrashing can be limited by using local page replacement algorithm instead of global page replacement algorithm.
- Local page replacement algorithm replaces the pages of same process instead of other process. Therefore frames of the other processes will not be taken and these processes will not thrash.

4.20 Locality (Working Set Model)

- Thrashing can be prevented by providing sufficient frames to the process as per its need.
- Locality model of process execution solve this problem by keeping track on number of frames process is currently using. Locality is the set of active pages used together.
- During execution process moves from locality to locality.

- Working Set Model is based on this assumption of locality. It examines most recent page references.
- All the pages in most recent page references are the working set. Suppose working set window size defined is 8. At time t1 if the page references are (2, 3, 2, 3, 4, 3, 3, 4) then working set is {2,3,4} and 3 frames are required.
- At this time if demand of the process is more that working set, thrashing will occur. Operating system will monitor working set of the process and allocate needed frames.
- If sufficient frames are not available it will suspend the process. In this way working set model helps to prevent the thrashing.

4.21 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Memory Management Requirements**

Q. What are different requirements for memory management ? *(Refer section 4.1)*

Q. Write short note on Relocation. *(Refer section 4.1.1)*

Q. With the help of example, clearly differentiate between logical and physical address space. *(Refer section 4.1.2)*

☞ **Syllabus Topic : Memory Partitioning - Fixed Partitioning**

Q. Explain memory management with Fixed and Variable Partitions. *(Refer section 4.2.3)*

Q. What is internal fragmentation and external fragmentation? Explain with example. *(Refer section 4.2.3)*

☞ **Syllabus Topic : Memory Partitioning - Dynamic Partition**

Q. What is compaction? Explain with example. *(Refer section 4.2.5)*

Q. Explain Swapping in detail. *(Refer section 4.5)*

☞ **Syllabus Topic : Memory Allocation Strategies**

Q. Explain different memory allocation strategies. *(Refer section 4.6)*

Q. Discuss partition selection algorithm in brief. *(Refer section 4.6)*

☞ **Syllabus Topic : Buddy System**

Q. Explain Buddy system. *(Refer section 4.7)*

☞ **Syllabus Topic : Paging**

Q. What is paging ? *(Refer section 4.9)* (5 Marks) **(June 2015)**

Q. What is Page Table? Explain the conversion of Virtual Address to Physical Address in Paging with example. *(Refer section 4.9.1)*

Q. Explain the hardware support for paging. *(Refer section 4.9.4)* (10 Marks) **(Dec. 2016)**

☞ **Syllabus Topic : Segmentation**

Q. Explain segmentation with paging. *(Refer section 4.10)*

Q. What is segmentation? Explain it with example. *(Refer section 4.10)*

Q. Explain the difference between paging and Segmentation. (Refer section 4.10.1) (5 Marks) **(Dec. 2016)**

☞ **Syllabus Topic : Virtual Memory**

Q. Write short note on Virtual Memory. *(Refer section 4.12)* (5 Marks) **(May 2016)**

☞ **Syllabus Topic : Hardware and Control Structures - Demand Paging**

Q. Write short note on Demand Paging. *(Refer section 4.13.1)* (5 Marks) **(Dec. 2014, Dec. 2016)**

Q. What is demand paging? Explain its advantages and disadvantages. *(Refer section 4.13.1)*

☞ **Syllabus Topic : Structure of Page Tables**

Q. Discuss various techniques for structuring page tables. (Refer section 4.14) (10 Marks) **(Dec. 2014)**

☞ **Syllabus Topic : Page Replacement Strategies**

Q. Write short note on various page replacement policies. *(Refer section 4.16)* (5 Marks) **(Dec. 2014)**

☞ **Syllabus Topic : FIFO**

Q. Compare Optimal, LRU and FIFO page replacement algorithms with Illustration. *(Refer section 4.16.1)*

Q. Compare FIFO and LRU page replacement algorithm. *(Refer section 4.16.1)*

☞ **Syllabus Topic : Optimal**

Q. Compare Optimal, LRU and FIFO page replacement algorithms with illustration. *(Refer section 4.16.2)*

☞ **Syllabus Topic : LRU**

Q. Compare Optimal, LRU and FIFO page replacement algorithms with illustration. *(Refer section 4.16.3)*

Q. Compare FIFO and LRU page replacement algorithm. *(Refer section 4.16.3)*

☞ **Syllabus Topic : LFU**

Q. Explain counting based approach. *(Refer section 4.16.5(A))*

Example 4.17.4 (10 Marks) **(Nov. 2014)**

Example 4.17.6 (10 Marks) **(June 2015)**

Example 4.17.7 (10 Marks) **(Dec. 2016)**

Example 4.17.8 (10 Marks) **(Dec. 2015)**

Example 4.17.9 (10 Marks) **(Dec. 2016)**

☞ **Syllabus Topic : Allocation of Frames**

Q. Explain different strategies for allocation of frames. *(Refer section 4.18)*

☞ **Syllabus Topic : Thrashing**

Q. What is thrashing ? *(Refer section 4.19)* (5 Marks) **(June 2015)**

□□□

CHAPTER 5

Module 5

File Management

Syllabus Topics

Overview, File Organization and Access, File Directories, File Sharing, Secondary Storage Management, Linux Virtual File System.

Syllabus Topic : File Management - Overview

5.1 Overview

5.1.1 Files and File Systems

- Every application needs to access and store the information. During execution, process can keeps the information in its address space.
- This address space to store information is sufficient for some type of applications but not for others, which requires huge size of information.
- After finishing the execution, process terminates and information also vanishes.
- For many applications, it is required that information should be retained forever.
- Even though computer crashes or process is killed, still information should remain retained. Most of the time, the information is sharable among many processes.
- If information kept in the address space of one process, other processes will not be able to access it.
- It is necessary to keep information independent of the processes.
- Following are the necessities for long term information storage.
  - It must be possible to store a very large amount of information.
  - The information should not loss after termination of the process using it.
  - Several processes must be able to access the information simultaneously.
  - In order to fulfill the above requirements, it is necessary to store information on disks and other secondary storage in units called **files.**
  - A file is a named collection of related information that is recorded on secondary storage. The data cannot be written directly to secondary storage unless they are within a file. Processes can then read information from the file.
  - It also can write new information into the file if needed. After process termination, information in file should remain retained and should not vanish. A file should only vanish when its holder clearly removes it. Operating system manages the files.
  - File system describes how files are structured, named, accessed, used, protected and implemented.

☞ **Files**

Following discussion about files from user's point of view, that is, how files are used, named, and what are the different properties they have.

5.1.2 File Naming

- Process gives the name to the file when it creates it.
- After completion of the execution, even though process terminates, file still exist and other processes can access that file by the same name.
- The rules for file naming are not same for all systems. It varies from system to system, but all current operating

Q. Compare FIFO and LRU page replacement algorithm. *(Refer section 4.16.1)*

☞ **Syllabus Topic : Optimal**

Q. Compare Optimal, LRU and FIFO page replacement algorithms with illustration. *(Refer section 4.16.2)*

☞ **Syllabus Topic : LRU**

Q. Compare Optimal, LRU and FIFO page replacement algorithms with illustration. *(Refer section 4.16.3)*

Q. Compare FIFO and LRU page replacement algorithm. *(Refer section 4.16.3)*

☞ **Syllabus Topic : LFU**

Q. Explain counting based approach. *(Refer section 4.16.5(A))*

Example 4.17.4 (10 Marks) (Nov. 2014)

Example 4.17.6 (10 Marks) (June 2015)

Example 4.17.7 (10 Marks) (Dec. 2015)

Example 4.17.8 (10 Marks) (Dec. 2016)

Example 4.17.9 (10 Marks) (Dec. 2018)

☞ **Syllabus Topic : Allocation of Frames**

Q. Explain different strategies for allocation of frames *(Refer section 4.18)*

☞ **Syllabus Topic : Thrashing**

Q. What is thrashing ? *(Refer section 4.19)* (5 Marks) (June 2019)

CHAPTER 5

Module 5

File Management

Syllabus Topics

Overview, File Organization and Access, File Directories, File Sharing, Secondary Storage Management, Linux Virtual File System.

Syllabus Topic : File Management - Overview

5.1 Overview

5.1.1 Files and File Systems

- Every application needs to access and store the information. During execution, process can keeps the information in its address space.
- This address space to store information is sufficient for some type of applications but not for others, which requires huge size of information.
- After finishing the execution, process terminates and information also vanishes.
- For many applications, it is required that information should be retained forever.
- Even though computer crashes or process is killed, still information should remain retained. Most of the time, the information is sharable among many processes.
- If information kept in the address space of one process, other processes will not be able to access it.
- It is necessary to keep information independent of the processes.
- Following are the necessities for long term information storage.
  - It must be possible to store a very large amount of information.
  - The information should not loss after termination of the process using it.
  - Several processes must be able to access the information simultaneously.
  - In order to fulfill the above requirements, it is necessary to store information on disks and other secondary storage in units called **files.**
  - A **file** is a named collection of related information that is recorded on secondary storage. The data cannot be written directly to secondary storage unless they are within a file. Processes can then read information from the file.
  - It also can write new information into the file if needed. After process termination, information in file should remain retained and should not vanish. A file should only vanish when its holder clearly removes it. Operating system manages the files.
  - **File system** describes how files are structured, named, accessed, used, protected and implemented.

☞ **Files**

Following discussion about files from user's point of view, that is, how files are used, named, and what are the different properties they have.

5.1.2 File Naming

- Process gives the name to the file when it creates it.
- After completion of the execution, even though process terminates, file still exist and other processes can access that file by the same name.
- The rules for file naming are not same for all systems. It varies from system to system, but all current operating

systems allow strings of one to eight letters as legal file names.

- If an operating system distinguishes the type of a file, it can then operate on the file in logical ways.
- Digits and special characters are also allowed to use while giving the file names. UNIX operating system differentiates between upper and lower case letters, whereas MS-DOS does not.
- Windows 95, Windows 98, Windows NT and Windows 2000 support the MS-DOS file system and thus inherit its properties.
- File name is divided into two parts - name and extension which are separated by period character. User and operating system can recognize the type of file from name only.
- MS-DOS allows the file names containing 1 to 8 characters along with an optional extension of 1 to 3 characters.
- On the contrary, in UNIX user has a choice of deciding the size of extension. In UNIX, file can have two or more extensions.
- Table 5.1.1 shows the some examples of file extension, its meaning, and type.

Table 5.1.1

Extension	Meaning	Type
gif	CompuServe Graphical Interchange Format image	Image
html	World Wide Web Hypertext Markup Language document	Web page
obj, o	Complied machine language not linked	Object
exe, bin, com	Ready to execute machine language program	Executable
txt,doc	Text, document, data	Text
lib, a, dll	Libraries of routines for programmers	Library
arc,zip,tar	Grouped files together sometime	Archive
	compressed for archiving or storage.	
mp3,avi, mpg	Binary files containing audio or audio/video information	Multimedia
pdf, ps	Its content cannot be changed	Portable document format and postscript

5.1.3 File Structure

Q. Explain the different techniques to structure the files.

Following are the three ways in which files can be structured.

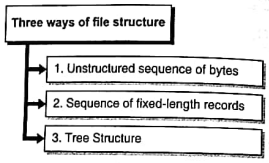


Fig. C5.1 : Three ways of file structure

➜ **1. Unstructured sequence of bytes**

- In this structuring, operating system treats content of the file as sequence of bytes. Any meaning must be imposed by user-level programs.
- Both UNIX and Windows use this approach. This structuring provides more flexibility.
- Any types of content can be kept in the file and as per convenience of the user, name can be given to the file.

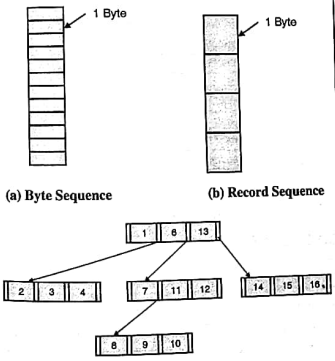
➜ **2. Sequence of fixed-length records**

- In this approach file is treated as sequence of fixed length records.
- Each record has its internal structure. Any read operation on file returns one record and write operation will append or overwrite one record.
- When punched cards were in use, record size was of 80 characters and of 132 characters proposed for line printers.

- Many operating systems based their file systems on files consisting of 80-character records when punched cards were in use.
- Programs read input from file in the unit of 80 characters and wrote in units of 132 characters keeping remaining 52 characters blank.

➜ **3. Tree structure**

- In this organization, a file consists of the records which are organized in tree structure.
- All the records not necessarily have same length. Each record has a key field in a fixed position.
- The records in the tree are arranged in the sorted order of key field, to permit quick searching for a particular key.
- The main aim in this type of organization is to search record on particular key.
- The next record also can be searched. Operating system decides where to add new record.
- User is not concern about this operation. This type of file is broadly used on the large mainframe computers and still used in some commercial data processing.
- Following Figs. 5.1.1 (a), (b) and (c) shows above three kinds of files. Fig. 5.1.1 (c) shows the tree of records of student file arranged in sorted order of roll numbers.



(a) Byte Sequence (b) Record Sequence

(c) Tree

Fig. 5.1.1

Disadvantage of supporting multiple file structures

- If multiple file structures are supported by the operating system then it would be huge in size.
- For supporting the different file structures, the corresponding code should also present in operating system.
- Additionally, it is required to define every file as operating system supports for one of the file type.
- If operating system does not support the structure of information needed by application then it can lead to severe problems.
- Operating systems like UNIX, MS-DOS enforce and support file structures which are least in numbers.
- UNIX treats each file as a sequence of 8-bit bytes and operating system does not carry out the understanding of these bits.
- As a result of this approach, maximum flexibility but little support is offered.
- In order to interpret an input file to the suitable structure, application should have included its own code.
- On the other hand, all operating systems must support minimum one structure of an executable file.
- Due to this support, the system will be capable of loading and running the programs.

5.1.4 File Types

Q. Write short note File Types.

Several operating systems support many types of files.

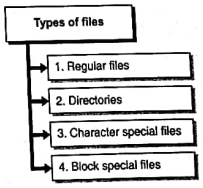


Fig. C5.2 : File types

➜ **1. Regular files**

Regular files contain user information. The byte sequence, record sequence and tree structured explained above are the examples of regular files.

➜ **2. Directories**

These are system files for maintaining the structure of the file system.

➜ **3. Character special files**

Character special files are related to input/output and used to model serial I/O devices such as terminals, printers, and networks.

➜ **4. Block special files**

These are used to model disks.

- All regular files are normally either ASCII files or binary files.
- ASCII files contain lines of text. Each line is terminated by either carriage return character or line feed character. MS-DOS uses both types of files. Lines can be of different length.
- The great benefit of ASCII files is that they can be displayed and printed as they are.
- Any text editor can edit ASCII file. Furthermore, if large numbers of programs use ASCII files for input and output, it is easy to connect the output of one program to the input of another, as in shell pipelines. Binary files are not ASCII files.
- Listing them on the printer gives listing which is beyond the understanding. It gives the random garbage if it listed on the printer. Usually, they have some internal structure known to programs that use them.
- UNIX binary file is an archive. It consists of a collection of library procedures compiled but not linked. The usefulness of file types can be understood from the TOPS-20 operating system.
- The recompilation of source file will be carried out automatically after its modification.
- This recompilation is done if user wants to execute the object file. It guarantees that the user all the time runs an up-to-date object file. It saves the waste of time in executing the old object file.
- To achieve this automatic recompilation of modified source file, the operating system must have the capability to make a distinction between source file and object file.
- Also OS should be able to check file creation time and time at which file was lastly modified. In order to choose the right compiler, OS must also have a ability to determine the language of the source program.
- Fig. 5.1.2 demonstrates the UNIX version of executable binary file.
- Although this file is just a sequence of bytes, without a proper format operating system cannot execute this file. The file is divided into five segments. These are header, text, data, relocation bits, and symbol table.
- The very first field in header is magic number. It recognizes the type of file as an executable file.
- It facilitates to prevent the unintentional execution of a file which is not in this format.
- After magic number, the sizes of the different sections of the file are shown. After these various section sizes, the address at which execution starts, and some flag bits are present.
- Following the header are the text and data of the program itself. These are loaded into memory and relocated using the relocation bits. The symbol table is used for debugging.

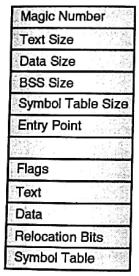


Fig 5.1.2 : An executable file

5.1.5 File Attributes

Q. Explain various file attributes in brief.

- File is identified by its name which is string of the characters. Every file has a name and its data. All operating systems associate other information with each file, for example, the date and time of file creation and the size of the file.
- Such extra information associated by operating system is called as attributes. The lists of attributes are not same for all the systems and vary from one operating system to another.
- No single existing system supports all of these attributes, but each one is present in some system.

☞ **Attributes and its meaning**

Attributes	Meaning
Protection	Access-control information determines who can do reading, writing, executing, and so on.
Size	The current size of the file
Type	Information is needed for systems that support different types of files.
Creator	ID of the person who created the file
Owner	Current owner of the file
Password	Password needed to access the file
Identifier	This unique number which identifies the file within the file system.
Hidden flag	0 for normal; 1 for do not display in listings
Read-only flag	0 for read/write; 1 for read only
ASCII/binary flag	0 for ASCII file; 1 for binary file
Lock flags	0 for unlocked; nonzero for locked
Key length	Number of bytes in the key field
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Maximum size	Number of bytes the file may grow to
System flag	0 for normal files; 1 for system file

5.1.6 File Operations

Q. Explain various file operations in brief.

- Files are used to store information which can be used later on. For storage and retrieval purpose, different types of systems offer different operations.
- For the operations like create, write, read, reposition, operating system offer the system calls. The majority of common system calls relating to files are listed in Fig. C5.3.

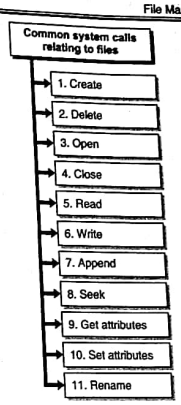


Fig. C5.3 : Common system calls relating to files

➜ **1. Create**

- The new file is created without data. It is needed to locate the space for this created file in file system.
- The intention of the call is to declare that the file is created and some of the attributes needs to be set.

➜ **2. Delete**

File need to be deleted to release the disk space occupied by file when it is no longer required.

➜ **3. Open**

- Before making use of file, it is required to open the file first by any process. To have quick accesses in future, all the attributes and disk addresses are bought into main memory.

➜ **4. Close**

- After use of the file completes, all accesses gets finished. The fetched attributes and disk addresses in memory are no longer required, so the file should be closed to release the internal table space.
- The disk space is divided in blocks and is written in blocks. Closing a file compel writing of the file's last block.

➜ **5. Read**

- The system call that is used for reading a file specifies the name of the file.

- Generally, the bytes are fetched from the current position. The system should maintain read pointer at the address in the file from where next read is to begin.
- The caller must state the amount of data desired and must also offer a buffer to put them in.

➜ **6. Write**

- System call to write the data in to file specify file name and information to be written to the file.
- Writing is again at the current position. The existing file size will increase if writing starts from end of the file. In this case, the current position is at the end of file.
- On the contrary, if writing is done at somewhere in the middle of the file, the previously existed data will be overwritten and vanished permanently.

➜ **7. Append**

- With the help of append system call; data gets appended only to the end of the file. Systems offering less number of system calls do not generally provide append.

➜ **8. Seek**

- The seek system call is used to place the pointer at a particular location in the file.
- It is needed for random access files to specify from where to obtain the data. After this call has completed, data can be read from, or written to, that position.

➜ **9. Get attributes**

Processes frequently need to read file attributes to complete their designated work. Get attributes system call is used for this purpose.

➜ **10. Set attributes**

User can change the some attributes of file after its creation. This system call makes that possible.

➜ **11. Rename**

User needs to change the name of an existing file. This system call makes that possible

Syllabus Topic : File Organization and Access

5.2 File Organization and Access

5.2.1 File Access

➜ **(June 15)**

Q. Explain different file access methods.

MU - June 2015. 10 Marks

Q. Write short note File Access (sequential access, direct random access).

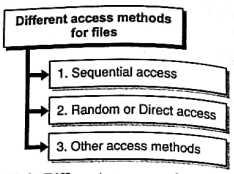


Fig. C5.4 : Different access methods for files

➜ **1. Sequential access**

- The simplest access method is sequential access. Early operating systems provided only this kind of file access.
- In this type of file access, process reads all the records in a file in order one record after other, starting at the beginning.
- While accessing, skipping of any record or reading them out of order is not possible.
- This access method was convenient for storage medium such as magnetic tape to a certain extent than disks.
- The reading of the next portion of the file is carried out by read operation and automatically file pointer is moved forward.
- The appending of new information to the end of the file is carried out by write operation and file pointer advances to the end of the newly written portion.

➜ **2. Random or Direct access**

- When use of disk started for storing files, it became possible to read the bytes or records of a file out of order. It is because; disks allow random access to any file block.
- It also became possible to access records by its key instead of by position. Files whose bytes or records can be read in any order are called **random access files.** They are required by many applications such as database systems.
- If railway customer calls up and wants to reserve a seat on a particular train, the reservation program must be able to access the record for that train directly instead of reading hundreds of records of other trains first.
- For the random access method, the file operations must be modified to include the block number as a parameter.

- Thus, we have read n and write n, where n is the relative block number. Actual absolute disk address of the block is different.
- The beginning of the relative block number is from address 0.
- Then next block number is 1 and so on although the absolute disk address of the first block is 14045 and second block is 3191.
- The relative block numbers permits the operating system to take the decision about location of placing the file and facilitate the user to stop from accessing file system portions which are other than his file portion.
- Every read operation gives the position in the file to start reading at. In the second one, a special operation, seek, is provided to set the current position.
- After a seek, the file can be read sequentially from the now-current position.

➜ **3. Other access methods**

- These access methods can be built on top of a random-access method.
- These methods generally involve the construction of an index for the file.
- The index has pointers to the various blocks.
- The record in the file is searched by searching the index first and then uses the pointer to access the file directly and to find the desired record.
- Several factors are important in choice of file organization. These are
  - Minimum access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

5.2.2 File Organizations

Q. Explain different types of file organization.

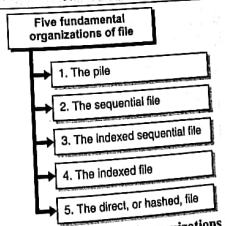


Fig. C5.5 : Five fundamental organizations of file

➜ **1. The Pile**

Q. Explain pile file organization.

- It is simplest form of file organization. In this file data is stored in the order of arrival. Each record in this file comprises one burst of data.
- The idea behind the pile is simply to collect the mass of data and save it.
- Records contain different or same type of fields in diverse orders. Each field includes a field name in addition to a value.
- The delimiters implicitly shows field length, or it is explicitly included as a subfield, or recognized as default for that field type.
- A pile file does not have any structure, so exhaustive search is used to access any record.
- To search record with particular field and value, the search inspects each record in the pile until the desired record is located or the complete file has been searched.
- The complete file must be searched to locate all records of particular field and value.

➜ **2. The Sequential File**

Q. Explain sequential file organization.

- It is most commonly used structure and records have a fixed format.
- Each and every record in the file has same length, with equal number of fixed-length fields in a particular order.
- As the length and location of each field are known, simply the values of fields are required to be stored. The field name and each field length are attributes of the file structure.
- The first field in each record is considered as key field and it is used to identify the record.
- As a result, the values of key in each record are always different. The records are stored in the logical order of key.
- For batch applications these files are more suitable. This file can be organized with no trouble on tape as well as disk.
- The sequential file physically can be organized as a linked list. Each physical block contains one or more records.
- Each block on disk holds a pointer to the next block.

- If new record is inserted, a pointer needs to be manipulated only and new records would not occupy a particular physical block position.

➜ 3. The indexed sequential file

- The indexed sequential file eliminate drawback of sequential file. Similar to sequential file, records are organized in logical order of a key field.
- The indexed sequential file contains index to the file to support random access, and an overflow file. The index offers a searching ability to reach speedily the required record.
- The overflow file is used in order that a record in the overflow file is searched by following a pointer from its ancestor record.
- In a single level of indexing, the index is a simple sequential file. In the index file, each record contains two fields. A key field identical to the key field in the main file, and a pointer into the main file.
- To search a particular field, the index is searched to locate the largest key value that is equal to or precedes the key value to be searched. The search carries on in the main file at the location denoted by the pointer.
- The indexed sequential file really decreases the time needed to access a single record, without giving up the sequential nature of the file.
- For sequentially processing of complete file, the records of the main file are visited sequentially until a pointer to the overflow file is located, then accessing goes on in the overflow file until a null pointer is came across, at which time accessing of the main file is resumed where it left off.

➜ 4. The Indexed File

Q. Explain indexed file organization.

- The sequential file and the indexed sequential file can search the record based on a single field of the file. Apart from using key field, they cannot search record using other attribute of record.
- To overcome this difficulty and accomplish the flexibility, a structure containing multiple indexes, one for each type of field is required.
- In indexed file, records are accessed only through their indexes.
- As a result, there is now no limit on the placing of records providing a pointer in as a minimum one index refers to that record. In addition, variable-length records can be used.
- In this structure, two kinds of indexes are used. An exhaustive index holds one entry for each record in the main file.
- In order to search easily, the index itself is organized as a sequential file.
- A partial index holds entries to records where the field of concern exists.
- As records are of variable-length, some records will not have all fields. After addition of a new record to the main file, all of the index files must be updated.

➜ 5. The Direct or Hashed File

Q. Explain hashed file organization.

- This type of file makes use of the ability found on disks to access straight any block of a known address. There is no idea of sequential ordering like sequential or indexed sequential file.
- A hashing operation is performed on the key value using hash function.
- Direct files are frequently used where very fast access is necessary, where there is use of fixed length records, and where records are always accessed one by one.

Syllabus Topic : File Directories

5.3 File Directories

5.3.1 Directory Structures

Q. Explain different directory structures.

- Directories keep track of files. Directories are itself files in many system. Systems store huge number of files on large capacity disk. In order to manage all these data, we need to organize them.
- This organization involves the use of directories.
- The most common methods for defining the logical structure of a directory shown in Fig. C5.6.

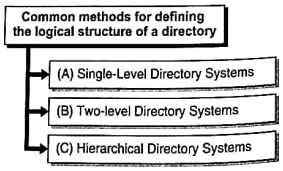


Fig. C5.6 : Common methods for defining the logical structure of a directory

➜ 5.3.1(A) Single-Level Directory Systems

- Single-level directory is the simplest directory structure. In this directory structure, one directory contains all the files.
- This single directory is also called as a root directory. Since on early personal computers, only one user was working, this system was more general.
- Fig. 5.3.1 shows single-level directory system containing five files, owned by three different users P, Q, and R. User P has two files, User Q has two files and R has one file in the directory.

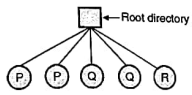


Fig. 5.3.1: Single-level directory system

☞ Advantages

- It is simple to implement.
- Locating files become faster as there is only one place to look.

☞ Limitations

- If single user has huge number of files kept in single directory, it becomes difficult to remember the name of each file.
- If more than one user keeps the files in a single directory, then different users may give the same names to their files, violating the rule of uniqueness of names.

➜ 5.3.1(B) Two-level Directory Systems

- Two-level Directory System overcomes the limitations of Single-level directory. In this directory system, a private directory is given to each user. When a user refers to a particular file, only his own directory is searched.
- As different users directories are different, the same name given to the files do not interfere each other. There is no problem in giving the same name to the files in different directories.
- Single user directory have a compulsion of having all files unique name. While creating the file for particular user, the operating system makes confirmation about whether another file of that name exists in the same user's directory or not. To know this existence of file or not, the OS searches the directory of the user for which file is to be created. To delete a file, the operating system has to search only local user directory.
- Hence operating system cannot fortuitously delete another user's file that has the same name. Fig. 5.3.2 shows two level directory systems with separate users directories and its files.

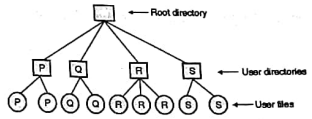


Fig. 5.3.2 : Two-level Directory Systems

- Above design multiuser computer or on a simple network of personal computers that shared a common file server over a local area network. In this system some sort of login procedure is needed.
- When a user attempts to open a file, the system knows which user it is in order to know which directory to search.
- Files are system programs such as loaders, assemblers, compilers, utility routines, libraries. Loader reads these files and executes when the right commands are given to the operating system.
- These commands are treated by command interpreters as the name of a file in order to load and execute. In case of two level directory systems, this file name would be searched in the current user directory.
- If system files are kept in each user directory then problem can be resolved.
- This solution leads to wastage of memory as each user directory contains the copied system files. If special user directory containing all system files is defined the above problem can be solved.

☞ Advantages

- Solve name collision problem as every user has separate directory.
- Independent user gets isolated from each other.

☞ Limitations

- If the users are co-operative, that is, working on common task, then some system does not allow accessing the other user's files.

- In some system, if permitted, one user must have the ability to name a file in another user's directory. To name a particular file uniquely in a two-level directory, we must give both the user name and the file name.
- It is not satisfactory for users with a large number of files. Even on a single-user personal computer, it is not convenient.

➜ 5.3.1(C) Hierarchical Directory Systems

- It is quite general and advantageous for users to group their files together in logical ways. A student for example, might have a collection of files related to different subjects of her curriculum.
- First collection can be the group of files related to one subject, a second collection of files related to second subject and so on. Here, some way is required to group these files together in flexible way.

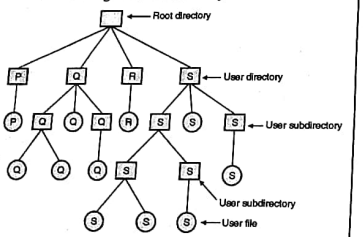


Fig. 5.3.3 : Hierarchical Directory Systems

- A tree of directories is the solution to store such group of files. A tree is the most common directory structure.
- In this approach, each user can have as many directories as are needed so that files can be grouped together in expected ways.
- The tree has a root directory, and every file in the system has a unique path name. The approach is shown in Fig. 5.3.3. Root directory contains directory P, Q, R, and S which belongs to different users. Users Q and S have created the subdirectories.
- As user can create random number of subdirectories, it offers a commanding structuring tool for users to organize their work. This is the reason, nearly all modern file systems are organized in this approach.

Advantage

- With a hierarchical directory system, in addition to their files, users can access the files of other users by specifying its pathname.

5.3.2 Path Names

- When file system is organized as tree of directories, a file is accessed by specifying the path name. Path names can be of two types. These are :

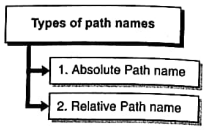


Fig. C5.7 : Types of path names

➜ 1. Absolute path name

It consists of the path from the root directory to the file. The meaning of the path */usr/myfolder/myfile* is that, the root directory contains a subdirectory *user*, which in turn contains a subdirectory *myfolder*, which contains the file *myfile*.

➜ 2. Relative path name

It consists of the path from the current working directory to the file. A user can assign one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory. If the current working directory is */usr/myfolder*, then the file whose absolute path */usr/myfolder/myfile* can be referenced simply as *myfile*.

- In UNIX operating system the elements of the path are separated by /. In Windows the separator is \. The same path */usr/myfolder/myfile* in UNIX, is specified in Windows as *\usr\myfolder\myfile*.
- If the first character of path is separator then path is absolute path. If during working, program always needs a particular file, it should use absolute path to access that file from any current working directory.

5.3.3 Directory Operations

Q. Explain different directory operations.

To manage the directories, different system calls demonstrate more dissimilarity from system to system with respect to system calls for files. In order to give an idea of working of system calls for directories, consider the following examples from UNIX.

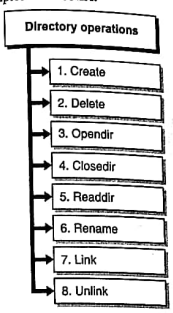


Fig. C5.8 : Directory operations

➜ 1. Create

Empty directory is created excluding dot and dotdot (. and ..), which are placed there automatically, by the operating system.

➜ 2. Delete

If only single dot and double dot (. and ..) is present in directory then it is treated as empty. single dot and double dot within directory cannot usually be deleted.

➜ 3. Opendir

Directory can be read to list all the files from it. Directory should be opened before reading just opening and reading the file.

➜ 4. Closedir

After reading completes, a directory should be closed to free up inner table space.

➜ 5. Readdir

It returns the next entry in an open directory. **readdir** always returns one entry in a standard format irrespective of possible directory structures is being used.

➜ 6. Rename

Directories can be renamed just like files. Rename renames the directory.

➜ 7. Link

- Due to linking, a file appears in more than one directory.
- Link system call specify an existing file and its path name, and creates a link from the existing file to the name specified by the path. In this fashion, the same file may come into view in several directories.

➜ 8. Unlink

Unlink removes the directory entry. If the file being unlinked only exist in one directory then it is deleted from file system.

5.3.4 Directory Implementation

Q. Explain the implementation of directory in detail.

- Every file must be opened before it read. User provides path name to access the file.
- When a file is opened, the operating system uses the same path name to find the directory entry.
- As we have discussed in file allocation methods, the directory entry contains the information needed to find the disk blocks of the file.
- In contiguous allocation, this information may be the disk address of the entire file.
- In linked list allocation, directory contains the starting and ending block number of a file and in indexed allocation, directory contains starting block number of a file. In all schemes, the main function of the directory system is to map the ASCII name of the file onto the information needed to find the data.
- Every file also has its attributes. Different file attributes we have already discussed.
- Directory entry stores these attributes of a file.
- In the simple design, a directory consists of a list of fixed-size entries, one per file, containing a file name of fixed length, a structure of the file attributes, and one or more disk addresses(up to some maximum) denoting where the disk blocks are. Fig. 5.3.4 shows this design.

File	Attributes
A	attributes
B	attributes
C	attributes
D	attributes

Fig. 5.3.4

- Fig. 5.3.4 shows a simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry

- Some systems use i-nodes. In this case, directory entries are just name of the file and i-node number. All the attributes of a file is stored in i-node.
- Fig. 5.3.5 shows this design.

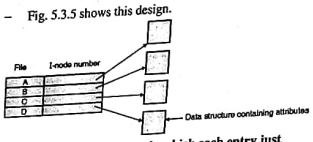


Fig. 5.3.5 : A directory in which each entry just refers to an i-node

- Above two approaches are corresponds to MS-DOS which have short file names of 1-8 characters and optional extension of 1-3 characters.
- In UNIX Version 7, file names were 1-14 characters, including any extensions.
- All modern operating system support longer and variable-length file names. Following are the two approaches to implement it.
- The simplest solution is to keep file length of maximum 255 characters.
- The above two designs then can be used in which maximum 255 characters is reserved for file name.
- Since it is not common to have all files such longer names, directory space would be wasted.
- In this approach, fixed size of all directory entries is not considered.
- Each directory entry contains a fixed section, which begins with length of the entry, and then followed by data with a fixed format which are generally file attributes. It is shown in Fig. 5.3.6.

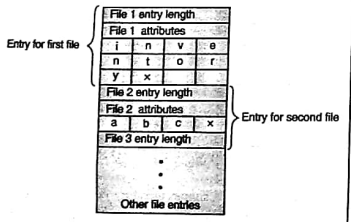


Fig. 5.3.6 : In-line handling of long names

- In this example two file inventory and abc is shown. Termination of each file is shown in Fig.5.3.6 by cross symbol.
- A drawback of this method is that, after removing file, a variable-sized free space is initiated into the directory. It may happen that, next file to come, may not be accommodated in this free space.
- Solution on this issue is compaction of directory. Another problem is that a if single directory entry spreads in several pages, page fault occurs during reading of a file.

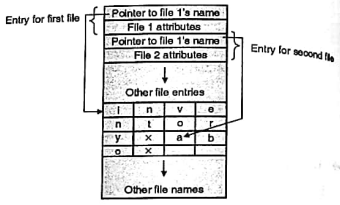


Fig. 5.3.7 : Long name handling using heap

- The second approach to deal with variable-length names is to allow directory entries of fixed length and maintain the file names jointly in a heap at the end of the directory as shown in Fig. 5.3.7.
- The drawback discussed above is defeated here. When an entry is removed, the next file entered will always fit in freed space. The management of heap will be the burden here and page faults cannot be avoided as in previous method.
- All the designs discussed, supports searching of directory sequentially from beginning to end for finding up the file name. Sequential search would be inefficient if directories are longer.
- Solution to this problem is use of hash table in each directory as hash search is efficient compare to sequential search.

Syllabus Topic : File Sharing

5.4 File Sharing

Q. Explain issues related to file sharing.

It is necessary to share a file among multiple of users in multiuser system. It needs to deal with two issues: that are access rights and the management of simultaneous access.

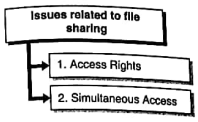


Fig. C5.9 : Issues related to file sharing

5.4.1 Access Rights

Q. Explain different access rights to the file.

- The file system should offer a flexible facility for permitting widespread file sharing among multiple users.
- It is also a responsibility of file system to offer a many alternatives in order to control the way of file accessing.
- In general, it important to grant particular access permissions or rights to users or groups of users to access a particular file. A broad category of access rights has been used.
- The following are the some of the examples of access rights that can be granted to a particular user for a particular file.

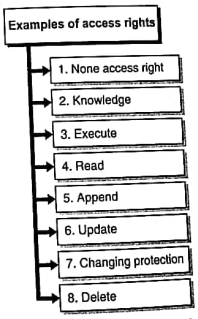


Fig. C5.10 : Examples of access rights

➜ **1. None access right**

The user may not still know about presence of the file. To put into effect this access limit, the user is restricted from reading the user directory that contains this file.

➜ **2. Knowledge**

With knowledge access right, the user can find out existence of file and owner of it. The user can appeal the owner for added access rights.

➜ **3. Execute**

- A loading and execution of program is possible to user but copying it is not allowed.

➜ **4. Read**

- A read access right allows the use to read the file for any work, together with copying and execution.

➜ **5. Append**

- The user can only add data to the file at the end.

➜ **6. Update**

- The update access right allows the user to modify, delete, and add the data to the file's data.

➜ **7. Changing protection**

- The user can alter the access rights assigned to other users. Normally, this right is associated only with the owner of the file.
- The owner is allowed to extend this right to other users in some systems

➜ **8. Delete**

- The user is allowed to delete the file from the file system as per need.

These rights assigned to user form a hierarchy, with each right involving those that precede it.

- Thus, if any user is assigned the read right for a particular file, then that user also gets following rights: knowledge, execute.
- The person responsible to create file is by default the owner of that file. The owner is granted all of the access rights and may grant rights to others.
- The different categories of users to which access can be given are :
  - **Specific user :** Individual users having its own user ID
  - **User groups :** A group of users who belonging to particular user group. The system keeps track of the membership of user groups.
  - **All :** Authenticated users of this system.

5.4.2 Simultaneous Access

- It is necessary to protect the shared file from simultaneous updating from more than one user.

- When more than one user granted access to append or update a file, the operating system or file management system must implement some way to restrict it.
- A brute-force scheme permits a user to lock the complete file when it is to be updated. It is also possible to lock individual records during update.

Syllabus Topic : Secondary Storage Management

5.5 Secondary Storage Management

- A file spans many blocks on secondary storage. These blocks are allocated to file by operating system or file management system.
- So first it is important to consider the allocation of space on secondary storage to files, and second, it is essential to keep track of the space available for allocation.

5.5.1 File Allocation

Several concerns are present in file allocation :

- After creation of new file, whether maximum space needed for the file should allocate at one time or not.
- A portion is a contiguous set of allocated blocks whose size can range from a single block to the whole file. The size of portion is another issue to be considered for allocation.
- The type of data structure or table used to keep track of the portions allocated to a file.

5.5.2 Pre-allocation versus Dynamic Allocation

- In pre-allocation strategy, the maximum size of a file is confirmed at the time of the file creation request.
- In many cases, a reliable estimation of size is possible; for example when file need to be transfer to other machine across the communication network.
- On the other hand, for many applications, it is not easy to guess reliably the maximum size of the file so as not to run out of space.
- Thus preallocation strategy has limitation for secondary storage allocation.
- Thus, it is beneficial to use dynamic allocation, which allocates space to a file in portions as required.

5.5.3 Portion Size

- The other issue that needs to be considered is, the size of the portion allocated to a file.
- In one case, a size of the portion can be as much as necessary to hold the entire file.
- In other case, space on the secondary storage is allocated one block at a time. While selecting a portion size, there is a trade-off between efficiency considering a single file versus overall system efficiency. Following four items needs to be considered in the trade-off:
- If blocks are allocated continuously, performance increases. It is true in particular when next block is to be accessed, and very much for transactions running in a transaction-oriented operating system.
- If great numbers of small size portions are present then it increases the size of tables required to manage the allocation information.
- If portions are of fixed size (e.g., blocks) then it makes easier the reallocation of space.
- If variable-size or small fixed-size portions are present then it reduces waste of unused storage because of over allocation.
- Certainly, these items act together and must be considered collectively. It gives two major alternatives:
  - **Variable, large contiguous portions:** As variable size keeps away from wastage of space and due to large portions, the file allocation tables are small. As a result performance is better but space is hard to use again.
  - **Blocks:** If portions are of fixed and small size, it offers better flexibility. They may need large tables or complex structures for their allocation. Contiguity has been neglected as a major objective; blocks are allocated as required.
- Any option is well-suited with preallocation or dynamic allocation. If we consider the case of variable, large contiguous portions, a file is preallocated one contiguous group of blocks.
- This gets rid of the need for a file allocation table; all that is essential is a pointer to the first block and the number of blocks allocated.
- If we consider the blocks, all of the portions required are allocated at once.
- So the file allocation table will have fixed size, since the number of blocks allocated is fixed. If portions are of variable-, it is necessary to deal with the fragmentation of free space.

- The following are policies used to avoid this fragmentation
  - **First fit:** Allocate the first available unused contiguous group of blocks of enough size from a free block list.
  - **Best fit:** Allocate the smallest unused group that is of enough size.
  - **Nearest fit:** Allocate the unused group of enough size that is closest to the previous allocation for the file to increase locality.
- It is very difficult to decide which policy is best.
- The complexity in modeling other substitute strategies is the interaction of many factors such as types of files, pattern of file.

5.5.4 File Allocation Methods

→ (Dec. 14, May 16)

Q. What are the different allocation methods with reference to File Systems?
MU - Dec. 2014, May 2016, 10 Marks

- Disks supports for direct access. Due to which we can have flexibility in the implementation of files.
- There should be optimum and effective utilization of disk space. The allocation of disk space to files should always allow for the optimum and effective utilization of disk space so that quick access to the files will be supported.
- Following methods are used majorly in different operating systems.

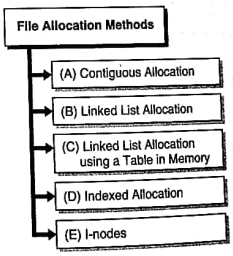


Fig. C5.11 : File allocation methods

→ 5.5.4(A)Contiguous Allocation

Q. Explain contiguous file allocation method with its advantages and disadvantages.

- In contiguous allocation each file takes up a set of contiguous blocks on the disk.
- Disk addresses define a linear ordering on the disk. If each block size on disk is 2 KB, then 30 KB file would be allocated 15 consecutive blocks.
- Contiguous allocation of a file is defined by the disk address of the first block and length (total blocks occupied). If the file is m blocks long and starts at location i, then it takes up blocks $i, i+1, i+2, ..., i+n-1$.
- The directory entry for each file specifies the address of the starting block and the total number of blocks allocated for this file.
- Directory entry in Fig. 5.5.1 shows that file A starts at block 0 and it is 3 block long occupying block 0, block 1 and block 2. Similarly, file B starts at block 6 and it is 5 block long and so on.

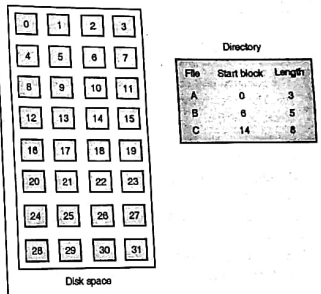


Fig. 5.5.1: Contiguous Allocation of disk space

Advantages

- It is simple to implement because, only information needed to keep track on files block is starting block and length (total number of blocks occupied).
- For sequential access, the file system memorizes the disk address of the last block referenced and, when required, reads the next block.
- To access block k of a file directly which starts at block i, we can immediately access block i + k.
- Thus, both sequential and direct access can be supported by contiguous allocation.
- Performance is high as whole file is read from disk in single disk operation. To reach to the first block only one seek is needed.

Disadvantages

- When allocated file is deleted, continuously allocated blocks to the file become free.
- For new file to allocate, these blocks might not be sufficient. If new file size is small than previously allocated size, some holes remains which are not enough to allocate any new file.
- As a result, the disk ultimately consists of files and holes causing external fragmentation.
- In some cases, it is simple it is simple to find out space is needed for a file to be allocated.
- For output file this size estimation becomes difficult. For CD-ROM, contiguous allocation is reasonable as all file sizes are known in advance.

→ 5.5.4(B) Linked List Allocation

Q. Explain linked list file allocation method with its advantages and disadvantages.

- Linked list allocation overcomes the limitations of contiguous allocation. In linked allocation, each file is a linked list of disk blocks.
- The scattered disk on the disk can be allocated to the file. The directory contains a pointer to the first and last blocks of the file.
- Creation of a new file is simple. For this, it is required to create a new entry in the directory.
- The directory entry holds a pointer to the first disk block of the file.
- The pointer value is nil for empty file and file size for this file is zero.
- The pointers need to be followed from block to block in order to read a file.

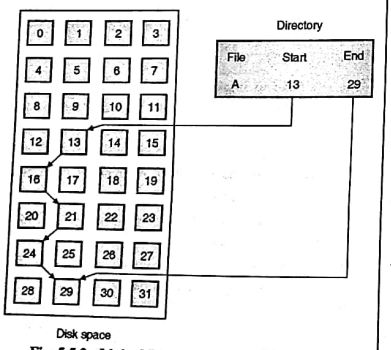


Fig. 5.5.2 : Linked List Allocation of disk space

- Fig. 5.5.2 shows linked list allocation for file A. The file A of five blocks start at block 13 and continue at block 16, then block 21, then block 24, and finally block 29.
- Each block holds a pointer to the next block. These pointers are unavailable to the user.
- User can see the block size excluding the size required to store the address.
- Accordingly, if each block size is 512 bytes, 4 bytes are required to store address, then the user sees blocks of 508 bytes.

Advantages

- In linked list allocation, every disk block can be used which is not possible in contiguous allocation.
- Hence, there is no external fragmentation except for internal fragmentation in the last block.
- Reading a file sequentially is simple.
- It is not necessary to compact disk space.

Disadvantages

- It is inefficient to support a random access capability. To retrieve to block i, the operating system has to start at the beginning and read the i – 1 blocks previous to it, one at a time.
- In each block pointer takes some space, so each file require slightly more disk space rather than its actual size.
- If pointer is lost or damaged file would become inaccessible.
- A wrong pointer can be chosen due to bug in the operating system software or breakdown of disk hardware. As a result of which, the error could turn result in linking into the free-space list or into another file.

→ 5.5.4(C) Linked List Allocation using a Table in Memory

- In linked list allocation, each block needs to store pointer information, therefore entire block is not fully used to store file content.
- This limitation can be eliminated by keeping pointer information in table which always remains in memory.
- If file A occupies block 5, 9, 1, 12 and 7 in the same order and file B occupies block 3, 8, 4, 14 and 6 in the same order.
- The File Allocation Table (FAT) is shown in Fig. 5.5.3.

Physical	Next block	
0		
1	12	
2		
3	8	← File B begins here
4	14	
5	9	← File A begins here
6	– 1	
7	– 1	
8	4	
9	1	
10		
11		
12	7	
13		
14	6	
15		

Fig. 5.5.3 : Linked List Allocation Using a Table in Memory

Advantages

- The entire block is available for data. So all disk space is utilized for allocation of file.
- Random access is much easier.
- The chain need to be followed to get a given offset in the file, but the chain is totally in memory, so it can be followed without making any disk references.
- Directory only stores the starting address (integer block number) of the file and still able to locate all blocks even though file size is very large.

Disadvantage

Whole table must be in memory all the time to make it work

→ 5.5.4(D) Indexed Allocation

Q. Explain indexed file allocation with advantages and disadvantages.

- With File Allocation Table (FAT) in memory, linked list allocation support random access, but the entire table must be in memory all the time.
- In indexed allocation, all the pointers are kept in one location called as index block.
- There is a index block assigned to each file and this index block holds the disk block addresses of that particular file.
- There is a pointer from i^{th} entry in the index block to the i^{th} block of the file. it means n^{th} entry in index block holds the address of the n^{th} disk block.
- The address of the index block of the file is maintained in directory.
- In order to locate and read the i^{th} block, pointer in the i^{th} index-block entry is used.

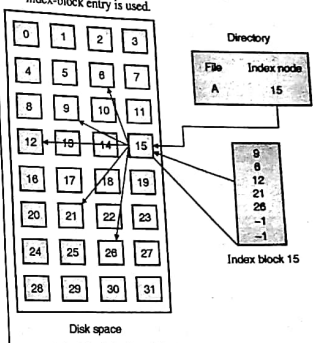


Fig. 5.5.4 : Indexed Allocation of disk space

- Initially when the file is created, all pointers in the index block are initialized to nil.
- When the i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} index-block entry.

Advantages

- Indexed allocation supports random access.
- There is no external fragmentation, because any free block on the disk can be allocated as per request for more space.

Disadvantages

- The pointer overhead of index block is more with compare to the pointer overhead of linked allocation.
- Indexed-allocation schemes suffer from some of the same performance problems as does linked allocation.

→ 5.5.4(E) I-nodes

Q. How I-node is used to allocate the file?

- I-node (Index-node) is the data structure which records the attributes and disk addresses of the files blocks.

I-node is associated with each file and it keeps track of which block belongs to which file.

- If particular file is open, only its i-node needs to be in memory. This is more beneficial with compare to linked list allocation which requires entire file allocation table in memory.

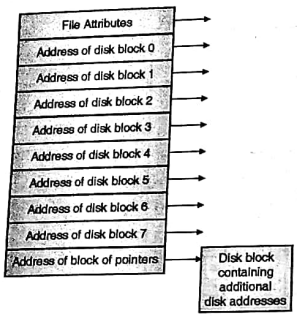


Fig. 5.5.5 : Example of i-node

- The size of file allocation table is proportional to the number of blocks that disk contain. If disk has k blocks file allocation table will have k entries.
- In contrast, the i-node scheme requires an array in memory whose size is proportional to the maximum number of files that may be open at once.
- Fig. 5.5.5 shows example of i-node.
- The last disk address is reserved for the address of a block containing more disk block addresses.
- The reason is, each i-node has space for fixed number of block addresses. If file grows beyond this limit, this last disk address point to disk block containing additional disk addresses.

5.5.5 Free Space Management

- When file is deleted, the disk space becomes free. This space should be reused to allocate to new files. System keeps track of free space using free space list. The disk blocks which are not allocated to any file or directory is free. Free space list records all such free blocks.
- To create a new file, free space list is searched for availability of needed free space. If found, free space gets allocated to the new file and it is took out from the free space list. Freed space due to the deletion of file is again gets included in free space list.
- To implement free space list following two methods are used.

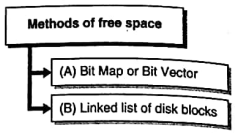


Fig. C5.12 : Methods of free space

→ 5.5.5(A) Bit Map or Bit Vector

Q. Explain bit map method of free space management on secondary storage.

- In this method, every block on disk signified by a 1 bit. Free block is represented by bit 1 and allocated block is represented by bit 0.
- Suppose block numbers 3, 5, 7, 9, 10, 15, 18, 20, 23, 25, and 28 are allocated and rest of the blocks are free.
- The free space map would be as follows. Allocated blocks are shown in bold space.

11**1**0**1**0**1**0**1**0**0**1**1**1**1**0**1**1**0**1**0**1**1**0**1**0**1**1**0**.....

- With a bitmap, it is also possible to keep just one block in memory, going to disk for another one only when it becomes full or empty. Since the bitmap is a fixed-size data structure, if the kernel is (partially) paged, the bitmap can be put in virtual memory and have pages of it paged in as needed.

Advantages

- It is relatively simple and requires less space, since it uses 1 bit per block.
- It is efficient in searching the first free block or k successive free blocks on the disk.

Disadvantages

- The whole vector is needed to keep in main memory. Otherwise it would be inefficient. It is written to disk occasionally for recovery needs.
- Keeping bit map in main memory is promising for smaller disks. It is not necessarily possible for larger disk size.

→ 5.5.5(B) Linked List of Disk Blocks

Q. Explain linked list method of free space management on secondary storage.

- In this method of free space management all the free disk blocks are linked together.
- The pointer to first free block is kept in particular location on disk and cached it in main memory.
- This first block contains a pointer to the next free disk block, and so on.
- In our bit map example, we would keep a pointer to fist free block which is block 0. Block 0 would contain the pointer to block 1, which would point to next free block, which is block 2 and so on. In figure 5.5.6 shown, block 0, 1, 2, 4, 6, 8, 11, 12, 13, 14, 16, 17, 19, 21, 22, 24, 26, and 27 are free blocks.
- This method is not efficient because, we must read each block in order to traverse the list, which requires extensive I/O time.
- But advantage is that, traversing the free list is not regular operation.
- Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.
- The FAT method includes free-block accounting into the allocation data structure. No separate method is needed.

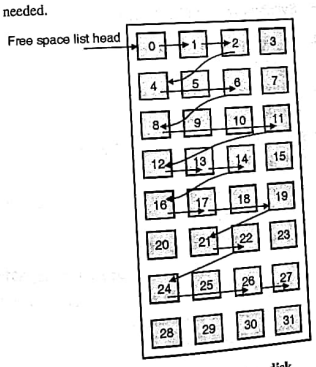


Fig. 5.5.6 : Linked free-space list on disk

Syllabus Topic : Linux Virtual File System

5.6 Linux Virtual File System

Q. Explain Linux virtual File System.

- The object-oriented principles are used to design VFS.
- It has two modules: a set of definitions that states what file-system objects are permissible to seems to be and software layer for these objects manipulation.
- Following four major object types are defined by VFS :
  1. An individual file is represented by i-node object.
  2. An open file is represented by file object.
  3. An entire file system is represented by a superblock object.
  4. An individual directory entry is represented by entry object.
- A set of operations are defined for each of the type of objects. Each object of one of these types points to a function table.
- The record of addresses of the actual functions is kept in function table.
- These functions implement the defined set of operations for that object.
- The VFS software layer need not recognize earlier about what kind of object it is dealing with.
- It can carry out an operation on one of the file-system objects by invoking the right function from the object's function table.
- The VFS remains unaware about whether an i-node represents a networked file, a disk file, a network socket, or a directory file. The function table contains suitable function to read the file.
- The VFS software layer will invoke that function without worrying about the way of reading the data. The file can be accesses with the help of i-node and file objects.
- An i-node object is a data structure that holds pointers to the disk blocks. The actual file data is present on disk block.
- The file object denotes a point of access to the data in an open file. In order to access the i-node' s contents, the process first has to access a file object pointing to the i-node.

- The i-node objects do not belong to single process. Whereas file objects belongs to single process.
- The i-node object is cached by the VFS to get better performance in near future access of the file. Therefore, although process is not currently using the file, its i-node is cached by VFS.
- All cached file data are linked onto a list in the file's i-node object. The i-node also keeps standard information about each file, for example the owner, size, and time most recently modified.
- Directory files are treated in a different way from other files.
- The programming interface of UNIX defines several operations on directories, for example creation, deletion, and renaming of a file in a directory.
- The system calls for these directory operations do not have need of the user open the files concerned, unlike the case for reading or writing data.
- Therefore these directory operations are defined by VFS in the i-node object, instead of in the file object.
- The superblock object represents files of entire file system.
- The operating-system kernel keeps a single superblock object for each disk device mounted as a file system and for each networked file system at present connected. The main duty of the superblock object is to offer access to i-nodes.
- The VFS recognize each i-node by a unique file-system/i-node number pair.
- It locates the i-node analogous to a particular i-node number by requesting the superblock object to return the i-node with that number.
- A entry object represents a directory entry that may include the name of a directory in the path name of a file (such as /usr) or the actual file.

5.7 Exam Pack (University and Review Questions)

Syllabus Topic : File Management - Overview

Q. Explain the different techniques to structure the files. *(Refer section 5.1.3)*

Q. Write short note File Types. *(Refer section 5.1.4)*

Q. Explain various file attributes in brief. *(Refer section 5.1.5)*

Q. Explain various file operations in brief. *(Refer section 5.1.6)*

Syllabus Topic : File Organization and Access

Q. Explain different file access methods. *(Refer section 5.2.1)* (10 Marks) **(June 2015)**

Q. Write short note File Access (sequential access and random access). *(Refer section 5.2.1)*

Q. Explain different types of file organization. *(Refer section 5.2.2)*

Q. Explain pile file organization. *(Refer section 5.2.2(1))*

Q. Explain sequential file organization. *(Refer section 5.2.2(2))*

Q. Explain indexed file organization. *(Refer section 5.2.2(4))*

Q. Explain hashed file organization. *(Refer section 5.2.2(5))*

Syllabus Topic : File Directories

Q. Explain different directory structures. *(Refer section 5.3.1)*

Q. Explain different directory operations. *(Refer section 5.3.3)*

Q. Explain the implementation of directory in detail. *(Refer section 5.3.4)*

Syllabus Topic : File Sharing

Q. Explain issues related to file sharing. *(Refer section 5.4)*

Q. Explain different access rights to the file. *(Refer section 5.4.1)*

Syllabus Topic : Secondary Storage Management

Q. What are the different allocation methods with reference to File Systems? *(Refer section 5.5.4)* (10 Marks) **(Dec. 2014, May 2016)**

Q. Explain contiguous file allocation method with its advantages and disadvantages. *(Refer section 5.5.4(A))*

Q. Explain linked list file allocation method with its advantages and disadvantages. *(Refer section 5.5.4(B))*

Q. Explain indexed file allocation with advantages and disadvantages. *(Refer section 5.5.4(D))*

Q. How i-node is used to allocate the file ? *(Refer section 5.5.4(E))*

Q. Explain bit map method of free space management on secondary storage. *(Refer section 5.5.5(A))*

Q. Explain linked list method of free space management on secondary storage. *(Refer section 5.5.5(B))*

Syllabus Topic : Linux Virtual File System

Q. Explain Linux virtual File System. *(Refer section 5.6)*

□□□

Module 6

CHAPTER 6

Input/Output Management and Disk Scheduling

Syllabus Topics

I/O Devices, Organization of the I/O Function, Operating System Design Issues, I/O Buffering, Disk Scheduling algorithm: FCFS, SSTF, SCAN, CSCAN, LOOK, C-LOOK. Disk Management, Disk Cache, Linux I/O.

Overview

- As we know that I/O is one of the main functions of an operating system and is used to control the entire computer's Input / Output devices.
- Basically it should have to issue commands to the devices, catch interrupts, and handle errors.
- It should also provide an interface between the devices and the rest of the system that is simple and easy to use.
- The I/O code represents a significant fraction of the total operating system.
- Computers operate with many kinds of devices. As we know that it include storage devices (disks, tapes), transmission devices (network cards, modems), and human-interface devices.

Syllabus Topic : I/O Devices

6.1 I/O Devices

Q. Explain different types of I/O devices.

Generally when computer system uses I/O devices as a external devices, these can be grouped into three categories :

1. **Human readable :** It is suitable for communicating with the computer user.

   Examples : printers and terminals, video display, keyboard, and mouse.

2. **Machine readable :** It is suitable for communicating with electronic equipment.

   Examples : Disk drives, USB keys, sensors, controllers, and actuators.

3. **Communication :** It is suitable for communicating with remote devices.

   Examples : Digital line drivers and modems.

- There are many types of devices that work with computer system.
- These are the devices such as disks, tapes, network cards, modems, screen, keyboard, mouse, and many more.
- In order to communicate with a computer system, device sends signals through wire or signals can be wireless.
- The device exchange information with the machine through a connection point termed a port (for example, a serial port).
- If many devices use a common wired communication media, the connection is called a bus.
- When cable of device A plugged into device B, and cable of device B plugged into device C, and device C plugged into a port on the computer, such type of arrangement is called a daisy chain.
- It generally operates as a bus.

Differences between I/O Devices

Q. Explain different parameters to distinguish I/O devices.

Devices can be distinguished with respect to following parameters.

(1) Data transfer rate	(2) Device application
(3) Control complexity	(4) Data transfer unit
(5) Data representation	(6) Errors

1. **Data transfer rate :** Data transfer rate of different devices can be different and can be of several order of magnitudes
2. **Device application :** The applications and use of the different devices varies in the system.
3. **Control complexity :** The complexity required to control the devices are also varies.
4. **Data transfer unit :** The transferring of data can be in stream of bytes or characters or in larger blocks.
5. **Data representation :** The data encoding methods used also varies different devices.
6. **Errors :** The characters of errors vary broadly from one device to another.

Syllabus Topic : Organization of the I/O Function

6.2 Organization of the I/O Function

Q. Explain different techniques to perform I/O.

There are three ways to perform I/O. These are shown in Fig. C6.1.

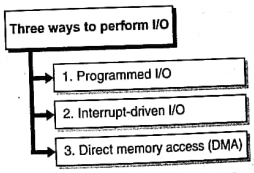


Fig. C6.1 : Ways to perform I/O

➜ 1. Programmed I/O

- In this I/O is the CPU carry out all the work.
- Programmed I/O is straightforward but has the drawback of tying up the CPU permanent until all the I/O is done.
- The CPU issues an I/O command to an I/O module and then that process waits until operation to be finished before proceeding.

➜ 2. Interrupt-driven I/O

- When CPU issues I/O command in support of process two possibilities exist. 0
- If I/O instruction issued by process is nonblocking, then the CPU keeps on executing instructions from the process that issued the I/O command.
- On the contrary, if the I/O instruction is blocking type, then the next instruction that the CPU executes is from the operating system.
- As a result the current process goes in a blocked state and schedule next process.

➜ 3. Direct memory access (DMA)

- The data transfer between main memory and an I/O module is controlled by DMA module.
- The CPU issues a request for the transfer of a block of data to the DMA module.
- In this case CPU cannot be interrupted until complete data is transferred.

6.2.1 The Evolution of the I/O Function

Q. Explain evolutionary steps of I/O function.

Following are evolutionary steps.

- A peripheral device straightway controlled by processor.
- Simple microprocessor-controlled devices use this technique.
- After introduction of controller or I/O module, the processor makes use of programmed I/O with no interrupts.
- As a result, the processor is separated from the specific facts of external device interfaces.
- In above configuration interrupt was introduced. As a result, efficiency is increased as there is no need for processor to wait until I/O operation is finished.
- The I/O module can directly control memory by using DMA.
- A block of data transfer from memory to disk and again disk to memory is handled by I/O module.
- Processor is not engaged in this transfer except at the start and end of the transfer.
- The I/O module is improved to work as separate processor, with a dedicated instruction set customized for I/O only.
- The CPU gives direction to the I/O processor to execute an I/O program in main memory.
- Without involvement of CPU, the I/O processor obtains and executes these instructions.
- As a result, CPU specifies a sequence of I/O activities and it is interrupted only after the complete sequence has been carried out.

- As I/O module has its own local memory, large number of I/O devices can be controlled, with negligible processor participation. The I/O processor looks after the tasks concerned in controlling the terminals.
- From above, it is observed that CPU is not involved in I/O activities and it relieved from it.
- Thereby the performance is improved.
- For I/O to memory transfer, the programmed I/O technique does not use interrupt.
- Whereas for the same interrupt driven I/O use the interrupt. For direct I/O to memory transfer, DMA uses the interrupt.

6.2.2 Direct Memory Access

Q. Explain steps in DMA transfer.

- When any special control unit will be provided to permit transfer of a block of data directly between an external device and the primary memory, without intervention by the processor, called as Direct Memory Access. (DMA).
- It can be used with either polling or interrupt software.
- DMA is rather more practical on devices like disks, where many bytes of information can be transferred in single I/O operations.
- When used along with an interrupt, the CPU is informed only after the entire block of data has been transferred.
- For each byte or word transferred, it must provide the memory address and all the bus signals that control the data transfer.
- Interaction with a device controller is managed through a device driver. Device drivers are part of the operating system.
- The operating system provides a simplified view of the device to user applications (e.g., character devices vs. block devices in UNIX).
- In some operating systems (e.g., Linux), devices are also accessible through the /dev file system.
- In some cases, the operating system buffers data that are transferred between a device and a user space program (disk cache, network buffer).
- This usually increases performance, but not always.
- The DMA controller has access to the system bus independent of the CPU, as shown in Fig. 6.2.1.
- It contains several registers that can be written and read by the CPU.
- These include a memory address register, a byte-count register, and one or more control registers.
- The control registers specify the I/O port to use, the direction of the transfer (reading from the I/O device or writing to the I/O device), the transfer unit (byte at a time or word at a time), and the number of bytes to transfer in one burst.
- Without DMA, let us see how disk read occurs. Initially the controller reads the block from the drive serially, bit by bit, until the complete block is in the controller's internal buffer.
- Next, it calculates the checksum to confirm that no read errors have occurred. Then the controller causes an interrupt.
- After this, operating system can read the disk block from the controller's buffer a byte or a word at a time by executing a loop. In one byte or word is read from controller device register and storing it in main memory.
- With DMA, the process is different. First the CPU programs the DMA controller by setting its registers so it becomes aware about what to transfer where (step 1 in Fig. 6.2.1).
- It also instructs to the disk controller to read data from the disk into its internal buffer and confirm the checksum. DMA can start only when valid data are in the disk controller's buffer.
- The DMA controller begins the transfer by issuing a read request over the bus to the disk controller (step 2).
- The disk controller remains unknown and does not care about whether read request came from the CPU or from a DMA controller.
- Obviously, the bus' address lines contain the memory address to be written.
- Consequently, at the time the disk controller fetches the next word from its internal buffer, it recognizes where to write it.
- The write to operation to memory is one more standard bus cycle denoted by step 3.
- After completion of write, the disk controller sends an ACK signal to the disk controller, also over the bus (Step 4).

- After this, for making use, the DMA controller increments the memory address and byte count is decremented by it.
- At this stage, if the byte count is again more than 0, steps 2 through 4 are carried out again until the count becomes 0.
- At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete.
- After the operating system boots up, it need not to copy the disk block to memory; it is already there. Following Fig. 6.2.1 shows DMA operation.

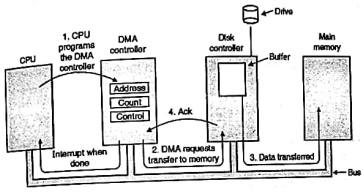


Fig. 6.2.1 : DMA transfer operations

Syllabus Topic : Operating System Design Issues

6.3 Operating System Design Issues

Q. Explain operating systems design issue related to I/O.

6.3.1 Design Objectives

- In the design of the I/O facility, efficiency and generality are the two major objectives.
- As I/O operations are slow and affects the computation, its efficiency is very important.
- Main memory and the processor perform more speedily as compared to many I/O devices.
- Multiprogramming is solution to handle this fact. It permits some processes to be waiting on I/O operations while another process is executing.
- Even though the enough main memory is available but fact is that, I/O is not keeping pace with the performance of the processor.
- To stay the processor busy, processes can be swapped to main memory which is also an I/O operation.
- As a result, I/O design should involve getting better efficiency of the I/O.
- The generality is another important objective. For error free operation and ease, it is advantageous to have a uniform way to handle all devices.
- It is applicable to the way of viewing I/O devices by processes and way of managing I/O devices and operations by operating system.
- Due to the range of device characteristics, it is not easy in reality to get true generality.
- The hierarchical, modular approach to the design of the I/O function hides most of the facts of device I/O in lower-level routines.
- As a result, user processes and upper levels of the operating system notice devices in terms of general functions, for example read, write, open, close, lock, and unlock.

6.3.2 Logical Structure of the I/O Function

Q. Explain logical structure of I/O function.

- As we know operating system is organized in layers, where each layer has its specified function.
- The upper layer interacts with user, where user issues command, may take more time.
- The lower layer dealing with hardware needs to perform in very short time; for example in few billionths of a second.
- Following Fig. 6.3.1 shows organization of I/O facility.
- The details of the organization will rely on the device type and the application.
- The local peripheral device communicates as a stream of bytes or records.
- The following are the layers involved in communication.
  - **Logical I/O :** The logical I/O module is not responsible for concerned the details of controlling the device. It handles the device as a logical resource. The logical I/O module deals with general I/O functions in aid of user processes, permitting them to deal with the device by using a device identifier and simple commands like open, close, read, and write.
  - **Device I/O :** It converts asked operations and data into right sequences of I/O instructions, channel commands, and controller orders. In order to get better utilization, it can make use of buffering strategies.

- **Scheduling and control :** The control of the operations, queuing and scheduling of I/O operation takes place at this layer. Hence, interrupts handling is carried out at this layer and collection and reporting of I/O status is done. It is software layer that in fact interact with the I/O module and therefore the device hardware.

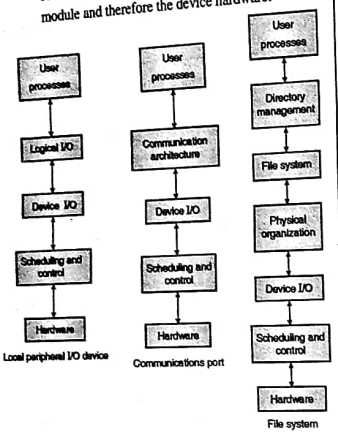


Fig. 6.3.1 : An I/O Organization Model

- If I/O structure of local peripheral device is compared that of with communication device, in second layer communication architecture is present in place of logical I/O.
- Communication I/O consist of number of layers such as TCP/IP.
- The structure for management of I/O on disk that supports file system is shown in Fig. 6.3.1.
- Following are the three new layers in this structure which are not discussed previously.

1. **Directory management :** This layer involves in conversion of symbolic file names to identifiers that either reference the file directly or indirectly through a file descriptor or index table. This layer is responsible for user operations that influence the directory of files; for example add, delete, and reorganize.
2. **File system :** This layer deals with the logical structure of files and with the operations that can be specified by users, such as open, close, read, and write. Access rights are also managed at this layer.
3. **Physical organization :** It is required to translate logical references to files and records to physical secondary storage addresses, by considering the physical track and sector structure of the disk. This layer also takes care of allocation of secondary storage space and main storage buffers.

Syllabus Topic : I/O Buffering

6.4 I/O Buffering

→ (Nov. 15, Dec. 16)

Q. Explain various I/O buffering techniques.
MU - Nov. 2015, 10 Marks, Dec. 2016, 5 Marks

Q. What is I/O buffering? Explain different types of I/O buffering.

- During the course of execution, user process read blocks of data from a disk.
- This data is read into a data area within the address space of the user process.
- To carry out this read operation, user process issues I/O command to disk and then wait for the data to arrive in memory.
- During waiting, the process may constantly test the device status or there can be process suspension on an interrupt.
- There are two troubles in this situation.
- First, due to slow I/O, program gets stuck in waiting until I/O completes.
- The second difficulty is that this type of I/O interferes with operating systems swapping decisions.
- The virtual locations in main memory at which data transfer was intended should remain in memory during the course of the block transfer.
- Otherwise, loss some of data would occur.
- If operating system uses paging then at least the page holding the intended locations for data transfer must be locked into main memory.
- It is not possible to swap the process entirely, though it is preferred by the operating system.
- A single-process deadlock can occur.

- After issuing an I/O command, process is suspended awaiting the I/O completion, and suppose before operation begins, it is swapped out.
- This situation leads to blocking of process which is waiting on the I/O event, and the blocking of I/O operation which is waiting for the process to be swapped in. So deadlock occurs.
- To keep away from this deadlock situation, the user memory concerned in the I/O operation must be locked in main memory without delay before the I/O request is issued, although the I/O operation is queued and may not be executed for a moment.
- The same thoughts have to do with an output operation. If a block transfer occurs from a user process area straight to an I/O module, then the process is blocked all through the transfer and the process may not be swapped out.
- To keep away from these expenses and inefficiencies, it is sometimes suitable to carry out input transfers in advance of requests being made and to carry out output transfers some time after the request is made.
- This method is called as buffering.

Two types of devices

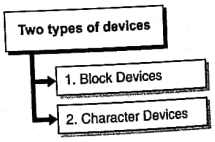


Fig. C6.2 : Types of devices

→ **1. Block Devices**

- It includes all devices that permit random access to entirely independent, fixed-sized blocks of data, together with hard disks and floppy disks, CD-ROMs, and flash memory.
- The use of the block devices is to store file systems.
- However, block devices can be directly accessed so that that the programs can create and repair the file system that the device contains.
- If required, applications can also access these block devices directly.
- Block devices offer the primary interface to every disk devices in a system.
- The block-device system must offer functionality to guarantee that disk access should be done as fast as possible.
- It requires because performance is the major factor for disks.
- The input-output operations scheduling helps to achieve this performance.
- The kernel carry out I/0 in the unit of block. A block is stored in a buffer after it is read into memory, as it is.
- A separate list of requests is kept for each block-device driver.

→ **2. Character Devices**

- These devices comprises devices like mice and keyboards.
- The block devices support for random access, while character devices can be only accessed sequentially.
- A character-device driver does not allow random access to fixed blocks of data.
- Every character-device drivers registered to the Linux kernel needs to be registering a set of functions that implement the file I/0 operations that the driver can handle.
- The kernel does not deal with file read or write request to a character device rather it only passes the request to the device under consideration and lets the device deal with the request.
- This is not true for terminal devices.
- A standard interface is maintained by kernel to these drivers through set of tty_struct structures.
- All these structures offer buffering and flow control on the data stream from the terminal device.
- These structures provide those data to a line discipline which is an interpreter for the information from the terminal device.

6.4.1 Single Buffer

- For the I/O request of user process, the operating system allots a buffer in the system portion of primary memory to this I/O operation.
- If devices are block-oriented, the single buffering technique works as follows :

a. The input is transferred to the system buffer and after completion of transfer, the process shift th block into user space and without delaying request another block.

b. This technique is known as reading ahead, or anticipated input which is carried out in the hope that the block will ultimately be required. Many types of processing require this logical assumption usually because data are as a rule accessed sequentially.

c. Only at the finish of a sequence of processing will a block be read in needlessly.

- The technique described above give a speedup with compare to not having the system buffering.
- When processing of one block of data by user process is going on, at the time the next block is being read in.
- The operating system can swap the process out of memory since the input operation is happening in system memory instead of user process memory.
- This scheme makes complex the logic in the operating system.
- The allocation of system buffers to user processes is the job of operating system.
- In this case, if the I/O operation is carried out on the same disk that is used for swapping, there is no sense to queue disk writes to the same device for swapping the process out.
- This effort of swapping to free main memory will commence until after the I/O operation come to an end, at which time swapping the process to disk may no longer be correct.
- If T_1 is the time required to input one block and that T_2 is the computation time that gets involved between input requests. If no buffers are used, the execution time for each block is basically $T_1 + T_2$.
- If a single buffer used, the execution time is max $[T_1, T_2] + T$, where T is the time elapsed between data transfer from the system buffer to user memory.
- Overall, execution time per block is considerably less with a single buffer compared to no buffer.
- A block-oriented output is considered in the same way. Before transferring data to a device, it is first copied from the user space into the system buffer, and after this it will be written.
- The requesting process is now free to carry on or to be swapped as needed.
- The single buffering technique in stream-oriented I/O is used in the way of line or byte at a time.
- Line at a time operation is suitable for dumb terminals or line printer. These devices consider user input as one line at a time.
- In line at a time input to terminal, a carriage return tells the end of a line, and output to the terminal is likewise one line at a time.
- The Forms-mode terminals, many other peripherals, such as sensors and controllers use byte at a time.
- In line-at-a-time I/O, the buffer is used to hold a single line.

6.4.2 Double Buffer

- In this case, instead single buffer as explained above, two buffers are used in system memory for operation. By doing so, gives improvement over a single buffering.
- The one buffer out of two is used by process to transfer data in both ways (process to buffer and buffer to process).
- At the same time the operating system empties (or fills) the other buffer. This technique is called as double buffering or buffer swapping.
- If $T_2 <= T_1$ then the block-oriented device can be kept going at full speed.
- In contrast, if $T_2 > T_1$, double buffering guarantees that the process need not to wait on I/O.
- In both cases, an enhancement over single buffering is attained. Due to this improvement complexity increases.
- If the input is stream-oriented, then for line at a time I/O, the user process need not be suspended for input or output, unless the process runs ahead of the double buffers.
- In case of byte at a time operation, the double buffer presents no benefit over a single buffer of twice the length.

6.4.3 Circular Buffer

- A double-buffer scheme should maintain flow control of data between an I/O device and a process.
- If the I/O operation is capable of keeping up with the process, then performance can be achieved.
- If the process carry out fast bursts of I/O then double buffering is no sufficient.
- The difficulty can often be eased by using more than two buffers.

- In buffering, the collection of more than two buffers is referred as a circular buffer. In circular buffer, each individual buffer is one unit.

6.4.4 The Utility of Buffering

- If demand for I/O is very high, then buffering helps to fulfill this demand.
- On the other hand, if buffering is not used, it permits an I/O device to match with a process demand for an indefinite period when the average demand of the process is greater than the I/O device can service.
- If we consider the multiple buffers, all of the buffers will finally fill up and the process will have to wait after processing each chunk of data.
- But, in a multiprogramming environment, buffering allows to enhance the efficiency of the operating system and the performance of every process.

Syllabus Topic : Disk Scheduling Algorithm

6.5 Disk Scheduling Algorithms

→ (Nov. 15)

Q. Explain techniques of disk scheduling. **MU - Nov. 2015. 10 Marks**

Firstly, we have to consider, how long it takes to read or write a disk block. The time required is determined by considering three factors:

1. **Seek time :** It is the time to move the arm to the proper cylinder.
2. **Rotational delay :** The time for the proper sector to rotate under the head.
3. **Actual data transfer time**

- For most of the disks, the seek time is greater than rotational delay and actual data transfer time.
- Hence by minimizing mean seek time, it is possible to enhance system performance significantly.
- If the processing of requests by disk driver is in First-Come, First-Served (FCFS) basis then it is not possible to optimize seek time.
- Yet, another approach is possible in case of heavily loaded disk.
- When the arm is seeking for one request, other disk requests may be produced by other processes.
- Several disk drivers keep a table which is indexed by cylinder number. In this table, all the pending requests for each cylinder chained together in a linked list headed by the table entries.

Syllabus Topic
Disk Scheduling Algorithm - FCFS

6.5.1 FCFS Scheduling Algorithm

→ (Dec. 16)

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. **MU - Dec. 2016. 10 Marks**

- The first-come, first-served (FCFS) algorithm is the simplest scheduling algorithm.
- This algorithm is basically fair. This algorithm in general does not offer the fastest service.
- Consider the disk queue with requests for I/O to blocks on cylinders in that order.

  96, 185, 35, 122, 16, 120, 55, 57

  Queue = 96, 185, 35, 122, 16, 120, 55, 57

  Head start at 51

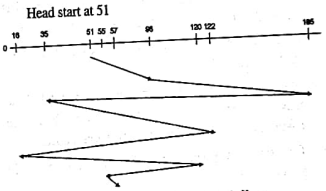


Fig. 6.5.1 : FCFS disk Scheduling

- Initially, the disk head is at cylinder 51. It will then first move from 51 to 96, then to 185, 35, 122, 16, 120, 55, and at last to 57, for a total head movement of **648** cylinders.
- This schedule is diagrammed in above Fig. 6.5.1.
- The larger swing from 122 to 16 and then back to 120 demonstrates the problem with this schedule.
- If the requests for cylinders 35 and 16 could be serviced together, before or after the requests at 122 and 120, the total head movement could be decreased considerably, and performance could be thereby enhanced.

Syllabus Topic
Disk Scheduling Algorithm - SSTF

6.5.2 Shortest-Seek-Time-First (SSTF) Scheduling Algorithm

→ (Dec. 16)

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. **MU - Dec. 2016. 10 Marks**

Q. Explain SSTF disk scheduling algorithm.

- This algorithm services all the requests near to the current head position before moving the head far away to service other requests.
- The SSTF algorithm chooses the request with the minimum seek time from the current head position.
- As seek time increases with the number of cylinders pass through by the head, SSTF chooses the awaiting request closest to the current head position.
- In our example, closest request to the original head position (51) is at cylinder 55.
- Once we are at cylinder 55, the next closest request is at cylinder 57. From there, the request at cylinder 35 is nearer than the one at 96, so 35 is served next.
- Continuing, we service the request at cylinder 16, then 96,120, 122, and finally 185.
- This scheduling method results in a total head movement of only 316 cylinders, very less with compare to FCFS scheduling of this request queue.
- This algorithm gives a considerable improvement in performance.
- Just like a shortest-job-first (SJF) scheduling, SSTF may lead to starvation of some requests.
- Remember that requests may arrive at any time.
- Suppose that we have two requests in the queue, for cylinders 16 and 185, and while servicing the request from 16, a new request near 16 arrives.
- This new request will be serviced next, making the request at 185 wait.
- While this request is being serviced, another request close to 16 could arrive. In theory, a continual stream of requests near one another could arrive, causing the request for cylinder 185 to wait indefinitely.

Queue = 96, 185, 35, 122, 16, 120, 55, 57

Head start at 51

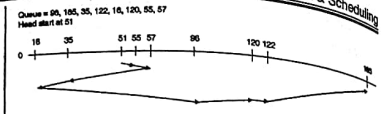


Fig. 6.5.2 : SSTF disk Scheduling

Syllabus Topic
Disk Scheduling Algorithm - SCAN

6.5.3 SCAN Scheduling Algorithm

→ (Dec. 16)

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. **MU - Dec. 2016. 10 Marks**

- The movement of the disk arm in this algorithm is from one end of the disk and goes in the direction of the other end.
- While doing so, it service requests as it arrive at each cylinder, until it arrive at the other end of the disk.
- After reaching at the other end, the direction of head movement gets reversed, and servicing continues.
- The head repetitively scans backward and forward across the disk.
- The disk arm acts just like an elevator in the building. Initially it services all the requests while going up and when reverse the direction towards down, it service the requests on this way.
- Therefore the SCAN algorithm is also called as the **elevator algorithm.**
- Let's consider the same example discussed above.
- Before proceeding with SCAN to schedule the requests on cylinders 96, 185, 35, 122, 16, 120, 55, and 57, the direction of head movement should be known along with the head's present position (51).
- If movement of the disk arm is towards direction of 0, the head first will service 35 and then 16.
- After reaching at cylinder 0, the arm will reverse and will go toward the other end of the disk, servicing the requests at 55, 57, 96, 120, 122, and 185.
- If the incoming request in queue is just in front of the head, then it will be serviced almost without delay; a incoming request just at the back the head need to wait until the arm goes to the end of the disk, reverses direction, and comes back.
- Considering an even allotment of requests for cylinders, consider the density of requests when the head arrives at one end and" reverses direction.

- At this moment, comparatively a small number of requests are right away in front of the head, since these cylinders have just been serviced.
- The density of requests is very heavy is at the other end of the disk.
- These requests have also waited for the longer period of time, so why not go there first? That is the consideration of the next algorithm.

Queue = 96, 185, 35, 122, 16, 120, 55, 57

Head start at 51

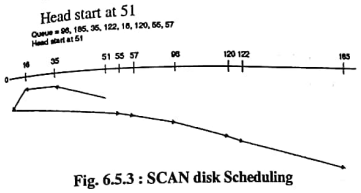


Fig. 6.5.3 : SCAN disk Scheduling

Syllabus Topic
Disk Scheduling Algorithm - CSCAN

6.5.4 C-SCAN Scheduling Algorithm

→ (Dec. 16)

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. **MU - Dec. 2016. 10 Marks**

- C-SCAN is called as Circular SCAN scheduling and it is a variation of SCAN.
- It offers a more uniform wait time. Just like the SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.
- When the head arrive at the other end, however, it straight away returns to the beginning of the disk, without servicing any requests on the return trip.
- The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Queue = 96, 185, 35, 122, 16, 120, 55, 57

Head start at 51

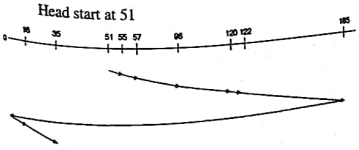


Fig. 6.5.4 : C-SCAN disk Scheduling

Syllabus Topic
Disk Scheduling Algorithm : LOOK, C-LOOK

6.5.5 LOOK Scheduling Algorithm

→ (Dec. 16)

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. **MU - Dec. 2016. 10 Marks**

Q. Explain LOOK disk scheduling algorithms.

- Above two algorithms, SCAN and C-SCAN, move the disk arm across the full width of the disk.
- Normally, the arm goes only as far as the final request in each direction.
- Then, it reverses direction straight away, without going all the way to the end of the disk.
- Versions of SCAN and C-SCAN that follow this above pattern are called **LOOK** and **C-LOOK scheduling,** because they *look* for a request before continuing to move in a given direction.

Queue = 96, 185, 35, 122, 16, 120, 55, 57

Head start at 51

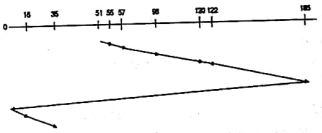


Fig. 6.5.5 : LOOK disk scheduling

6.6 Examples on Disk Scheduling Algorithms

Example 6.6.1 **MU - Dec. 2014. 10 Marks**

Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 Starting from the current head position, what is the total distance ((in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling Algorithms.

1. FCFS
2. SCAN

Solution :

The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. The total seek distance is 7081.

The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86. The total seek distance is 9769

Example 6.6.2

Disk requests come in to the disk for cylinders 10, 22, 20, 2, 40, 6 and 38. A seek takes 6 msec per cylinder move. How much seek time is for Closest cylinder next algorithm? Initially arm is at cylinder 20.

Solution :

First-Come, First-Served algorithm accepts requests once at a time and caries them out in that order;

Total time

((20 – 10) + (22 – 10) + (22 – 20) + (20 – 2) + (40 – 2) + (40 – 6)+ (38 – 6)) * 6 msec = 146 * 6 msec = 876 msec

Cylinder Seek Algorithms : SSF Shortest Seek First algorithm always handles the closest request first, to minimise seek time, independently on the requests order;

Total time

((20 – 20) + (22 – 20) + (22 – 10) + (10 – 6) + (6 – 2) + (38 – 2) + (40 – 38)) * 6 msec = 60 * 6 msec = 360 msec

Cylinder Seek Algorithms

Elevator algorithm handles nearest requests in increasing (decreasing) order until no more requests on the way. Afterwards the direction is reversed

Total time

((20 – 20) + (22 – 20) + (38 – 22) + (40 – 38) + (40 – 10) + (10 – 6) + (6 – 2)) * 6 msec = 58 * 6 msec = 348 msec

Example 6.6.3

Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

a. FCFS b. SSTF c. SCAN d. LOOK
e. C-SCAN f. C-LOOK

Solution :

a. The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022,1750, 130.

The total seek distance is 7081.

b. The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750,1774.

The total seek distance is 1745.

c. The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,4999, 130, 86.

The total seek distance is 9769.

d. The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,130, 86.

The total seek distance is 3319.

e. The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,4999, 86, 130.

The total seek distance is 9813.

f. (Bonus) The C-LOOK schedule is 143, 913, 948, 1022, 1470, 1509,1750, 1774, 86, 130.

The total seek distance is 3363.

Example 6.6.4

None of the disk-scheduling disciplines, except FCFS, is truly fair (starvation may occur).

a. Explain why this assertion is true.
b. Describe a way to modify algorithms such as SCAN to ensure fairness.
c. Explain why fairness is an important goal in a time-sharing system.
d. Give three or more examples of circumstances in which it is important that the operating system be unfair in serving I/O requests.

Solution :

a. New requests for the track over which the head currently resides can theoretically arrive as quickly as these requests are being serviced.

b. All requests older than some predetermined age should be compulsorily placed the top of the queue, and an associated bit for each could be set to specify that no new request could be moved ahead of these requests. For SSTF, the rest of the queue would have to be reorganized with respect to the last of these "old" requests.

c. To avoid unusually long response times.

d. User requests should have lower priority over paging and swapping. It may be desirable for other kernel-initiated I/O, such as the writing of file system metadata, to take priority over user I/O. If the kernel supports real-time process priorities, the I/O requests of those processes should be favoured.

Example 6.6.5

Disk requests come into the disk for cylinders 10, 22, 20, 2, 40, 6 and 38. A seek takes 6 millisecond per cylinder move. How much seek time is for Closest cylinder next algorithm? Initially arm is at cylinder 20.

Solution :

First - Come, First - Served algorithm accepts requests once at a time and caries them out in that order;

Total time

((20 – 10) + (22 – 10) + (22 – 20) + (20 – 2) + (40 – 2) + (40 – 6) + (38 – 6)) * 6 msec = 146 * 6 msec = 876 msec.

Cylinder Seek Algorithms

SSF Shortest Seek First algorithm always handles the closest request first, to minimise seek time, independently on the requests order;

Total time

((20 – 20) + (22 – 20) + (22 – 10) + (10 – 6) + (6 – 2) + (38 – 2) + (40 – 38)) * 6 msec = 60 * 6 msec = 360 msec.

Cylinder Seek Algorithms

Elevator algorithm handles nearest requests in increasing (decreasing) order until no more requests on the way. Afterwards the direction is reversed.

Total time

((20 – 20) + (22 – 20) + (38 – 22) + (40 – 38) + (40 – 10) + (10 – 6) + (6 – 2)) * 6 msec = 58 * 6 msec = 348 msec.

Example 6 6.6

Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 155 and the previous request was at cylinder 170. The queue of pending requests is 86, 1350, 948, 130, 1500, 50. Starting from the current head position, what is the total distance ((in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling Algorithms.

1. FCFS 2. SSTF 3. SCAN 4 C-SCAN
5. LOOK 6. C-LOOK

Solution :

1. FCFS

The FCFS schedule is 155,86,1350,948, 130,1500,50

It gives

(155 – 86) = 69, (1350 – 86) = 1264, (1350 – 948) = 402, (948 – 130) = 818, (1500 – 130) = 1370, (1500 – 50) = 1450

Total head movements are :

69 + 1264 + 402 + 818 + 1370 + 1450 = 5373 Cylinders

2. SSTF :

The SSTF schedule is 155,130,86, 50,948,1350,1500

It gives (155 – 130) = 25, (130 – 86) = 44, (86 – 50) = 36, (948 – 50) = 898, (1350 – 948) = 402, (1500 – 1350) = 150

Total head movements are :

25 + 44 + 36 + 898 + 402 + 150 = 1555 Cylinders

3. SCAN :

The SCAN schedule is 155, 130, 86, 50, 0, 948, 1350, 1500

It gives (155 – 130) = 25, (130 – 86) = 44, (86 – 50) = 36, (50 – 0) = 50, (948 – 0) = 948, (1350 – 948) = 402, (1500 – 1350) = 150

Total head movements are :

25 + 44 + 36 + 50 + 948 + 402 + 150 = 1655 Cylinders

4. C – SCAN :

The C-SCAN schedule is 155, 130, 86, 50, 0, 4999, 1500, 1350, 948.

It gives (155 – 130) = 25, (130 – 86) = 44, (86 – 50) = 36, (50 – 0) = 50, (4999 – 0) = 4999, (4999 – 1500) = 3499, (1500 – 1350) = 150, (1350 – 948) = 402.

Total head movements are :

25 + 44 + 36 + 50 + 4999 + 3499 + 150 + 402 = 9205 Cylinders

5. C – LOOK : The C - LOOK schedule is 155,130,86, 50,1500, 1350, 948

It gives (155 – 130) = 25, (130 – 86) = 44, (86 – 50) = 36,(1500 – 50) = 1450, (1500 – 1350) = 150 , (1350 – 948) = 402.

Total head movements are :

25 + 44 + 36 + 1450 + 150 + 402 = 2107 Cylinders

6. LOOK : The LOOK schedule is 155,130,86, 50 , 948, 1350, 1500

It gives (155 – 130) = 25, (130 – 86) = 44, (86 – 50) = 36, (948 – 50) = 898, (1350 – 948) = 402 , (1500 – 1350) = 150.

Total head movements are :

25 + 44 + 36 + 898 + 402 + 150 = 1555 Cylinders.

Example 6.6.7

Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143,

and the previous request was at cylinder 125. The queue of pending requests in FIFO is ordered as 80, 1470, 913, 1777, 948, 1022, 1750,130. What is the total distance that the disk arm moves for following.

1. FCFS 2. SSTF 3. LOOK 4. SCAN 5. C-SCAN

Solution :

1. **FCFS :** The FCFS schedule is

   143, 80, 1470, 913, 1777, 948, 1022, 1750,130 .

   It gives (143 – 80) = 63, (1470 – 80) = 1390, (1470 – 913) = 557,

   (1777 – 913) = 864, (1777 – 948) = 829,

   (1022 – 948) = 74, (1750 – 1022) = 728,

   (1750 – 80) = 1670

   Total head movements are :

   63 + 1390 + 557 + 864 + 829 + 74 + 728 + 1670 = 6125 Cylinders

2. **SSTF :** The SSTF schedule is

   143,130, 80, 913, 948, 1022, 1470, 1750, 1777 .

   It gives (143 – 130) = 13, (130 – 80) = 50, (913 – 80) = 833, (948 – 913) = 35, (1022 – 948) = 74, (1470 – 1022) = 448, (1750 – 1470) = 280, (1777 – 1750) = 27

   Total head movements are :

   13 + 50 + 833 + 35 + 74 + 448 + 280 + 27 = 1760 Cylinders

3. **LOOK :** The LOOK schedule is

   143, 913, 948, 1022, 1470, 1750, 1777,130, 80.

   It gives (913 – 143) = 770, (948 – 913) = 35, (1022 – 948) = 74, (1470 – 1022) = 448, (1750 – 1470) = 280, (1777 – 1750) = 27, (1777 – 130) = 1647,(130 – 80) = 47

   Total head movements are

   770 + 35 + 74 + 448 + 280 + 27 + 1647 + 50 = 3331 Cylinders

4. **SCAN :** The SCAN schedule is

   143, 913, 948, 1022, 1470, 1750,1777, 4999, 130, 80.

   It gives (913 – 143) = 70, (948 – 913) = 35, (1022 – 948) = 74, (1470 – 1022) = 448, (1750 – 1470) = 280, (1777 – 1750) = 27, (4999 – 1777) = 3222,(4999 – 130) = 4869, (130 – 80) = 50

   Total head movements are :

   770 + 35 + 74 + 448 + 280 + 27 + 3222 + 4869 + 50 = 9775 Cylinders

5. **C-SCAN :** The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1750,1777, 4999, 0, 80, 130.

   It gives (913 – 143) = 770, (948 – 913) = 35, (1022 – 948) = 74,

   (1470 – 1022) = 448, (1750 – 1470) = 280,

   (1777 – 1750) = 27,(4999 – 1777) = 3222,

   (4999 – 0)= 4999, (80 – 0) = 80, (130 – 80) = 50

   Total head movements are :

   770 + 35 + 74 + 448 + 280 + 27 + 3222 + 4999 + 80 + 50 = 9985 Cylinders

Example 6.6.8

Suppose that a disk drive has 200 cylinders, numbered 0 to 199. The initial head position is at 100th track . The queue of pending requests in FIFO is 55, 58, 39, 18 90, 160, 150, 38, 184. Calculate average seek time for each of the following algorithm.

1. FCFS 2. SSTF 3. SCAN
4. LOOK 5. C-SCAN 6. C-LOOK

Solution :

1. **FCFS :** The FCFS schedule is

   100, 55, 58, 39, 18, 90, 160, 150, 38, 184.

   It gives (100 – 55) = 45, (58 – 55) = 3, (58–39) = 19, (39 – 18) = 21, (90 – 18) = 72, (160 – 150) = 10, (150 – 38) = 112, (184 – 38) = 146

   Total head movements are :

   45 + 3 + 19 + 21 + 72 + 10 + 112 + 146 = 428 Cylinders

2. **SSTF :** The SSTF schedule is

   100, 90, 58, 55, 39, 38, 18, 150, 160, 184

   It gives (100 – 90) = 10, (90 – 58) = 32, (58 – 55) = 3, (55 – 39) = 16, (39 – 38) = 1 , (38 – 18) = 20, (150 – 18) = 132, (160 – 150) = 10, (184 – 160) = 24

   Total head movements are :

   10 + 32 + 3 + 16 + 1 + 20 + 132 + 10 + 24 = 248 Cylinders

3. **SCAN :** The SCAN schedule is

   100, 90, 58, 55, 39, 38, 18, 0, 150, 160, 184

   It gives (100 – 90) = 10, (90 – 58) = 32, (58 – 55) = 3, (55 – 39) = 16, (39 – 38) = 1 , (38 – 18) = 20,(18 – 0) = 18, (150 – 0) = 150, (160 – 150) = 10,(184 – 160) = 24

Total head movements are :

10 + 32 + 3 + 16 + 1 + 20 + 18 + 150 + 10 + 24 = 284 Cylinders

4. **LOOK :** The LOOK schedule is

   100, 90, 58, 55, 39, 38, 18, 150, 160, 184

   It gives (100 – 90) = 10, (90 – 58) = 32, (58 – 55) = 3, (55 – 39) = 16, (39 – 38) = 1 , (38 – 18) = 20, (150 – 18) = 132, (160 – 150) = 10,(184 – 160) = 24

   Total head movements are :

   10 + 32 + 3 + 16 + 1 + 20 + 132 + 10 + 24 = 248 Cylinders

5. **C - SCAN :** The C – SCAN schedule is

   100, 150, 160, 184, 199, 0, 18, 38, 39, 55, 58, 90

   It gives (150 – 100) = 50, (160 – 150) = 10, (184 – 160) = 24, (199 – 184) = 15, (199 – 0) = 199, (18 – 0) = 18, (38 – 18) = 20, (39 – 38) = 1, (55 – 39) = 16, (58 – 55) = 3, (90 – 58) = 32

   Total head movements are :

   50 + 10 + 24 + 15 + 199 + 18 + 20 + 1 + 16 + 3 + 32 = 388 Cylinders

6. **C-LOOK :** The C – SCAN schedule is

   100, 150, 160, 184, 18, 38, 39, 55, 58, 90

   It gives (150 – 100) = 50, (160 – 150) = 10, (184 – 160) = 24, (184 – 18) = 166, (38 – 18) = 20, (39 – 38) = 1, (55 – 39) = 16, (58 – 55) = 3, (90 – 58) = 32

   Total head movements are :

   50 + 10 + 24 + 166 + 20 + 1 + 16 + 3 + 32 = 322 Cylinders

Example 6.6.9

Suppose that the head of a moving disk with 200 tracks, numbered 0 to 199, is currently serving a request at track 143 and has just finished a request at track 125. The queue of requests is kept in the FIFO order:

86, 147, 91, 177, 94, 150, 102, 175, 130

What is the total number of Head movements needed to satisfy these requests for the following disk scheduling algorithms:

1. FCFS 2. SSTF 3. Elevator

Solution :

1. **FCFS :** The FCFS schedule is

   143,86, 147, 91, 177, 94, 150, 102, 175, 130

   It gives (143 – 86) = 57, (147 – 86) = 61, (147 – 91) = 56, (177 – 91) = 86, (177 – 94) = 83, (150 – 94) = 56 , (150 – 102) = 48,

   (175 – 102) = 73, (175 – 130) = 45

   Total head movements are :

   57 + 61 + 56 + 86 + 83 + 56 + 48 + 73 + 45 = 565 Cylinders.

2. **SSTF :** The SSTF schedule is 143, 147, 150, 130, 102, 94, 91, 86, 175, 177

   It gives (147 – 143) = 4, (150 – 147) = 3, (150 – 130) = 20, (130 – 102) = 28, (102 – 94) = 8, (94 – 91) = 3, (91 – 86) = 5, (175 – 86) = 89, (177 – 175) = 2

   Total head movements are :

   4 + 3 + 20 + 28 + 8 + 3 + 5 + 89 + 2 = 162 Cylinders

3. **Elevator :** The elevator schedule is

   143, 147, 150, 175, 177, 199, 130, 102, 94, 91, 86

   It gives (147 – 143) = 4, (150 – 147) = 3, (175 – 150) = 25, (177 – 175) = 2, (199 – 177) = 22, (199 – 130) = 69, (130 – 102) = 28, (102 – 94) = 8, (94 – 91) = 3, (91 – 86) = 5

   Total head movements are :

   4 + 3 + 25 + 2 + 22 + 69 + 28 + 8 + 3 + 5 = 169 Cylinders.

Example 6.6.10 MU - May 2016. 10 Marks

Assume that the disk head is initially positioned over track 100. For the disk space request of 27, 129, 110, 186, 147, 41, 10, 64 and 120. Show how disk scheduling is carried out for SSTF, C-SCAN, C-LOOK. Calculate the average seek length and show the tracing of the requests.

Solution :

SSTF Schedule is: 100, 110, 120, 129, 147, 186, 64, 41, 27, 10

(110 - 100) + (120-110) + (129-120) + (147-129) + (186-147) + (186-64) + (64-41) + (41-27) + (27-10) = 10 + 10 + 9 + 18 + 39 + 122 + 23 + 14 + 17 = 262/9 = Average 29.11

C-SCAN Schedule is: 100, 64, 41, 27, 10, 186, 147, 129, 120, 110

(100 - 64) + (64-41) + (41-27 + (27-10) + (186-10) + (186-147) + (147-129) + (129-120) + (120-110) = 36+23+14+17+176+39+18+9+10 = 342/9 = Average 38

C-LOOK Schedule is : 100, 110, 120, 129, 147, 186, 10, 27, 41, 64

(110 - 100) + (120-110) + (129-120) + (147-129) + (186-147) + (186-10) + (27-10) + (41-27) + (64-41) = 10+10+9+18+39+176+17+14+23 = 316/9 = Average 35.11

Example 6.6.11 MU - June 2015. 10 Marks

The requested tracks in the order received are 54, 57, 40, 20, 80, 120, 150, 45, 180. Apply the following disk scheduling algorithms. Starting track is 90.

(i) FCFS (ii)SSTF (iii) C-SCAN (Section 5.7, example 10)

Solution :

(i) FCFS

FCFS schedule is 90, 54, 57, 40, 20, 80, 120, 150, 45, 180

It gives (90-54) + (57-54) + (57-40) + (40-20) + (80-20) + (120-80) + (150-120) + (150-45) + (180-45)

Total head movements are

36+3+17+20+60+40+30+105+35 = 343

(ii) SSTF

SSTF schedule is 90, 80, 57, 54, 45, 40, 20,120, 150, 180

It gives (90-80) + (80-57) + (57-54) + (54-45) + (45-40) + (40-20) + (120-20) + (150-120) + (180-150)

Total head movements are

10 + 23 + 3 + 9 + 5 + 20 + 100 + 30 + 30 = 230

(ii) C-SCAN

C-SCAN schedule is 90, 80, 57, 54, 45, 40, 20, 0, 120, 150, 180

It gives (90-80) + (80-57) + (57-54) + (54-45) + (45-40) + (40-20) + (20-0) + (120-0) + (150-120) + (180-150)

Total head movements are

10 + 23 + 3 + 9 + 5 + 20 + 20 + 120 + 30 + 30 = 27

Example 6.6.12 MU - Dec. 2015. 10 Marks

On a disk with 1000 cylinders numbered 0 to 999 compute the number of tracks, the disk arm must move to satisfy all the requests in the disk queue. Assume the last request service was at track 345 and the head is moving toward track 0. The queue in FIFO order contains requests for the following tracks : 123, 874, 692, 475, 105, 376 . Perform the computation for the following scheduling algorithms.

i) FIFO ii) SSTF iii) SCAN

Solution :

(i) FIFO

The FIFO schedule is 345,123,874,692,475,105 and 376.

Total head movement = (345-123) + (874-123) + (874-692) + (692-475) + (475-105) + (376-105) = 2013

(ii) SSTF

The SSTF schedule is 345,376,475, 692,874, 123 and 105

Total head movement = (376-345) + (475-376) + (692-475) + (874-692) + (874-123) + (123-105) =1298.

(iii) SCAN

345, 123, 105, 0, 376, 475, 692 and 874

Total head movement = (345-123) + (123-105) + (105-0) + (0-376) + (376-475) + (475-692) + (692-874) =1219.

Syllabus Topic : Disk Management

6.7 Disk Management

The disk management activities of the operating system include disk initialization, booting from disk, and bad-block recovery.

6.7.1 Disk Formatting

Q. Explain disk formatting activity of disk management of operating system.

- If the magnetic disk is new then it is a platter of a magnetic recording material.
- Directly on such surface, we cannot store data.
- The surface needs to be divided in sectors so that disk controller is able to perform read and write operations.
- The procedure of doing this is called low level or physical formatting.
- This formatting fills each sector with data structures :
  1. Header
  2. A data area (usually 512 bytes in size)
  3. Trailer
- The header and trailer hold the information used by disk controller such as error correcting codes ECC and sector number.
- ECC is updated with a value calculated from all the bytes in the data.
- This is done during the controller writes a sector of data at the time of normal I/O, the area.
- The ECC is recalculated at the time sector is read and compared with the stored value.
- If the stored and calculated numbers matches data is correct and not corrupted.

- If mismatch occurs, it indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- The ECC contains sufficient information, if few bits of data have been corrupted, it facilitate the controller to recognize which bits have changed and calculate what their correct values should be. It then reports a recoverable soft error.
- The controller automatically does the ECC processing whenever a sector is read or written.
- During the manufacturing process, low level formatting of hard disk is carried out.
- It helps to manufacturer to initialize the mapping from logical block numbers to defect-free sectors on the disk.
- For several hard disks, the disk controller is instructed for how many bytes of data space to leave between the header and trailer of all sectors at the time when it perform low level formatting.
- For example, the sizes can be 256,512, and 1,024 bytes. If disk is formatted with larger sector size, fewer sectors can fit on each track.
- As a result fewer headers and trailers are written on each track and more space is obtainable for user data.
- Some operating systems can handle a sector size of 512 bytes.
- Operating system keeps its own data structures on disk before it use disk to store the files. It performs this with following two steps :
  1. It partitions the disk into one or more groups of cylinders. Each partition is treated by OS as a separate disk.
  2. **Logical formatting :** That means creation of file system.
- In order to increase the efficiency, file system groups blocks in chunks called as clusters.
- Some operating systems give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures.
- This array is sometimes called the raw disk, and I/O to this array is called as raw I/O.

6.7.2 Boot Block

Q. Explain boot block activity of disk management of operating system.

- The bootstrap program is required for a computer to initiate the booting after it is powered up or rebooted.
- It initializes all components of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.
- The bootstrap program then locates the OS kernel on disk, loads that kernel into memory, and jumps to an initial address to start the operating-system execution.
- The Read Only Memory (ROM) does not require initialization and is at a fixed location that the processor can begin executing when powered up or reset. Therefore bootstrap is stored in ROM.
- Because of read only feature of ROM; it cannot be infected by a computer virus.
- The difficulty is that modification of this bootstrap code requires changing the ROM hardware chips.
- Therefore, most systems store a small bootstrap loader program in the boot ROM which invokes and bring full bootstrap program from disk into main memory.
- The modified version of full bootstrap program can be simply written onto the disk.
- The fixed storage location of full bootstrap program is in the "boot blocks".
- A disk that has a boot partition is called a boot disk or system disk.
- Boot ROM code gives instruction to controller for reading the boot blocks into memory (no device drivers are loaded at this point) and then starts executing that code.
- The full bootstrap program is more sophisticated than the bootstrap loader in the boot ROM; it is able to load the complete operating system from a non-fixed location on disk and to start the operating system execution. Yet, the full bootstrap code may be small.

6.7.3 Bad Blocks

Failure of the disk can be :

1. Complete, means there is no way other than replacing the disk. Back up of content must be taken on new disk.
2. One or more sectors become faulty.
3. After manufacturing, the bad blocks exist. Depending on the disk and controller in use, these blocks are handled in a different ways.

- If the disks are with IDE controllers then these are simple disks. In these disks bad blocks are handled manually.
- For example, the MS-DOS format command carry out logical formatting and, as a part of the process, scans the disk to discover bad blocks.
- If format discovers a bad block, it writes a special value into the related FAT entry to inform the allocation routines not to utilizethat block.
- If blocks go bad du ring normal operation, a special program (such as chkdsk) must be run manually to look for the bad blocks and to lock them away.
- Data that resided on the bad blocks generally are lost.
- The SCSI disks are used in high-end PCs and most workstations and servers, are more intelligent about recovery of bad-block.
- The controller preserves a record of bad blocks on the disk.
- The list is initialized during the low-level formatting at the factory and is updated over the life of the disk. Low-level formatting also sets aside spare sectors not visible to the operating system.
- The controller can be told to replace each bad sector logically with one of the spare sectors. This is called as sector sparing or forwarding.
- A usual dealing with bad-sector is like follows :
  a. The OS attempt to read logical block 67.
  b. The controller computes the ECC and concludes that the sector is bad. The same result is reported to OS.
  c. When the system is rebooted, a special command is run to inform the SCSI controller to replace the bad sector with a spare.
  d. Afterward, at whatever time the system requests logical block 67, the request is translated into the replacement sector's address by the controller.

Syllabus Topic : Disk Cache

6.8 Disk Cache

→ (Nov. 15)

Q. Explain disk cache. MU - Nov. 2015, 5 Marks

- In main memory, a buffer is reserved for disk sectors called as disk cache.
- The cache keeps a copy of data in some of the sectors on the disk.
- To fulfill an I/O request for a particular sector, first disk cache is checked to see whether the sector is in disk cache.
- If it is present, then request is fulfilled via the cache.
- If not present, then required sector is read into the disk cache from the disk.
- The disk cache improves the performance as some request is satisfied from it.
- The property of locality of reference is used .
- When a block of data is present in cache to fulfill a single I/O request, it is expected that same block will be needed in future.

Design Issues

- In first design issue, after the I/O request is fulfilled from the disk cache, there are two ways to deliver the data in the disk cache to the requesting process.
- In first approach, the block of data can be transferred from the disk cache to memory allocated to the user process.
- In second approach, a shared memory is used and a pointer is passed to the right slot in the disk cache.
- The second approach save the time of a memory-to-memory transfer and also permits shared access by other processes.
- A second design issue concern with the replacement of existing block from disk cache.
- It is required to create space for a newly transferred sector into the disk cache. This is analogues to page replacement in memory management.
- A page replacement algorithm is required to replace the page. The least recently used (LRU) is used most commonly.
- This policy suggests that replace the block in cache that resides in for longest period of time without reference.
- A stack of the blocks is maintained with recently used at top of stack.
- When a block is brought in from disk, the block at the bottom of the stack is replaced incoming block is kept onto the top of the stack.
- A stack of pointers to the blocks can be associated with the cache.

- Other algorithm is least frequently used (LFU) which suggest replacing block in the set that has minimum references.
- The implementation of LFU can be done by assigning counter with each block.
- When a block is transferred in memory, it is assigned initially a count of 1.
- After this each time it is referenced a count is incremented by 1. A block with the smallest count is selected for replacement.

Syllabus Topic : Linux I/O

6.9 Linux Input Output

Q. Explain Linux I/O.

- Linux input/output is very similar to SRV4 which we have discussed.
- The Linux kernel associates a special file with each I/O device driver.
- It also recognizes Block, character, and network devices.

6.9.1 Disk Scheduling

- The disk scheduler in Linux 2.4 is called the Linux Elevator.
- In Linux 2.6, the elevator algorithm has been supplemented by two further algorithms which are the deadline I/O scheduler and the anticipatory I/O scheduler.

6.9.1(A) The Elevator Scheduler

Q. Explain elevator disk scheduling algorithm.

- The elevator schedulers keep a single queue for disk read and write requests.
- It also accomplishes both sorting and merging functions on the queue. Usually, the elevator scheduler maintains the list of requests sorted by block number.
- Thus, as the disk requests are handled, the drive moves in a single direction, fulfilling each request as it is came across.
- When a new request is added to the queue, four operations are considered in order :
  (a) If the request is to the same disk sector or a directly neighbouring sector to a awaiting request in the queue, then the existing request and the new request are merged into one request.
  (b) If a request in the queue is adequately old, the new request is added at the end of the queue.
  (c) If there is a appropriate location, the new request is added in sorted order.
  (d) If there is no appropriate location, the new request is positioned at the end of the queue.

Deadline Scheduler

- Second operation given above, is to avoid starvation of a request but is not very effectual.
- It does not service requests in an agreed time structure but simply stop insertion and sorting requests after a appropriate delay.
- There are two problems in the elevator scheme.
- The first problem is that a far-away block request can be postponed for a considerable time because the queue is dynamically updated.
- For example, consider the following flow of requests for disk blocks number 10, 13, 618, and 22.
- The elevator scheduler rearranges these in order that the requests are positioned in the queue as 10, 13, 22, 618, with 10 being the head of the queue.
- If a constant sequence of low-numbered block requests comes then the request for 618 goes on to be deferred.
- Still additional severe problem concerns the difference between read and write requests.
- Usually, a write request is issued asynchronously.
- If process issues the write request, it is not necessary to wait for the request to in fact be fulfilled.
- When an application issues a write, the kernel copies the data into a suitable buffer, to be written out as time allows.
- Application only can continue after data gets copied in kernel buffer.
- In contrast to this, for many read operations, the process should wait until the requested data are delivered to the application before proceeding.

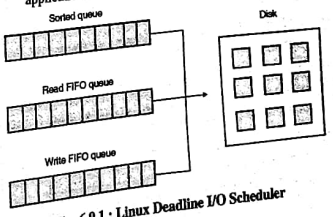


Fig. 6.9.1 : Linux Deadline I/O Scheduler

- Thus, a stream of write requests can block a read request for a substantial amount of time and therefore block a process.
- To deal with these problems, the deadline I/O scheduler makes use elevator queue, as before.
- Both read and write queues keep a list of requests in the sequence in which they were issued.
- Expiration time for read and write request is default. It is 0.5 seconds for read request and 5 seconds for write request. In general, the scheduler sends out request from the sorted queue.
- Once a request is satisfied, it is taken away from the head of the sorted queue and also from the appropriate FIFO queue.
- But, in case if the item at the head of one of the FIFO queues becomes older than its expiration time, then the scheduler after that send outs from that FIFO queue, taking the expired request together with the next few requests from the queue.
- As each request is sent out, it is also deleted from the sorted queue.
- The deadline I/O scheduler scheme removes the starvation problem and also the read versus write problem.

Anticipatory I/O Scheduler

- Sometime it is possible that consecutive reads from the same process will be to disk blocks that are close to one another.
- This is called principle of locality of reference.
- Generally performance of the system could be better improved if scheduler were to holdup a short period of time after fulfilling a read request, to see if a new close by read request is issued.
- This idea is behind anticipatory scheduler implemented in Linux 2.6.
- In Linux, the anticipatory scheduler is placed over the deadline scheduler.
- When a read request is dispatched, the anticipatory scheduler causes the scheduling system to delay for up to 6 ms, depending on the configuration.
- During this small delay, there is a good chance that the application that issued the last read request will issue another read request to the same region of the disk.
- If so, that request will be serviced without delay. If no such read request occurs, the scheduler resumes using the deadline scheduling algorithm.

6.9.1(B) Linux Page Cache

- In Linux 2.2 and former versions, for reads and writes from regular file system files and for virtual memory pages, kernel had maintained a page cache.
- A distinct buffer cache for block I/O was also maintained by kernel.
- On the contrary, in Linux 2.4 and in releases after it, a single joint page cache is introduced which is concerned in all traffic between disk and main memory.
- The page cache provides two benefits. Initial, when dirty pages are needed to write back to disk; a group of these pages can be ordered appropriately and written out efficiently.
- Second, because of the principle property of locality of reference, pages in the page cache are probable to be referenced in another time prior to they are flushed from the cache, thus saving a disk I/O operation.

Dirty pages are written back to disk in two situations :

- When free memory available goes below the specified threshold, the kernel decreases the size of the page cache to free memory to be added to the free memory pool.
- When dirty pages grow older than a specified threshold, a number of dirty pages are written back to disk.

6.10 Exam Pack (University and Review Questions)

Syllabus Topic : I/O Devices

Q. Explain different types of I/O devices. *(Refer section 6.1)*

Q. Explain different parameters to distinguish I/O devices. *(Refer section 6.1)*

Syllabus Topic : Organization of the I/O Function

Q. Explain different techniques to perform I/O. *(Refer section 6.2)*

Q. Explain evolutionary steps of I/O function. *(Refer section 6.2.1)*

Q. Explain steps in DMA transfer. *(Refer section 6.2.2)*

Syllabus Topic : Operating System Design Issues

Q. Explain operating systems design issue related to I/O. *(Refer section 6.3)*

Q. Explain logical structure of I/O function. *(Refer section 6.3.2)*

Syllabus Topic : I/O Buffering

Q. Explain various I/O buffering techniques. *(Refer section 6.4)* (10/5 Marks) **(Nov. 2015, Dec. 2016)**

Q. What is I/O buffering? Explain different types of I/O buffering. *(Refer section 6.4)*

Syllabus Topic : Disk Scheduling Algorithm

Q. Explain techniques of disk scheduling. *(Refer section 6.5)* (10 Marks) **(Nov. 2015)**

Syllabus Topic : Disk Scheduling Algorithm - FCFS

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. *(Refer section 6.5.1)* (10 Marks) **(Dec. 2016)**

Syllabus Topic : Disk Scheduling Algorithm - SSTF

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. *(Refer section 6.5.2)* (10 Marks) **(Dec. 2016)**

Q. Explain SSTF disk scheduling algorithm. *(Refer section 6.5.2)*

Syllabus Topic : Disk Scheduling Algorithm - SCAN

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. *(Refer section 6.5.3)* (10 Marks) **(Dec. 2016)**

Syllabus Topic : Disk Scheduling Algorithm - CSCAN

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. *(Refer section 6.5.4)* (10 Marks) **(Dec. 2016)**

Syllabus Topic : Disk Scheduling Algorithm : LOOK, C-LOOK

Q. Compare the following Disk scheduling algorithms using appropriate example-SSTF, FCFS, SCAN, C-SCAN, LOOK. *(Refer section 6.5.5)* (10 Marks) **(Dec. 2016)**

Q. Explain LOOK disk scheduling algorithms. *(Refer section 6.5.5)*

Example 6.6.1 (10 Marks) **(Dec. 2014)**

Example 6.6.10 (10 Marks) **(May 2016)**

Example 6.6.11 (10 Marks) **(June 2015)**

Example 6.6.12 (10 Marks) **(Dec. 2015)**

Syllabus Topic : Disk Management

Q. Explain disk formatting activity of disk management of operating system. *(Refer section 6.7.1)*

Q. Explain boot block activity of disk management of operating system. *(Refer section 6.7.2)*

Syllabus Topic : Disk Cache

Q. Explain disk cache. (Refer section 6.8) (5 Marks) **(Nov. 2015)**

Syllabus Topic : Linux I/O

Q. Explain Linux I/O. *(Refer section 6.9)*

Q. Explain elevator disk scheduling algorithm. *(Refer section 6.9.1(A))*

□□□

Lab Manual

L1. Explore the internal commands of linux like ls,chdir, mkdir, chown, chmod, chgrp, ps etc.

Ans. :

(a) ls : List information about the FILEs (the current directory by default)

```
root@rajesh-optiplex-3020:~$ ls
abc.cpp                 grpA10.cpp              pointfun.cpp
a.out                   grpA13.cpp              Public
cg2.cpp                 grpA2.cpp               saddle1.cpp
classtoclass.cpp        grpA5.cpp               saddle.cpp
client.py               grpC3.cpp               server.py
compose.cpp             hello.                  static.cpp
Desktop                 inode.py                synopsis.log
Documents               libgraph-1.0.2          synopsis.tex
Downloads               libgraph-1.0.2.tar.gz   Templates
examples.desktop        missfont.log            texput.log
f.cpp                   Music                   this1.cpp
filehandle1.cpp         oop                     this.cpp
filehandle.cpp          pointer.cpp             Videos
first.py                Pictures
root@rajesh-optiplex-3020:~$
```

(b) chdir : Displays the name of the current directory or changes the current folder.

```
cd /home/root

root@rajesh-optiplex-3020:~$ ls
abc.cpp                 grpA10.cpp              pointfun.cpp
a.out                   grpA13.cpp              Public
cg2.cpp                 grpA2.cpp               saddle1.cpp
classtoclass.cpp        grpA5.cpp               saddle.cpp
client.py               grpC3.cpp               server.py
compose.cpp             hello.                  static.cpp
Desktop                 inode.py                synopsis.log
Documents               libgraph-1.0.2          synopsis.tex
Downloads               libgraph-1.0.2.tar.gz   Templates
examples.desktop        missfont.log            texput.log
f.cpp                   Music                   this1.cpp
filehandle1.cpp         oop                     this.cpp
filehandle.cpp          opointer.cpp            Videos
first.py                Pictures
root@rajesh-optiplex-3020:~$
```

(c) mkdir - It create the DIRECTORY(ies), if they do not already exist.

```
root@rajesh-optiplex-3020:~$ mkdir rajesh
root@rajesh-optiplex-3020:~$ ls
abc.cpp                 grpA10.cpp              Pictures
a.out                   grpA13.cpp              pointfun.cpp
cg2.cpp                 grpA2.cpp               Public
classtoclass.cpp        grpA5.cpp               saddle1.cpp
client.py               grpC3.cpp               saddle.cpp
```

```
compose.cpp          hello.          server.py
Desktop              inode.py        synopsis.aux
Documents            rajesh          synopsis.log
Downloads            libgraph-1.0.2  synopsis.tex
examples.desktop     LINES.txt       test.cpp
f.cpp                missfont.log    texput.log
filehandle1.cpp      Music           this1.cpp
filehandle.cpp       oop             this.cpp
first.py             opointer.cpp    Videos
root@rajesh-optiplex-3020:~$
```

(d) chmod- It change file mode bits.

```
root@rajesh-optiplex-3020:~$ chmod 777 cg2.cpp
root@rajesh-optiplex-3020:~$
```

(e) ps – It report a snapshot of the current processes.

```
root@rajesh-optiplex-3020:~$ ps
  PID TTY          TIME CMD
 5366 pts/5    00:00:00 bash
 5618 pts/5    00:00:00 ps
root@rajesh-optiplex-3020:~$
```

(f) chown command is used to change the owner and group of files, directories and links.

By default, the owner of a file system object is the user that created it. The group is a set of users that share the same access permissions (i.e., read, write and execute) for that object.

The basic syntax for using chown to change owners is

```
chown [options] new_owner object(s)
```

new_owner is the user name or the numeric user ID (UID) of the new owner, and object is the name of the target file, directory or link. The ownership of any number of objects can be changed simultaneously.

For example, the following would transfer the ownership of a file named file1 and a directory dir1 to a new owner named alice:

```
chown alice file1 dir1
```

(g) chgrp command is used to change the group ownership of the file

For Example

```
chgrp abc xyz.txt
```

changes the owning group of the file abc.txt to the group xyz.

L2. Write shell scripts to do the following :

1. **Display top 10 processes in descending order**
2. **Display processes with highest memory usage**
3. **Display current logged user and login name**
4. **Display current shell, home directory, operating system type, current path setting, current working directory**
5. **Display OS name, release number, kernel version**
6. **Illustrate the use of sort, grep, awk, etc.**

Ans. :

(1) Display top 10 processes in descending order

Program

```
echo " Top 10 processes in decending order is:";
ps axl | head -n 10
```

Output

```
root@rajesh-optiplex-3020:~$ sh exp1.sh
Top 10 processes in decending order is:
F  UID   PID  PPID  PRI  NI    VSZ    RSS  WCHAN  STAT TTY      TIME COMMAND
4    0     1     0   20   0 121712   5692      -  Ss   ?      0:01 /sbin/init splash
1    0     2     0   20   0      0      0      -  S    ?      0:00 [kthreadd]
1    0     3     2   20   0      0      0      -  S    ?      0:00 [ksoftirqd/0]
1    0     5     2  0-20   0      0      -      S  <    ?      0:00 [kworker/0:0H]
1    0     7     2   20   0      0      0      -  S    ?      0:44 [rcu_sched]
1    0     8     2   20   0      0      0      -  S    ?      0:00 [rcu_bh]
1    0     9     2 -100   -      0      0      -  S    ?      0:00 [migration/0]
1    0    10     2 -100   -      0      0      -  S    ?      0:00 [watchdog/0]
5    0    11     2 -100   -      0      0      -  S    ?      0:00 [watchdog/1]
```

(2) Display processes with highest memory usage

Program

```
echo " Display processes with highest memory usage";
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
```

Output

```
root@rajesh-optiplex-3020:~$ gedit exp2.sh
root@rajesh-optiplex-3020:~$ sh exp2.sh
Display processes with highest memory usage
  PID  PPID CMD                         %MEM %CPU
 2227  1581 /usr/lib/firefox/firefox    15.9  3.3
 5077  5059 /usr/lib/libreoffice/progra  5.4  2.0
 2222  1581 /usr/lib/x86_64-linux-gnu/z  4.3  0.0
 2041  1581 compiz                       4.0  0.5
 2369  1581 /usr/bin/python3 /usr/bin/u  3.1  0.0
  980   872 /usr/lib/xorg/Xorg -core :0  2.6  0.7
 1973  1816 /usr/bin/gnome-software --g  2.5  0.0
 1989  1816 nautilus -n                   1.7  0.1
 2014  1581 /usr/lib/evolution/evolutio  1.5  0.0
root@rajesh-optiplex-3020:~$
```

(3) Display current logged user and login name

Program

```
echo "LOGGED IN USERS ARE:-\n";
who -u
echo "NUMBER OF LOGGED IN USERS ARE :-\n";
who -u | wc -l
```

Output

```
root@rajesh-optiplex-3020:~$ sh exp3.sh
LOGGED IN USERS ARE:-
```

```
root    tty7         2017-09-09 09:57 04:21      1581 (:0)
NUMBER OF LOGGED IN USERS ARE :-1
root@rajesh-optiplex-3020:~$
```

(4) Display current shell, home directory, operating system type, current path setting, current working directory

Program

```
echo " Current home directory is:";
whoami
echo " Current  operating system type is:";
uname
echo " Current  current working directory is:";
pwd
```

Output

```
root@rajesh-optiplex-3020:~$ sh exp4.sh
 Current home directory is:
root
 Current  operating system type is:
Linux
 Current  current working directory is:
/home/root
root@rajesh-optiplex-3020:~$
```

(5) Display OS name, release number, kernel version

Program

```
echo " OS Name is:"
uname;
echo " Release number is:"
uname -a;
echo " Release number is:"
uname -r;
```

Output

```
root@rajesh-optiplex-3020:~$ sh exp5.sh
 OS Name is:
Linux
Release number is:
Linux rajesh-optiplex-3020 4.4.0-81-generic #104-Ubuntu SMP Wed Jun 14 08:17:06 UTC 2017 x86_64 x86_64 x86_64
GNU/Linux
 Release number is:
4.4.0-81-generic
root@rajesh-optiplex-3020:~
```

(6) Use of sort, grep, awk, etc.

Sort -sorts the contents of a text file, line by line

```
sort syntax
sort [OPTION]... [FILE]...

sort [OPTION]... --files0-from=F
Let's say you have a file, data.txt, which contains the following ASCII text:

apples
```



```
oranges
pears
Kiwis
bananas
```

To sort the lines in this file alphabetically, use the following command :

```
sort data.txt
```

which will produce the following output :

```
apples
bananas
Kiwis
oranges
pears
```

Note that this command does not actually change the input file, **data.txt**. If you want to write the output to a new file, **output.txt**, redirect the output like this :

```
sort data.txt > output.txt
```

Which will not display any output, but will create the file **output.txt** with the same sorted data from the previous command. To check the output, use the **cat** command :

```
cat output.txt
```

grep

grep- print lines matching a pattern , **grep** searches the named input *FILE*s (or standard input if no files are named, or if a single hyphen-minus (-) is given as file name) for lines containing a match to the given *PATTERN*.

Use grep to search words only

When you search for boo, grep will match fooboo, boo123, barfoo35 and more. You can force the grep command to select only those lines containing matches that form whole words i.e. match only boo word :

```
$ grep -w "boo" file
```

Use grep to search 2 different words

```
Use the egrep command as follows:
$ egrep -w 'word1|word2' /path/to/file
```

Awk

AWK is an interpreted programming language. It is very powerful and specially designed for text processing.

Consider a text file **marks.txt** to be processed with the following content

```
1) Root   Physics  80
2) Rahul  Maths    90
3) Shyam  Biology  87
4) Kedar  English  85
5) Hari   History  89
```

Printing Column or field

You can instruct AWK to print only certain columns from the input field. The following example demonstrates this

```
[rajesh]$ awk '{print $3 "\t" $4}' marks.txt

Physics  80
Maths    90
Biology  87
English  85
History  89
```

L3. (a) Create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system call.

(b) Explore the following system calls : open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid

Ans. :

(a) Create child process in Linux using fork system call

Program

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int pid;

pid=fork();

if(pid==0)
{
printf("\n After fork");
printf("\n The new child process created by fork system call %d\n", getpid());
}
else {
printf("\n Before fork");

printf("\n The parent process id is :- %d ", getppid());
printf("\n parent excuted successfully\n");

}

return 0;

}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc B3.c
root@rajesh-optiplex-3020:~$ ./a.out

Before fork
The parent process id is :- 4070
parent excuted successfully

After fork
The new child process created by fork system call 4428
root@rajesh-optiplex-3020:~$ ^C
root@rajesh-optiplex-3020:~$
```

(b) System calls : open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid

Open

open- open and possibly create a file

open() return the new file descriptor, or -1 if an error occurred (in which case, errno is set appropriately).

Read

Read- read from a file descriptor.

If count is zero, **read**() returns zero and has no other results

Write

write- write to a file descriptor

On success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned, and errno is set appropriately.

Close

Close - close a file descriptor

close() closes a file descriptor, so that it no longer refers to any file and may be reused, **close**() returns zero on success. On error, -1 is returned, and errno is set appropriately.

getpid

Getpid - get process identification

getpid() returns the process ID of the current process.

Setpid

Setpid- **setpgid**() sets the process group ID of the process specified by pid to pgid.

Getpid

Getuid- get user identity.

getuid() returns the real user ID of the current process

geteuid

geteuid - geteuid() returns the effective user ID of the current process.

getgid

Getgid - get group identity

getgid() returns the real group ID of the current process.

L4. Implement basic commands of linux like ls, cp, mv and others using kernel APIs

Ans. :

(a) Implement "cp" using system calls

1. First check if both source & the destination files are received from the command line argument and exit if argc counter is not equal to 3.There is also a check to handle "–help" option to print the usage of cpcmd.
2. Open the source file with read only flag set.
3. Open the destination file with the respective flags & modes. O_WRONLY -> Open the file in write only mode O_TRUNC -> Truncates the contents of the existing file O_CREAT -> Creates a new file if it doesn't exist S_IXUSR are file permissions for the current user ('X' can be R for read & W for write) S_IXGRP are file permissions for the groups ('X' can be R for read & W for write) S_IXOTH are file permissions for the groups ('X' can be R for read & W for write)
4. Start data transfer from source file to destination file till it reaches EOF (nbread == 0).

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>

#define BUFF_SIZE 1024
```

```
int main(int argc, char* argv[])
{
   int srcFD,destFD,nbread,nbwrite;
   char *buff[BUFF_SIZE];

   /*Check if both src & dest files are received or --help is received to get usage*/
   if(argc != 3 || argv[1] == "--help")
   {
         printf("\nUsage: cpcmd source_file destination_file\n");
         exit(EXIT_FAILURE);
   }

   /*Open source file*/
   srcFD = open(argv[1],O_RDONLY);

   if(srcFD == -1)
   {
         printf("\nError opening file %s errno = %d\n",argv[1],errno);
         exit(EXIT_FAILURE);
   }

   /*Open destination file with respective flags & modes          O_CREAT & O_TRUNC is to truncate existing file or create
a new file       S_IXXXX are file permissions for the user,groups & others*/

destFD = open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |
S_IROTH | S_IWOTH);

   if(destFD == -1)
   {
         printf("\nError opening file %s errno = %d\n",argv[2],errno);
         exit(EXIT_FAILURE);
   }

   /*Start data transfer from src file to dest file till it reaches EOF*/
   while((nbread = read(srcFD,buff,BUFF_SIZE)) > 0)
   {
         if(write(destFD,buff,nbread) != nbread)
               printf("\nError in writing data to %s\n",argv[2]);
   }

   if(nbread == -1)
         printf("\nError in reading data from %s\n",argv[1]);

   if(close(srcFD) == -1)
         printf("\nError in closing file %s\n",argv[1]);

   if(close(destFD) == -1)
         printf("\nError in closing file %s\n",argv[2]);

   exit(EXIT_SUCCESS);
}
```


Output

```
./cpcmd src.txt dest.txt

Error opening file src.txt errno = 2

touch src.txt

echo "This program is a simulation of the Linux copy command "cp" using I/O system calls. The copy command can be used to
make a copy of your files and directories, but here in this implementation only the basic functionality i.e., copy the content
from one file to another is handled." > src.txt

./cpcmd src.txt dest.txt

cat dest.txt
This program is a simulation of the Linux copy command cp using I/O system calls. The copy command can be used to make a
copy of your files and directories, but here in this implementation only the basic functionality i.e., copy the content from one
file to another is handled.
```

(b) Implement "ls" command using system calls

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<dirent.h>
int main(int argc,char *argv[])
{
DIR *dp;
struct dirent *sd;
dp=opendir(argv[1]);
while((sd=readdir(dp))!=NULL)
{
printf("%s\t",sd->d_name);
}
closedir(dp);
}
Output:
root@litmus:/home# gcc lscmd.c
root@litmus:/home#./a.out dirname
.  ..  f.txt f1.c
```

(c) Implement "mv" command using system calls

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<dirent.h>
int main(int argc,char *argv[])
{
int i, fd1,fd2;
char *file1,*file2,buf[2];
file1=argv[1];
file2=argv[2];
printf("file1=%s file2=%s", file1,file2);
fd1=open(file1,O_RDONLY,0777);
fd2=creat(file2,0777);
```

```
while(i=read(fd1,buf,1)>0)
write(fd2,buf,1);
remove(file1);
close(fd1);
close(fd2);
}
```

Output:

```
root@rdk:$gcc mv.c -o mv.out
root@rdk:$cat > f1
hello world
root@rdk:$ ./mv.out f1 f2
root@rdk:$cat f1
cat:f1 No such file or directory
root@rdk:$cat f2
hello world
```

L5. Write a program to implement any two CPU scheduling algorithms like FCFS, SJF, Round Robin etc.

Ans. :

(a) FCFS

Program

```
#include<stdio.h>

int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }

   wt[0]=0;   //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
   }

   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

   //calculating turnaround time
   for(i=0;i<n;i++)
   {
```



```
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
      printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
   }

   avwt/=i;
   avtat/=i;
   printf("\n\nAverage Waiting Time:%d",avwt);
   printf("\nAverage Turnaround Time:%d",avtat);

   return 0;
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc fcfs.c
root@rajesh-optiplex-3020:~$ ./a.out
Enter total number of processes(maximum 20):5

Enter Process Burst Time
P[1]:10
P[2]:5
P[3]:6
P[4]:8
P[5]:7

Process     Burst Time    Waiting Time    Turnaround Time
P[1]        10            0               10
P[2]        5             10              15
P[3]        6             15              21
P[4]        8             21              29
P[5]        7             29              36

Average Waiting Time:15
Average Turnaround Time:22
root@rajesh-optiplex-3020:~$
```

(b) SJF

Program

```
#include<stdio.h>

void main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
```

```
    scanf("%d",&bt[i]);
    p[i]=i+1;          //contains process number
  }

  //sorting burst time in ascending order using selection sort
  for(i=0;i<n;i++)
  {
    pos=i;
    for(j=i+1;j<n;j++)
    {
      if(bt[j]<bt[pos])
        pos=j;
    }

    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;

    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
  }

  wt[0]=0;          //waiting time for first process will be zero

  //calculate waiting time
  for(i=1;i<n;i++)
  {
    wt[i]=0;
    for(j=0;j<i;j++)
      wt[i]+=bt[j];

    total+=wt[i];
  }

  avg_wt=(float)total/n;     //average waiting time
  total=0;

  printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
  for(i=0;i<n;i++)
  {
    tat[i]=bt[i]+wt[i];     //calculate turnaround time
    total+=tat[i];
    printf("\np%d\t\t  %d\t\t    %d\t\t%d",p[i],bt[i],wt[i],tat[i]);
  }

  avg_tat=(float)total/n;     //average turnaround time
  printf("\n\nAverage Waiting Time=%f",avg_wt);
  printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```


Output

```
root@rajesh-optiplex-3020:~$ gcc sjf.c
root@rajesh-optiplex-3020:~$ ./a.out
Enter number of process:5

Enter Burst Time:
p1:10
p2:5
p3:6
p4:4
p5:12

Process     Burst Time      Waiting Time    Turnaround Time
p4          4               0               4
p2          5               4               9
p3          6               9               15
p1          10              15              25
p5          12              25              37

Average Waiting Time=10.600000
Average Turnaround Time=18.000000
root@rajesh-optiplex-3020:~$
```

(c) Round Robin

```
void RR()
 {
  struct Process Proc_arr[20];
  float avg_time=0,turn_around_time=0,t=0;
  int time,maxproc,i,j,k,time_qtm;
  char str[10];

  printf("\n\t#ENTER HOW MANY PROCESSES ARE THERE :- ");
  scanf("%d",&maxproc);
  for(i=1;i<=maxproc;i++)
  {
   printf("\n\t#ENTER THE BRUST TIME FOR THE PROCESS P%d :- ",i);
   scanf("%d",&Proc_arr[i].bst_time);
   turn_around_time+=Proc_arr[i].bst_time;
   Proc_arr[i].id=i;
   Proc_arr[i].wt_time=0;
  }

  printf("\n#ENTER THE TIME QUANTUM FOR THE PEOCESSES :- ");
  scanf("%d",&time_qtm);
  system("clear");

  i=0;
  k=0;
  time=0;

  printf("\n# TURN AROUND TIME IS = %.2f ",turn_around_time);
```

```
while(i<maxproc)
{
 i++;
 if(Proc_arr[i].bst_time>0)
 {

  if(Proc_arr[i].bst_time<time_qtm)
   time+=Proc_arr[i].bst_time;
  else
   time+=time_qtm;

  for(j=1;j<=maxproc;j++)
  {
  if(Proc_arr[j].bst_time>0 && j!=i)
   {
    if(Proc_arr[j].bst_time<time_qtm)
        Proc_arr[j].wt_time+=Proc_arr[i].bst_time;
    else
        Proc_arr[j].wt_time+=time_qtm;
   }
  }//for

  Proc_arr[i].bst_time-=time_qtm;

  if(Proc_arr[i].bst_time<=0)
    k++;
 }//if

 if(i==maxproc)
   i=0;
}//while

for(j=1;j<=maxproc;j++)
  printf("\n Waiting time[%d] = %d",j,Proc_arr[j].wt_time);

printf("\n\n\n\n\n\tPROCESS\t\tWAITING TIME\n\n");
avg_time=0;
for(i=1;i<=maxproc;i++)
{
 printf("\tP%d\t\t\t",Proc_arr[i].id);
 printf("%d\n\n",Proc_arr[i].wt_time);
 turn_around_time+=Proc_arr[i].wt_time;
 avg_time+=Proc_arr[i].wt_time;
}
avg_time=avg_time/maxproc;

printf("\n# AVERAGE WAITING TIME IS = %.2f ",avg_time);
printf("\n# AVERAGE TURN AROUND TIME IS = %.2f ",turn_around_time/maxproc);

}//RR
```

L6. Write a program to implement dynamic partitioning placement algorithms i.e Best Fit, First-Fit, Worst-Fit etc.

Ans. :

(a) Best Fit

Program

```
#include<stdio.h>

void main()
{
  int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
  static int barray[20],parray[20];

  printf("\n\t\t\tMemory Management Scheme - Best Fit");
  printf("\nEnter the number of blocks:");
  scanf("%d",&nb);
  printf("Enter the number of processes:");
  scanf("%d",&np);

  printf("\nEnter the size of the blocks:-\n");
  for(i=1;i<=nb;i++)
  {
    printf("Block no.%d:",i);
    scanf("%d",&b[i]);
  }

  printf("\nEnter the size of the processes :-\n");
  for(i=1;i<=np;i++)
  {
    printf("Process no.%d:",i);
    scanf("%d",&p[i]);root@rajesh-optiplex-3020:~$ gcc bestfit.c
  }

  for(i=1;i<=np;i++)
  {
    for(j=1;j<=nb;j++)
    {
      if(barray[j]!=1)
      {
        temp=b[j]-p[i];
        if(temp>=0)
          if(lowest>temp)
          {
            parray[i]=j;
            lowest=temp;
          }
      }
    }

    fragment[i]=lowest;
    barray[parray[i]]=1;
    lowest=10000;
  }
```

```
  printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
  for(i=1;i<=np && parray[i]!=0;i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc bestfit.c
root@rajesh-optiplex-3020:~$ ./a.out

                Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:10
Block no.2:15
Block no.3:42
Block no.4:12
Block no.5:12

Enter the size of the processes :-
Process no.1:12
Process no.2:45
Process no.3:12
Process no.4:10

Process_no    Process_size  Block_no     Block_size Fragment
1             12            4            12         0
root@rajesh-optiplex-3020:~$
```

(b) First-Fit

Program

```
#include<stdio.h>

void main()
{
  int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

  for(i = 0; i < 10; i++)
  {
    flags[i] = 0;
    allocation[i] = -1;
  }

  printf("Enter no. of blocks: ");
  scanf("%d", &bno);

  printf("\nEnter size of each block: ");
  for(i = 0; i < bno; i++)
    scanf("%d", &bsize[i]);

  printf("\nEnter no. of processes: ");
```



```
  scanf("%d", &pno);

  printf("\nEnter size of each process: ");
  for(i = 0; i < pno; i++)
    scanf("%d", &psize[i]);
  for(i = 0; i < pno; i++)        //allocation as per first fit
    for(j = 0; j < bno; j++)
      if(flags[j] == 0 && bsize[j] >= psize[i])
      {
        allocation[j] = i;
        flags[j] = 1;
        break;
      }

  //display allocation details
  printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
  for(i = 0; i < bno; i++)
  {
    printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
    if(flags[i] == 1)
      printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
    else
      printf("Not allocated");
  }
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc first.c
root@rajesh-optiplex-3020:~$ ./a.out
Enter no. of blocks: 5

Enter size of each block: 10
15
42
45
10

Enter no. of processes: 4

Enter size of each process: 20
45
12
41

Block no.  size        process no.     size
1          10          Not allocated
2          15          3               12
3          42          1               20
4          45          2               45
5          10          Not allocated

root@rajesh-optiplex-3020:~$
```

(c) Worst Fit

Program

```
#include<stdio.h>

int main()
{
    int fragments[10], blocks[10], files[10];
    int m, n, number_of_blocks, number_of_files, temp, top = 0;
    static int block_arr[10], file_arr[10];
    printf("\nEnter the Total Number of Blocks:\t");
    scanf("%d",&number_of_blocks);
    printf("Enter the Total Number of Files:\t");
    scanf("%d",&number_of_files);
    printf("\nEnter the Size of the Blocks:\n");
    for(m = 0; m < number_of_blocks; m++)
    {
        printf("Block No.[%d]:\t", m + 1);
        scanf("%d", &blocks[m]);
    }
    printf("Enter the Size of the Files:\n");
    for(m = 0; m < number_of_files; m++)
    {
        printf("File No.[%d]:\t", m + 1);
        scanf("%d", &files[m]);
    }
    for(m = 0; m < number_of_files; m++)
    {
        for(n = 0; n < number_of_blocks; n++)
        {
            if(block_arr[n] != 1)
            {
                temp = blocks[n] - files[m];
                if(temp >= 0)
                {
                    if(top < temp)
                    {
                        file_arr[m] = n;
                        top = temp;
                    }
                }
            }
            fragments[m] = top;
            block_arr[file_arr[m]] = 1;
            top = 0;
        }
    }
    printf("\nFile Number\tFile Size\tBlock Number\tBlock Size\tFragment");
    for(m = 0; m < number_of_files; m++)
    {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], file_arr[m], blocks[file_arr[m]], fragments[m]);
    }
```



```
    printf("\n");
    return 0;
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc worst.c
root@rajesh-optiplex-3020:~$ ./a.out

Enter the Total Number of Blocks:    5
Enter the Total Number of Files: 5

Enter the Size of the Blocks:
Block No.[1]: 12
Block No.[2]: 12
Block No.[3]: 15
Block No.[4]: 42
Block No.[5]: 45
Enter the Size of the Files:
File No.[1]:    10
File No.[2]:    15
File No.[3]:    42
File No.[4]:    41
File No.[5]:    05

File Number   File Size       Block Number      Block Size    Fragment
0             10              4                 45            35
1             15              0                 12            0
2             42              0                 12            0
3             41              0                 12            0
4             5               0                 12            0
root@rajesh-optiplex-3020:~
```

L7. Write a program to implement various page replacement policies.

Ans. :

(a) FIFO

Program

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
```

```
        {
                printf("%d\t\t",a[i]);
                avail=0;
                for(k=0;k<no;k++)
if(frame[k]==a[i])
                        avail=1;
                if (avail==0)
                {
                        frame[j]=a[i];
                        j=(j+1)%no;
                        count++;
                        for(k=0;k<no;k++)
                        printf("%d\t",frame[k]);
}
                printf("\n");
}
        printf("Page Fault Is %d",count);
        return 0;
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc fifo.c
root@rajesh-optiplex-3020:~$ ./a.out

 ENTER THE NUMBER OF PAGES:
10

 ENTER THE PAGE NUMBER :
7 0 1 2 0 3 0 4 2 3

 ENTER THE NUMBER OF FRAMES :3
ref string  page frames
7       7   -1  -1
0       7   0   -1
1       7   0   1
2       2   0   1
0
3       2   3   1
0       2   3   0
4       4   3   0
2       4   2   0
3       4   2   3
```

(b) Optimal

Program

```
#include<stdio.h>

int main()
{
  int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
  printf("Enter number of frames: ");
  scanf("%d", &no_of_frames);
```



```
  printf("Enter number of pages: ");
  scanf("%d", &no_of_pages);

  printf("Enter page reference string: ");

  for(i = 0; i < no_of_pages; ++i){
    scanf("%d", &pages[i]);
  }

  for(i = 0; i < no_of_frames; ++i){
    frames[i] = -1;
  }

  for(i = 0; i < no_of_pages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j){
      if(frames[j] == pages[i]){
        flag1 = flag2 = 1;
        break;
      }
    }

    if(flag1 == 0){
      for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == -1){
          faults++;
          frames[j] = pages[i];
          flag2 = 1;
          break;
        }
      }
    }

    if(flag2 == 0){
      flag3 =0;

      for(j = 0; j < no_of_frames; ++j){
        temp[j] = -1;

        for(k = i + 1; k < no_of_pages; ++k){
          if(frames[j] == pages[k]){
            temp[j] = k;
            break;
          }
        }
      }

      for(j = 0; j < no_of_frames; ++j){
        if(temp[j] == -1){
          pos = j;
```

```
            flag3 = 1;
            break;
          }
        }

        if(flag3 ==0){
          max = temp[0];
          pos = 0;

          for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
              max = temp[j];
              pos = j;
            }
          }
        }

        frames[pos] = pages[i];
        faults++;
      }

      printf("\n");

      for(j = 0; j < no_of_frames; ++j){
        printf("%d\t", frames[j]);
      }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc optimal.c
root@rajesh-optiplex-3020:~$ ./a.out
Enter number of frames: 3
Enter number of pages: 10
Enter page reference string: 7 0 1 2 0 3 0 4 2 3

7   -1    -1
7   0     -1
7   0     1
2   0     1
2   0     1
2   0     3
2   0     3
2   4     3
2   4     3
2   4     3
Total Page Faults = 6
root@rajesh-optiplex-3020:~$
```


(c) LRU

Program

```
#include<stdio.h>
void main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
        scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
                                k++;
                                for(j=0;j<k;j++)
                                printf("\t%d",q[j]);
                                printf("\n");
                        }
                        else
                        {
                                for(r=0;r<f;r++)
                                {
                                        c2[r]=0;
                                        for(j=i-1;j<n;j--)
                                        {
                                        if(q[r]!=p[j])
                                        c2[r]++;
                                        else
                                        break;
                                        }
                                }
                                for(r=0;r<f;r++)
```

```
                b[r]=c2[r];
                for(r=0;r<f;r++)
                {
                        for(j=r;j<f;j++)
                        {
                                if(b[r]<b[j])
                                {
                                        t=b[r];
                                        b[r]=b[j];
                                        b[j]=t;
                                }
                        }
                }
                for(r=0;r<f;r++)
                {
                        if(c2[r]==b[0])
                        q[r]=p[i];
                        printf("\t%d",q[r]);
                }
                printf("\n");
        }
    }
}
printf("\nThe no of page faults is %d",c);
}
```

Output

```
root@rajesh-optiplex-3020:~$ gcc lru.c
root@rajesh-optiplex-3020:~$ ./a.out
Enter no of pages:10
Enter the reference string:7
0
1
2
0
3
0
4
2
3
Enter no of frames:3

   7
   7    1
   7    1    2
   0    1    2
   0    3    2
   0    3    4
   0    2    4
   3    2    4
The no of page faults is 8
root@rajesh-optiplex-3020:~$
```

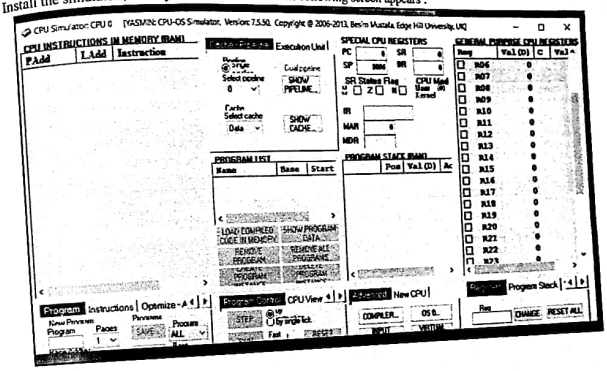

LS. Using the CPU-OS simulator analyse and synthesize the following: a. Process Scheduling algorithms. b. Thread creation and synchronization. c. Deadlock prevention and avoidance.

Ans. :

1. Download the YASS simulator from the link:

http://www.teach-sim.com

Install the simulator , when you run the simulator the following screen appears :



(a) Process scheduling

Follow the steps as given in the link //www.teachsim.com/Operating systems-Investigating process scheduling.

Step 1 : Entering source code in the compiler and compiling it to an executable program.

(a) You need to create some executable code so that it can be run by the CPU simulator under the control of the OS simulator. In order to create this code, you need to use the compiler which is part of the system simulator. This compiler is able to compile simple high-level source statements similar to Visual Basic. To do this, open the compiler window by selecting the COMPILER... button in the current window. You should now be looking at the compiler window.

(b) In the compiler window, enter the following source code in the compiler's source editor window (under PROGRAM SOURCE frame title) as shown in the screen shot :

```
program LoopTest  i = 0  for n = 0 to 40  i = i + 1  next end
```

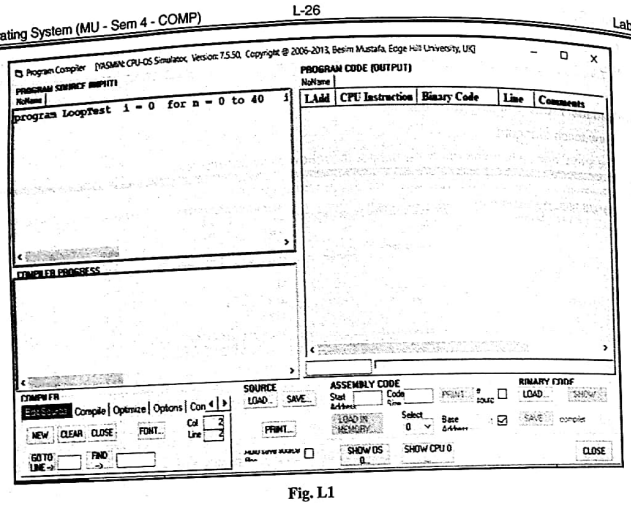


Fig. L1

(a) Now you need to compile this in order to generate the executable code. To do this, click on the COMPILE... button. You should see the code created on the right in PROGRAM CODE view. Make a habit of saving your source code. The code is shown below.

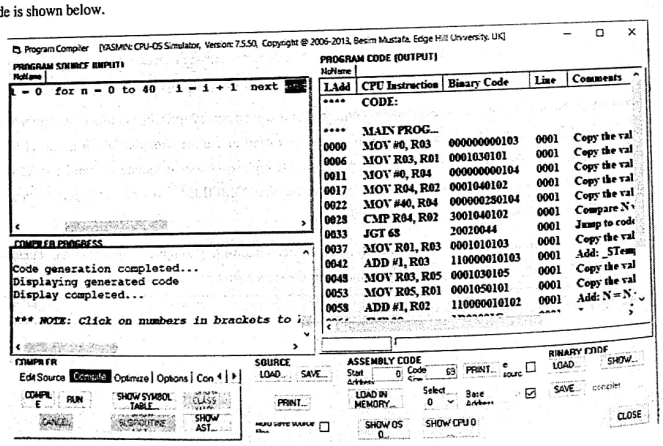


Fig. L2

(b) Click on the button SHOW... in BINARY CODE view as indicated by an arrow in the above figure. You should now see the Binary Code for LOOPTEST window.

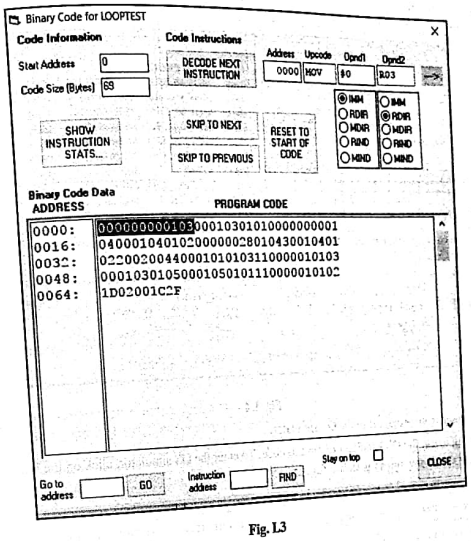


Fig. L3

Step 2 : Loading the program into the CPU simulator's memory.

Now, this code needs to be loaded in memory so that the CPU can execute it. To do this, first we need to specify a base address (in ASSEMBLY CODE view): uncheck the box next to the edit box with label Base Address, and then enter 100 in the edit box. Now, click on the LOAD IN MEMORY... button in the current window. You should now see the code loaded in memory ready to be executed. You are also back in the CPU simulator at this stage. This action is equivalent to loading the program code normally stored on a disc drive into RAM on the real computer systems.

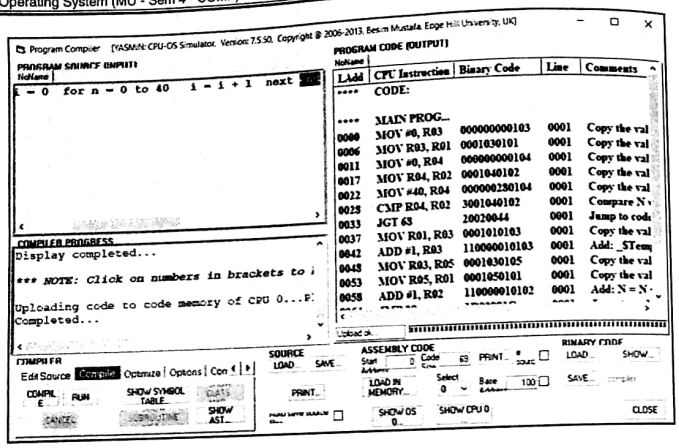


Fig. L4

Step 3 : Create processes from programs in the OS simulator.

We are now going to use the OS simulator to run this code. To enter the OS simulator, click on the OS 0… button in the current window. The OS window opens as shown Fig. L5.

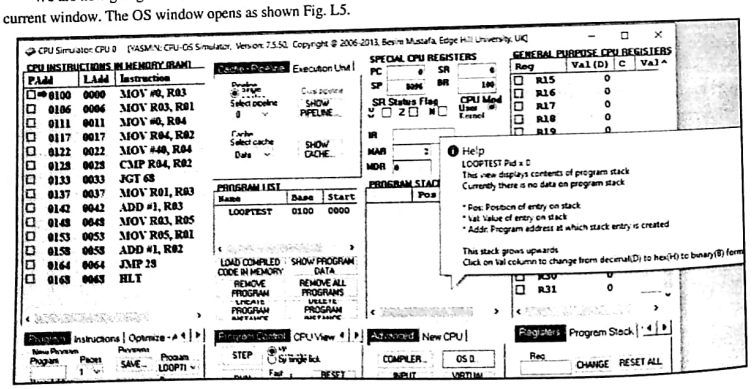


Fig. L5

You should see an entry, titled LoopTest, in the PROGRAM LIST view. Now that this program is available to the OS simulator, we can create as many instances, i.e. processes, of it as we like. You do this by clicking on the CREATE NEW PROCESS button. Repeat this four times. Observe the four instances of the program being queued in the ready queue which is represented by the READY PROCESSES view.

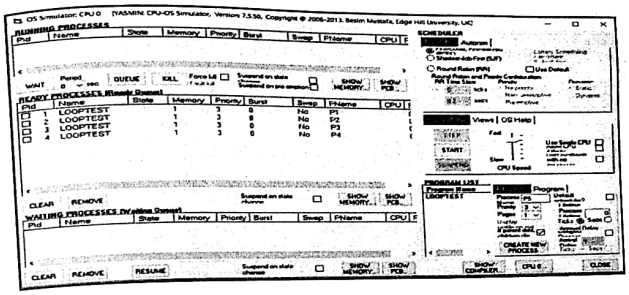


Fig. L6

Now, it is very important that you follow the instructions below without any deviation. If you do, then you must re-do the exercise from the beginning as any follow-up action(s) may give the wrong results.

Step 4 : Select different scheduling policies and run the processes in the OS simulator.

Make sure the First-Come-First-Served (FCFS) option is selected in the SCHEDULER/Policies view. At this point the OS is inactive. To activate, then click on the START button. This should start the OS simulator running the processes. Observe the instructions executing in the CPU simulator window. Concentrate on the two views: RUNNING PROCESSES and the READY PROCESSES during this period. You can view the process list as shown in Fig. L7.

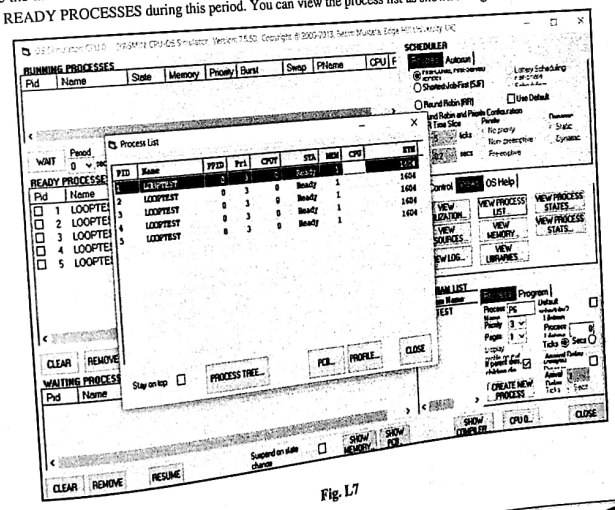


Fig. L7

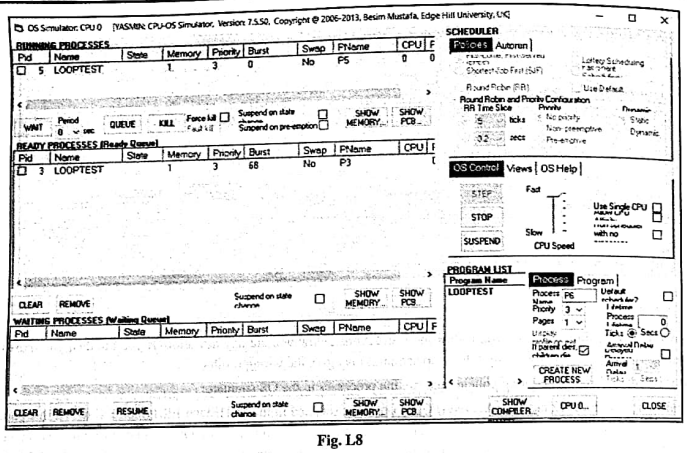


Fig. L8

On the same lines select other scheduler policies and check for the output. The activity log can be viewed from views tab. The activity log for RR scheduling with three processes is as shown in Fig. L9.

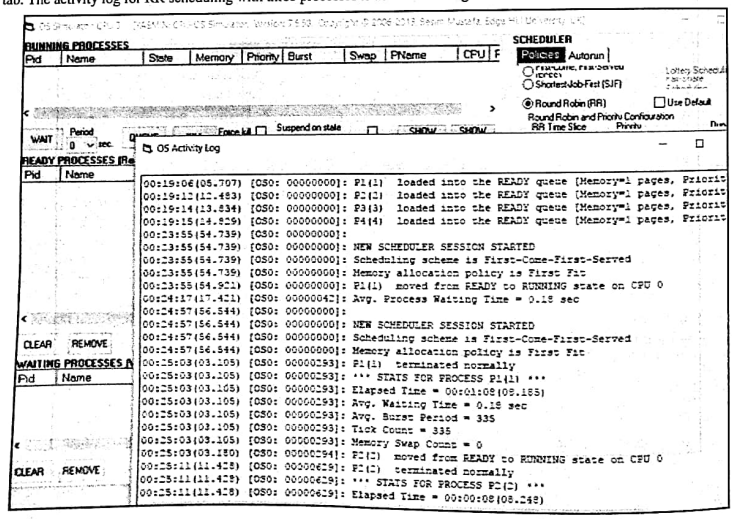


Fig. L9

(b) Investigating the use of Threads

Follow the steps as in link //www.teach-sim.com/Operating systems-Investigating threads and synchronization.

Step 1 : Write a source code that creates threads.

Start the CPU/OS simulator. In the compiler window enter the following source code :

```
program ThreadTest1
sub thread1 as thread
  writeln("In thread1")
while true
wend
end sub

sub thread2 as thread
  call thread1
       writeln("In thread2")
                while true
    wend
end sub

       call thread2
  writeln("In main")

  do loop
  end
```

Step2 : Compile the source code to get the assembly code.

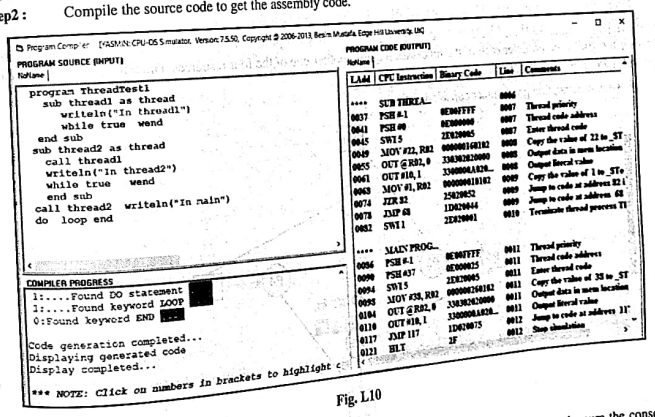


Fig. L10

Step 3 : Load the code in the memory using load to memory.

Step 4 : Make the console window visible by clicking on the INPUT/OUTPUT...button. Also make sure the console window stays on top by checking the Stay on top check box.

Step 5 : Now, go to the OS simulator window (use the OS... button in the CPU simulator window) and create a single process of program ThreadTest1 in the program list view. For this use the CREATE NEW PROCESS button.

Make sure the scheduling policy selected is Round Robin and that the simulation speed is set at maximum. Hit the START button and at the same time observe the displays on the console window as shown in Fig. L11.

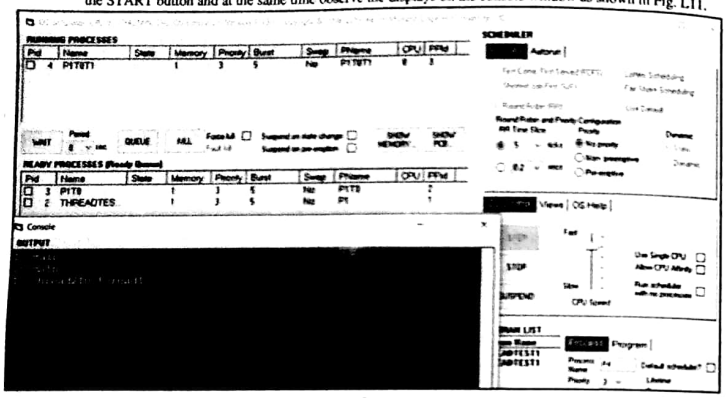


Fig. L11

(c) Deadlocks Prevention

Four processes are running. They are called P1 to P4. There are also four resources available (only one instance of each). They are named R0 to R3. At some point of their existence each process allocates a different resource for use and holds it for itself forever. Later each of the processes request another one of the four resources. Draw the resource allocation graph for a four process deadlock condition.

Resource Allocation graph under consideration.

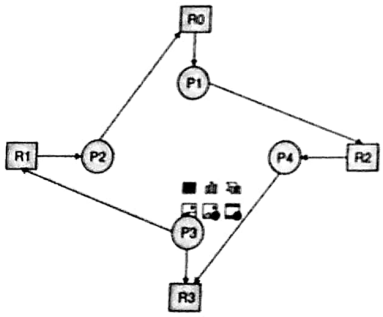


Fig. L12

Step 1 : In the compiler window, enter the following source code in the compiler source editor area (under PROGRAM SOURCE frame title).


```
program DeadlockPN
resource(X, allocate)
wait(3)
resource(Y, allocate)
for n = 1 to 20
next
end
```

The above code creates a program which attempts to allocate two resources for itself. After the first allocation it waits for 3 seconds and tries to allocate another resource. Finally it counts from 1 to 20 in a loop and then terminates. Other than that it does nothing sensible or useful.

Step 2 :

a. Copy the above code and paste it in three more edit windows so that you have a total of four pieces of source code.
b. In each case change N in the program name to 1 to 4, e.g. DeadlockP1, DeadlockP2, etc.
c. Look at your graph you constructed in (1) above and using that information fill in the values for each of the Xs and Ys in the four pieces of source code.
d. Compile each one of the four source code.
e. Load in memory the four pieces of code generated.

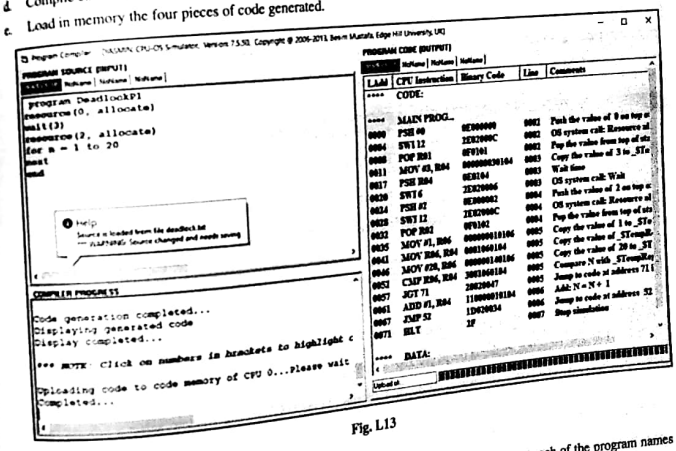


Fig. L13

f. Now switch to the OS simulator.
g. Create a single instance of each of the programs. You can do this by double-clicking on each of the program names in the PROGRAM LIST frame under the Program Name column.
h. In the SCHEDULER frame select Round Robin (RR) scheduling policy in the Policies tab.
i. In OS Control tab, push the speed slider up to the fastest speed.
j. Select the Views tab and click on the VIEW RESOURCES... button.
k. Select Stay on top check box in the displayed window.
l. Back in the OS Control tab use the START button to start the OS scheduler and observe the changing process states for few seconds.

m. Have you got a deadlock condition same as you constructed in (1) above? If you haven't then check and if necessary re-do above. Do not proceed to (n) or (3) below until you get a deadlock condition. n. If you have a deadlock condition then click on the SHOW DEADLOCKED PROCESSES... button in the System Resources

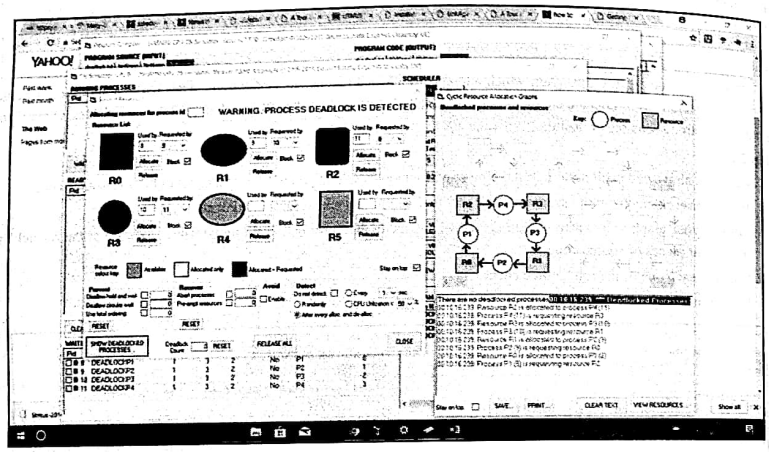


Fig. L14

Step 3 : Now that you created a deadlock condition let us try getting out of this condition :

a. In the System Resources window, there should be four resource shapes that are in red colour indicating they are both allocated to one process and requested by another.

b. Select one of these resources and click on the Release button next to it.

c. Observe what is happening to the processes in the OS Simulator window.

d. Is the deadlock situation resolved? Explain briefly why this helped resolve the deadlock.

e. Re-create the same deadlock condition (steps in 2 above should help).

f. Once the deadlock condition is obtained again do the following: In the OS Simulator window, select a process in the waiting queue in the WAITING PROCESSES frame.

g. Click on the REMOVE button and observe the processes.

h. Has this managed to resolve the deadlock.

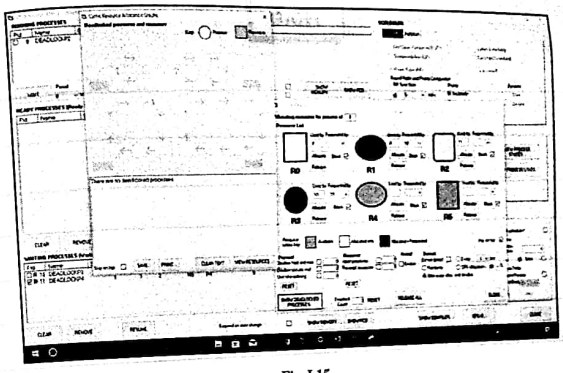


Fig. L15