Paper Name: Operating System
Paper Code: CS502
Contact Hours/Week: 3
Credit: 3
Total Contact Hours: 32L

Objective(s)

1. To understand the services provided by and the design of an operating system.
2. To understand the structure and organization of the file system.
3. To understand what a process is and how processes are synchronized and scheduled.
4. To understand different approaches to memory management.
5. Students should be able to use system calls for managing processes, memory and the file system.
6. Students should understand the data structures and algorithms used to implement an OS.

Outcome(s)

1. Describe how computing resources (such as CPU, memory and I/O) are managed by the operating system.
2. Analyze kernel and user mode in an operating system.
3. Solve different CPU scheduling problem to achieve specific scheduling criteria.
4. Apply the knowledge of process management, synchronization, deadlock to solve basic problems.
5. Evaluate and report appropriate design choices when solving real-world problems

Prerequisites:

1. Computer organization
2. Computer Architecture
3. Data Structures
4. Algorithms & Programming Concept

Module – 1: [3L]

Functionalities of Operating System, Evolution of Operating System.
Types of Operating System: batch, multi-programmed, time-sharing, real-time, distributed, parallel, Structural overview, Protection & Security. [3L]

Module – 2: [9L]

Processes: Concept of processes, process states, PCB, process scheduling, co-operating processes, independent process, suspended process, Interaction between processes and OS, Inter-process communication: Message passing. [2L]
Threads: overview, benefits of threads, user and kernel level threads, Thread models. [2L]
CPU scheduling: scheduling criteria, preemptive & non-preemptive scheduling, scheduling algorithms (FCFS, SJF, SRTF, RR, priority, multilevel queue, multilevel feedback queue scheduling). [5L]

Module – 3: [9L]

Process Synchronization: background, critical section problem, synchronization hardware, classical problems of synchronization(producer-consumer, readers-writer, dining philosophers, etc), semaphores, monitors.
[5L]

Deadlocks: deadlock characterization, methods for handling deadlocks, deadlock prevention, deadlock avoidance, deadlock detection, recovery from deadlock.
[4L]

Module – 4: [6L]

Background, logical vs. physical address space, swapping, contiguous memory allocation, paging, Segmentation, TLB. [3L]

Virtual Memory: background, demand paging, page replacement algorithms (FCFS, LRU, Optimal), thrashing, Working set model. [3L]

Module – 5: [5L]

Disk structure, disk scheduling (FCFS, SSTF, SCAN,C-SCAN,LOOK,C-LOOK etc),disk reliability, disk formatting, boot block, bad blocks. [2L]

File: File concept, access methods, directory structure, file system structure, UNIX file structure, allocation methods (contiguous, linked, indexed), free-space management (bit vector).
[2L]
I/O: I/O hardware, polling, interrupts, DMA, caching, buffering, blocking-non blocking I/O.
[1L]

Text Book:

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts.
2. *Operating Systems* & Systems Programming by P *Balakrishna Prasad*

Reference Book:

1. Dietel H. N., "An Introduction to Operating Systems", Addison Wesley.
2. Andrew Tanenbaum, Modern Operating Systems, Prentice Hall.
3. William Stallings, Operating Systems, Prentice Hall.

Lesson Plan for B.Tech Computer Science and Engineering Programme(Autonomy)
Paper Name: Operating System
Paper Code: CS502
Contact Hours/Week: 3
Credit: 3
Total Contact Hours: 36L

Module No.	Course Content	Lecture Required	Reference / Text Books
1	Functionalities of Operating System, Evolution of Operating System. Types of Operating System: batch, multi-programmed, time-sharing, real-time, distributed, parallel, Structural overview, Protection & Security. [3L]	3L	**Text Book:** 1.Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts. *2.Operating Systems &* Systems Programming by P *Balakrishna Prasad* **Reference Book:** 1. Dietel H. N., "An Introduction to Operating Systems", Addison Wesley. 2.Andrew Tanenbaum, Modern Operating Systems, Prentice Hall.
2	**Module – 2:** [9L] Processes: Concept of processes, process states, PCB, process scheduling, co-operating processes, independent process, suspended process, Interaction between processes and OS, Inter-process communication: Message passing. [2L] Threads: overview, benefits of threads, user and kernel level threads, Thread models. [2L] CPU scheduling: scheduling criteria, preemptive & non-preemptive scheduling, scheduling algorithms (FCFS, SJF, SRTF, RR, priority, multilevel queue, multilevel feedback queue scheduling). [5L]	9L	**Text Book:** 1.Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts. *2.Operating Systems &* Systems Programming by P *Balakrishna Prasad* **Reference Book:** 1.William Stallings, Operating Systems, Prentice Hall.
3	**Module – 3:** [9L]		**Text Book:** 1.Abraham

	Process Synchronization: background, critical section problem, synchronization hardware, classical problems of synchronization(producer-consumer, readers-writer, dining philosophers, etc), semaphores, monitors. [5L] Deadlocks: deadlock characterization, methods for handling deadlocks, deadlock prevention, deadlock avoidance, deadlock detection, recovery from deadlock. [4L]	9L	Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts. *2.Operating Systems* & Systems Programming by P *Balakrishna Prasad* **Reference Book:** 1. Dietel H. N., "An Introduction to Operating Systems", Addison Wesley. 2.William Stallings, Operating Systems, Prentice Hall.
4	**<u>Module – 4:</u>** [6L] Background, logical vs. physical address space, swapping, contiguous memory allocation, paging, Segmentation, TLB. [3L] Virtual Memory: background, demand paging, page replacement algorithms (FCFS, LRU, Optimal), thrashing, Working set model. [3L]	6L	**Text Book:** 1.Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts. *2.Operating Systems* & Systems Programming by P *Balakrishna Prasad* **Reference Book:** 1.Andrew Tanenbaum, Modern Operating Systems, Prentice Hall. 2.William Stallings, Operating Systems, Prentice Hall.
5	**<u>Module – 5:</u>** [5L] Disk structure, disk scheduling (FCFS, SSTF, SCAN,C-SCAN,LOOK,C-LOOK etc),disk reliability, disk formatting, boot block, bad blocks. [2L] File: File concept, access methods,	5L	**Text Book:** 1.Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts. *2.Operating Systems* & Systems Programming by P *Balakrishna Prasad*

	directory structure, file system structure, UNIX file structure, allocation methods (contiguous, linked, indexed), free-space management (bit vector). [2L] I/O: I/O hardware, polling, interrupts, DMA, caching, buffering, blocking-non blocking I/O. [1L]		1.Andrew Tanenbaum, Modern Operating Systems, Prentice Hall. 2.William Stallings, Operating Systems, Prentice Hall.

MODULE 1

OVERVIEW

LECTURE: 1

Introduction to Operating Systems

A computer system has many resources (hardware and software), which may be require to complete a task. The commonly required resources are input/output devices, memory, file storage space, CPU etc. The operating system acts as a manager of the above resources and allocates them to specific programs and users, whenever necessary to perform a particular task. Therefore operating system is the resource manager i.e. it can manage the resource of a computer system internally. The resources are processor, memory, files, and I/O devices. **In simple terms, an operating system is the interface between the user and the machine.**

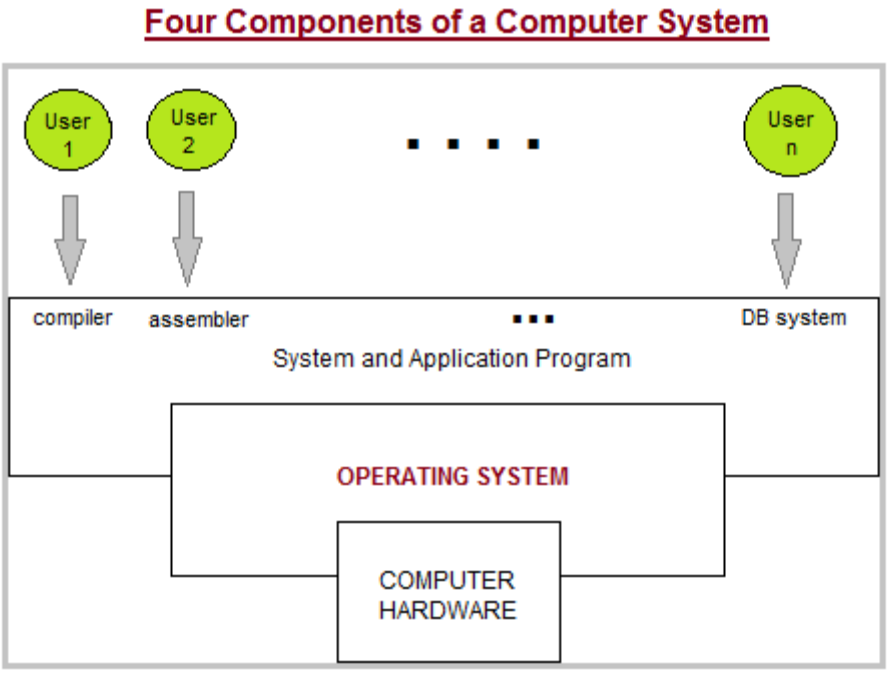


Fig 1.1

Two Views of Operating System

1. User's View
2. System View

Operating System: User View

The user view of the computer refers to the interface being used. Such systems are designed for one user to monopolize its resources, to maximize the work that the user is performing. In these cases, the operating system is designed mostly for ease of use, with some attention paid to performance, and none paid to resource utilization.

Operating System: System View

Operating system can be viewed as a resource allocator also. A computer system consists of many resources like - hardware and software - that must be managed efficiently. The operating system acts as the manager of the resources, decides between conflicting requests, controls execution of programs etc.

Operating System Management Tasks

1. **Processor management** which involves putting the tasks into order and pairing them into manageable size before they go to the CPU.
2. **Memory management** which coordinates data to and from RAM (random-access memory) and determines the necessity for virtual memory.
3. **Device management** which provides interface between connected devices.
4. **Storage management** which directs permanent data storage.
5. **Application** which allows standard communication between software and your computer.
6. **User interface** which allows you to communicate with your computer.

Functions of Operating System

1. It boots the computer
2. It performs basic computer tasks e.g. managing the various peripheral devices e.g. mouse, keyboard
3. It provides a user interface, e.g. command line, graphical user interface (GUI)
4. It handles system resources such as computer's memory and sharing of the central processing unit(CPU) time by various applications or peripheral devices.
5. It provides file management which refers to the way that the operating system manipulates, stores, retrieves and saves data.

6. Error Handling is done by the operating system. It takes preventive measures whenever required to avoid errors.

Evolution of Operating Systems

The evolution of operating systems is directly dependent on the development of computer systems and how users use them. Here is a quick tour of computing systems through the past fifty years in the timeline.

Early Evolution

- 1945: **ENIAC**, Moore School of Engineering, University of Pennsylvania.
- 1949: **EDSAC** and **EDVAC**
- 1949: **BINAC** - a successor to the ENIAC
- 1951: **UNIVAC** by Remington
- 1952: **IBM** 701
- 1956: The interrupt
- 1954-1957: **FORTRAN** was developed

Operating Systems - Late 1950s

By the late 1950s Operating systems were well improved and started supporting following usages:

- It was able to perform **Single stream batch processing**.
- It could use Common, standardized, input/output routines for device access.
- Program transition capabilities to reduce the overhead of starting a new job was added.
- **Error recovery** to clean up after a job terminated abnormally was added.
- Job control languages that allowed users to specify the job definition and resource requirements were made possible.

Operating Systems - In 1960s

- 1961: The dawn of minicomputers
- 1962: Compatible Time-Sharing System (CTSS) from MIT
- 1963: Burroughs Master Control Program (MCP) for the B5000 system
- 1964: IBM System/360
- 1960s: Disks became mainstream
- 1966: Minicomputers got cheaper, more powerful, and really useful.
- 1967-1968: **Mouse** was invented.
- 1964 and onward: Multics
- 1969: The UNIX Time-Sharing System from Bell Telephone Laboratories.

Supported OS Features by 1970s

- **Multi User** and **Multi tasking** was introduced.
- **Dynamic address** translation hardware and **Virtual machines** came into picture.
- **Modular architectures** came into existence.
- Personal, interactive systems came into existence.

Accomplishments after 1970

- 1971: Intel announces the microprocessor
- 1972: IBM comes out with VM: the Virtual Machine Operating System
- 1973: UNIX 4th Edition is published
- 1973: Ethernet
- 1974 The Personal Computer Age begins

- 1974: Gates and Allen wrote BASIC for the Altair
- 1976: Apple II
- August 12, 1981: IBM introduces the IBM PC
- 1983 Microsoft begins work on MS-Windows
- 1984 Apple Macintosh comes out
- 1990 Microsoft Windows 3.0 comes out
- 1991 GNU/Linux
- 1992 The first Windows virus comes out
- 1993 Windows NT
- 2007: iOS
- 2008: Android OS

And as the research and development work continues, we are seeing new operating systems being developed and existing ones getting improved and modified to enhance the overall user experience, making operating systems fast and efficient like never before.

Also, with the onset of new devies like **wearables**, which includes, **Smart Watches**, **Smart Glasses**, **VR gears** etc, the demand for unconventional operating systems is also rising.

LECTURE: 2

Types of Operating Systems

Following are some of the most widely used types of Operating system.

1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Realtime Operating System
8. Handheld System

Simple Batch Systems

- In this type of system, there is **no direct interaction between user and the computer**.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.

Advantages of Simple Batch Systems

1. No interaction between user and computer.

2. No mechanism to prioritise the processes.

Fig:1.2

Multiprogramming Batch Systems

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on **minimizing the response time**, while in multiprogramming the prime focus is to maximize the CPU usage.

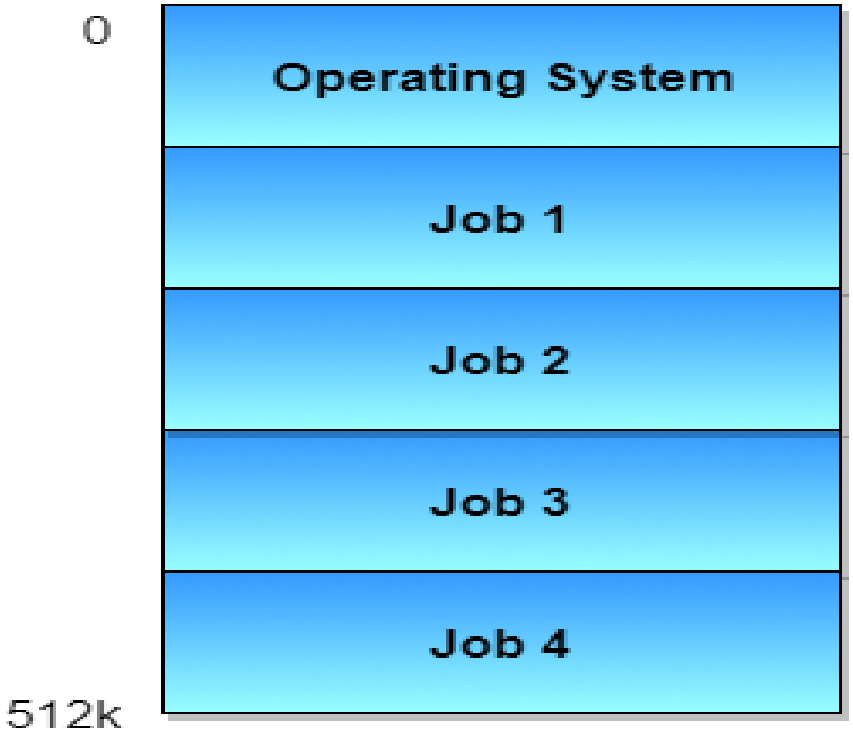


Fig:1.3

Multiprocessor Systems

A Multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

Advantages of Multiprocessor Systems

1. Enhanced performance
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

Desktop Systems

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither **multiuser** nor **multitasking**. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These

systems are called **Desktop Systems** and include PCs running `Microsoft Windows` and the `Apple Macintosh`. Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.

Microcomputers were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

Distributed Operating System

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

Advantages Distributed Operating System

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

Types of Distributed Operating Systems

Following are the two types of distributed operating systems used:

1. Client-Server Systems
2. Peer-to-Peer Systems

Client-Server Systems

Centralized systems today act as **server systems** to satisfy requests generated by **client systems**. The general structure of a client-server system is depicted in the figure below:

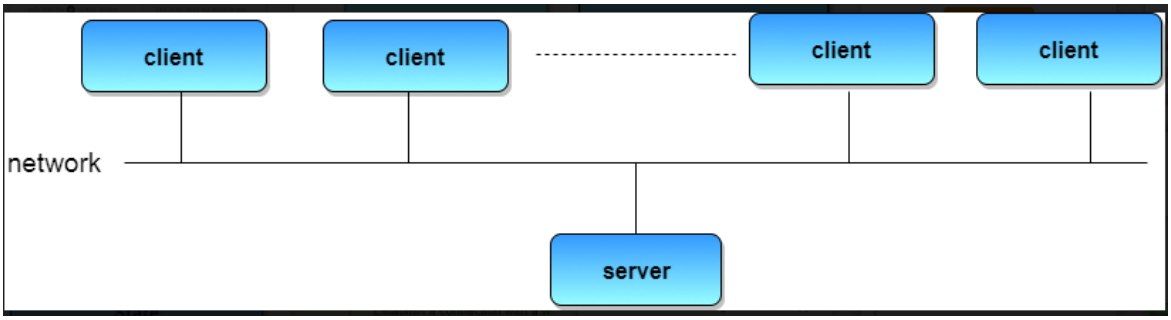


Fig:1.4

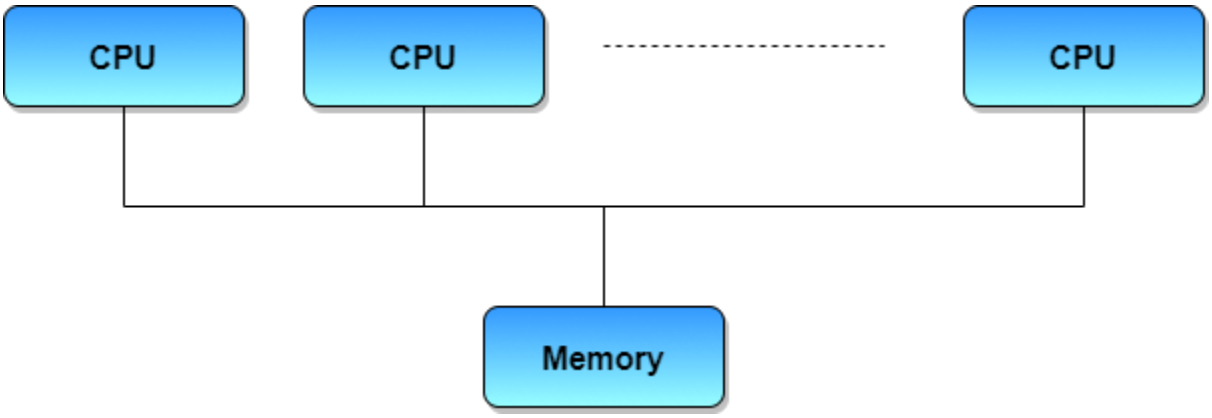
Server Systems can be broadly categorized as: **Compute Servers** and **File Servers**.

- **Compute Server systems**, provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File Server systems**, provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-Peer Systems

The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems. When PCs were introduced in the 1970s, they were designed for **personal** use and were generally considered standalone computers. With the beginning of widespread public use of the Internet in the 1990s for electronic mail and FTP, many PCs became connected to computer networks.

In contrast to the **Tightly Coupled** systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled systems (or distributed systems). The general structure of a client-server system is depicted in the figure below:



Clustered Systems Fig:1.5

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
- Clustering is usually performed to provide **high availability**.
- A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.
- **Asymmetric Clustering -** In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.
- **Symmetric Clustering -** In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.
- **Parallel Clustering -** Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system's usage and it's features should expand greatly as **Storage Area Networks(SANs)**. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

Real Time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.**

While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**.

Handheld Systems

Handheld systems include **Personal Digital Assistants(PDAs)**, such as `Palm-Pilots` or `Cellular Telephones` with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

- Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
- Currently, many handheld devices do **not use virtual memory** techniques, thus forcing program developers to work within the confines of limited physical memory.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
- The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology such as **BlueTooth**, allowing remote access to e-mail and web browsing. **Cellular telephones** with connectivity to the Internet fall into this category. Their use continues to expand as network connections become more available and other options such as `cameras` and `MP3 players`, expand their utility.

Questions :-

Multiple Choice Questions

i)Which one of the following is not shared by threads?

a) program counter

b) stack

c) both program counter and stack

d) none of the mentioned

ii) A process can be

a) single threaded

b) multithreaded

c) both single threaded and multithreaded

d) none of the mentioned

iii)If one thread opens a file with read privileges then

a) other threads in the another process can also read from that file

b) other threads in the same process can also read from that file

c) any other thread can not read from that file

d) all of the mentioned

iv)The time required to create a new thread in an existing process is

a) greater than the time required to create a new process

c) less than the time required to create a new process

c) equal to the time required to create a new process

d) none of the mentioned

v)When the event for which a thread is blocked occurs,

a) thread moves to the ready queue

b) thread remains blocked

c) thread completes

d) a new thread is provided

vi) What is not a important part of security protection ?

a) Large amount of RAM to support antivirus

b) Strong passwords

c) Audit log periodically

d) Scan for unauthorized programs in system directories

vii)What is used to protect network from outside internet access ?

a) A trusted antivirus

b) 24 hours scanning for virus

c) Firewall to separate trusted and untrusted network

d) Deny users access to websites which can potentially cause security leak

viii) What is are two safe computing practices ?

a) Not to open software from unknown vendors

b) Open and execute programs in admin level/root

c) Open and execute programs in presence of antivirus

d) None of the mentioned

ix)How do viruses avoid basic pattern matchof antivirus ?

a) They are encrypted

b) They act with special permissions

c) They modify themselves

d) None of the mentioned

x) How does an antivirus of today identify viruses ?

a) Previously known patterns

b) It can detect unknown patterns

c) It can take high priority to increase scanning speed

d) None of the mentioned

2. Short AnswerType Questions :-

i)What is an operating system?

ii) What are its main functions?

iii)Describe system calls and its type

vi)What is a Kernel?

v)What are the main functions of a Kernel?

vi)What are the different types of Kernel?

vii)What is a command interpreter?

MODULE 2

Process Management

LECTURE: 1

Process

A process basically is a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −

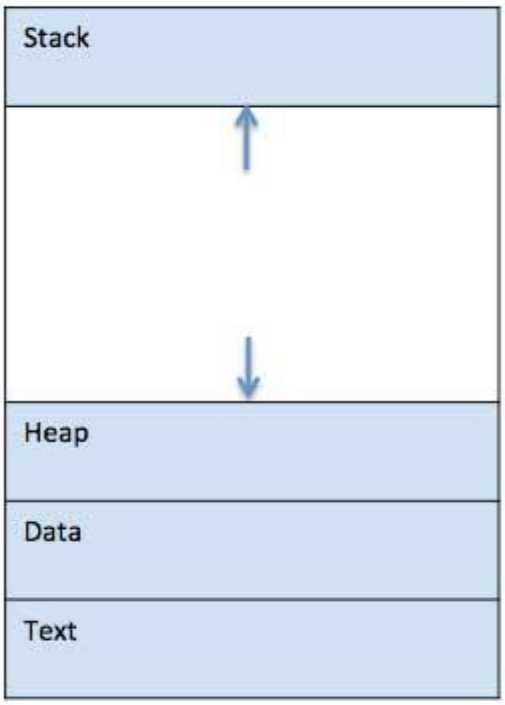


Fig:2.1

Component & Description

1. **Stack**

The process Stack contains the temporary data such as method/function parameters, return address and local variables.

2. **Heap**

This is dynamically allocated memory to a process during its run time.

3. **Text**

This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

4. **Data**

This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include <stdio.h>
int main() {
   printf("Hello, World! \n");
   return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

State & Description of a Process LifeCycle

Start

This is the initial state when a process is first started/created.

Ready

The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.

Running

Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Waiting

Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

Terminated or Exit

Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

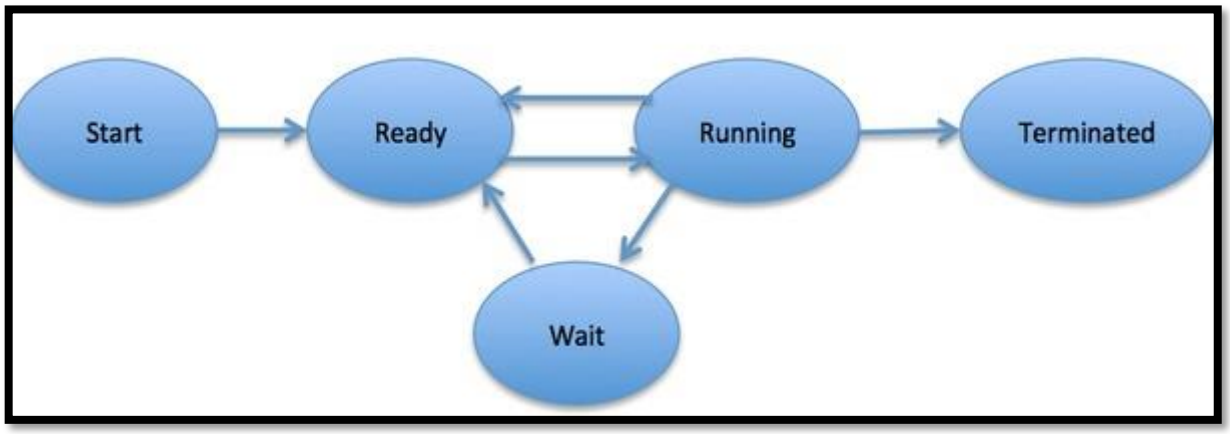


Fig: 2.2

Process Control Block(PCB)-

In modern sophisticated multitasking systems, the PCB stores many different items of data, all needed for correct and efficient process management.[1] Though the details of these structures are obviously system-dependent, we can identify some very common parts, and classify them in three main categories:

- Process identification data
- Process state data
- Process control data

The approach commonly followed to represent this information is to create and update status tables for each relevant entity, like memory, I/O devices, files and processes.

Memory tables, for example, may contain information about the allocation of main and secondary (virtual) memory for each process, authorization attributes for accessing memory areas shared among different processes, etc. I/O tables may have entries stating the availability of a device or its assignment to a process, the status of I/O operations being executed, the location of memory buffers used for them, etc.

File tables provide info about location and status of files. Finally, process tables store the data the OS needs to manage processes. At least part of the process control data structure is always maintained in main memory, though its exact location and configuration varies with the OS and the memory management technique it uses.

Process identification data always include a unique identifier for the process (almost invariably an integer number) and, in a multiuser-multitasking system, data like the identifier of the parent process, user identifier, user group identifier, etc. The process id is particularly relevant, since it is

often used to cross-reference the OS tables defined above, e.g. allowing to identify which process is using which I/O devices, or memory areas.

Process state data are those pieces of information that define the status of a process when it is suspended, allowing the OS to restart it later and still execute correctly. This always includes the content of the CPU general-purpose registers, the CPU process status word, stack and frame pointers etc. During context switch, the running process is stopped and another process is given a chance to run. The kernel must stop the execution of the running process, copy out the values in hardware registers to its PCB, and update the hardware registers with the values from the PCB of the new process.

Process control information is used by the OS to manage the process itself. This includes:

- **The process scheduling state**, e.g. in terms of "ready", "suspended", etc., and other scheduling information as well, like a priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for.
- **Process structuring information:** process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures.
- **Interprocess communication information:** various flags, signals and messages associated with the communication among independent processes may be stored in the PCB.
- **Process Privileges** in terms of allowed/disallowed access to system resources.
- **Process State:** State may enter into new, ready, running, waiting, dead depending on CPU scheduling.
- **Process Number:** A unique identification number for each process in the operating system.
- **Program Counter:** A pointer to the address of the next instruction to be executed for this process.
- **CPU Registers:** Indicates various register set of CPU where process need to be stored for execution for running state.
- **CPU scheduling Information**: indicates the information of a process with which it uses the CPU time through scheduling.
- **Memory Management Information:** includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- **Accounting information:** Includes the amount of CPU used for process execution, time limits, execution ID etc.
- **I/O Status Information:** Includes a list of I/O devices allocated to the process.

LECTURE: 2

Process Scheduling-

Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

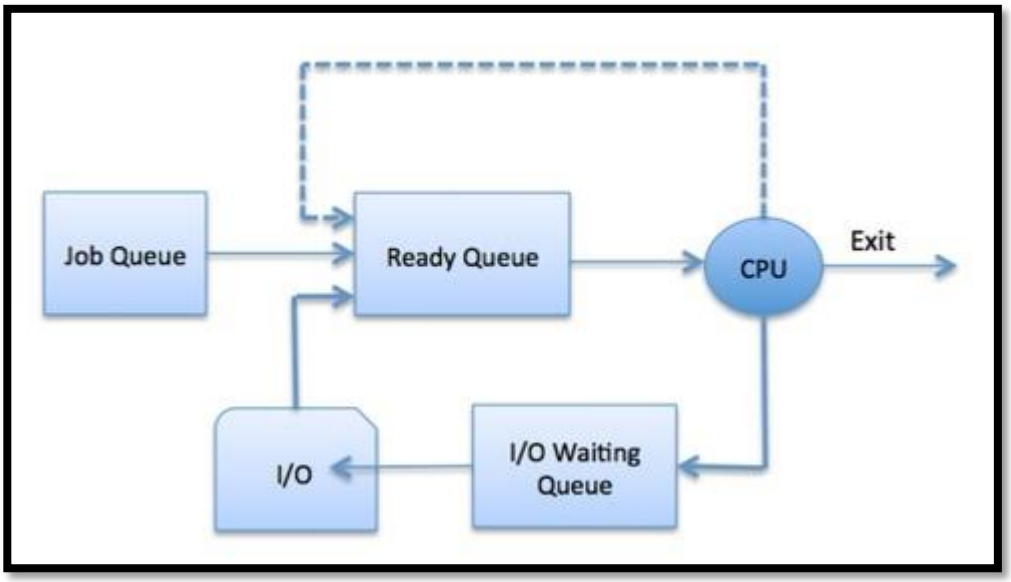


Fig: 2.3

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Process Co-operation is needed for –

i)Information sharing:

Several users may need to access same piece of information, so there should be an environment to allow concurrent access to these types of resources.

ii)Computation speedup:

To make any task run faster , It should be divided into subtasks, each of which will be executing in parallel with the others. Speedup can achieved if computer has multiple processing elements such as CUP or I/O channels.

iii)Modularity: Constructing system in a modular fashion, by dividing the system function into separate processes or thread.

iv)Convenience:

Concurrent execution of cooperation processes requires mechanisms that allow communicating processes with each other's and synchronizing their actions.

The concurrent processes in operating system are of two types

1. Independent processes

Independent process *is* the process that can not affect or be affected by the other processes. Independent processes does not share any data like temporary or persistent with any other process.

2. Cooperating processes

Cooperating process is affect or be affected by the other processes executing in the system. Cooperating process shares data with other processes.

Suspended Process-

Some of the reasons to suspend a process are...

1.If one process is ready to excecute,but there is no space in the main memory,then it is suspended.

2.when one process in main memory which was blocked & there is another process ready to excecute,but waiting in secondary memory,then the process in main memory is suspended.

3.when the parent process suspends,then the sub process is also suspended.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.

Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

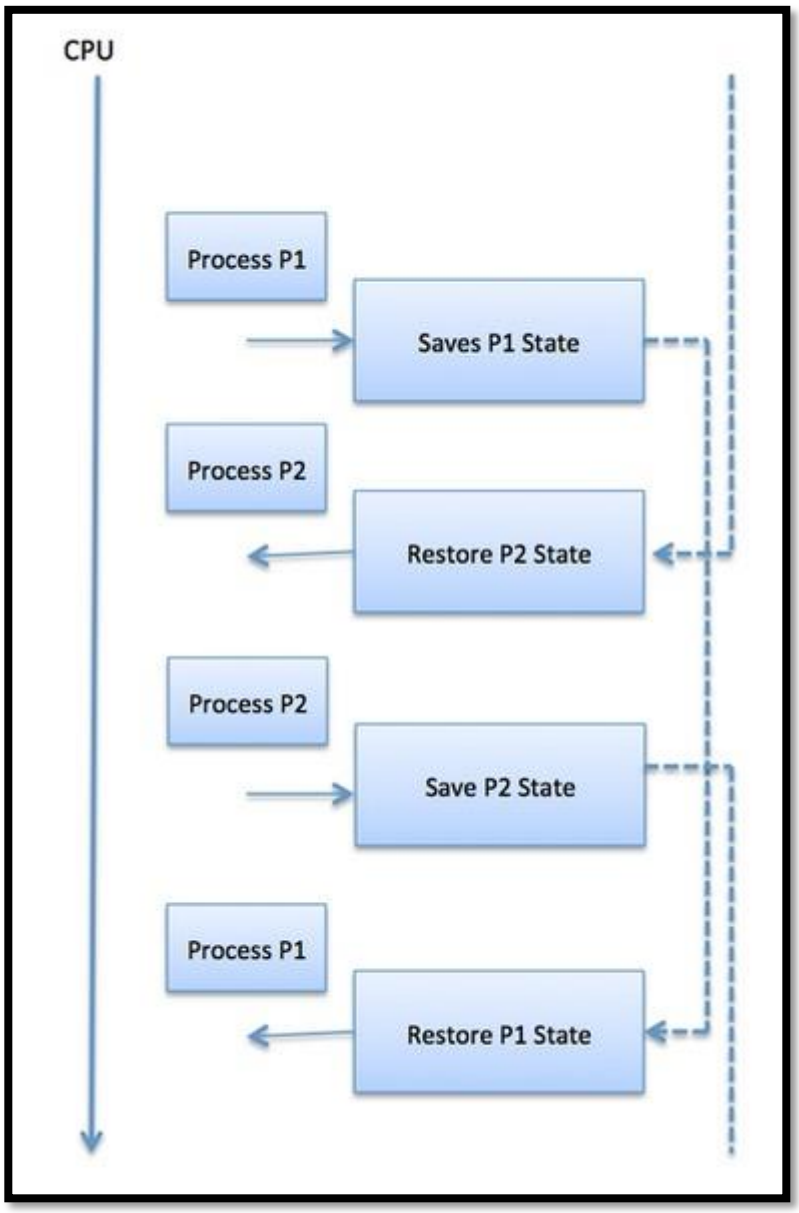


Fig: 2.4

Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information

- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

IPC-

In this section you will learn about the various working capabilities of IPC (Inter process communication) within an Operating system along with usage. Processes executing concurrently in the operating system might be either independent processes or cooperating processes. A process is independent if it cannot be affected by the other processes executing in the system.

There are numerous reasons for providing an environment or situation which allows process co-operation:

- Information sharing: Since a number of users may be interested in the same piece of information (for example, a shared file), you must provide a situation for allowing concurrent access to those information.
- Computation speedup: If you want a particular work to run fast, you must break it into sub-tasks where each of them will get execute in parallel with the other tasks. Note that such a speed-up can be attained only when the computer has compound or various processing elements like CPUs or I/O channels.
- Modularity: You may want to build the system in a modular way by dividing the system functions into split processes or threads.
- Convenience: Even a single user may work on many tasks at a time. For example, a user may be editing, formatting, printing, and compiling in parallel.

Working together multiple processes, require an inter process communication (IPC) method which will allow them to exchange data along with various information. There are two primary models of inter process communication:

1. shared memory and
2. message passing.

In the shared-memory model, a region of memory which is shared by cooperating processes gets established. Processes can then able to exchange information by reading and writing all the data to the shared region. In the message-passing form, communication takes place by way of messages exchanged among the cooperating processes.

The two communications models are contrasted in figure below:

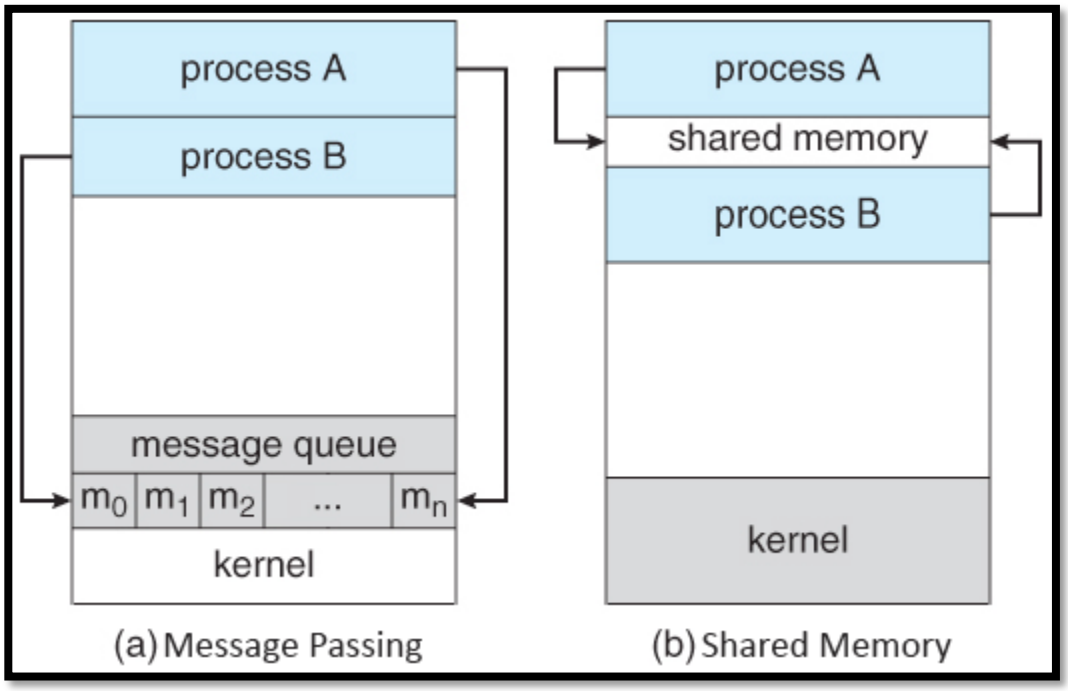


Fig: 2.5

Shared Memory Systems

Inter process communication (IPC) usually utilizes shared memory that requires communicating processes for establishing a region of shared memory. Typically, a shared-memory region resides within the address space of any process creating the shared-memory segment. Other processes that wish for communicating using this shared-memory segment must connect it to their address space.
More on Inter Process Shared Memory

Note that, normally what happens, the operating system tries to check one process from accessing other's process's memory. Shared memory needs that two or more processes agree to remove this limitation. They can then exchange information via reading and writing data within the shared areas.

The form of the data and the location gets established by these processes and are not under the control of operating system. The processes are also in charge to ensure that they are not writing to the same old location simultaneously.

LECTURE: 3

Thread

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

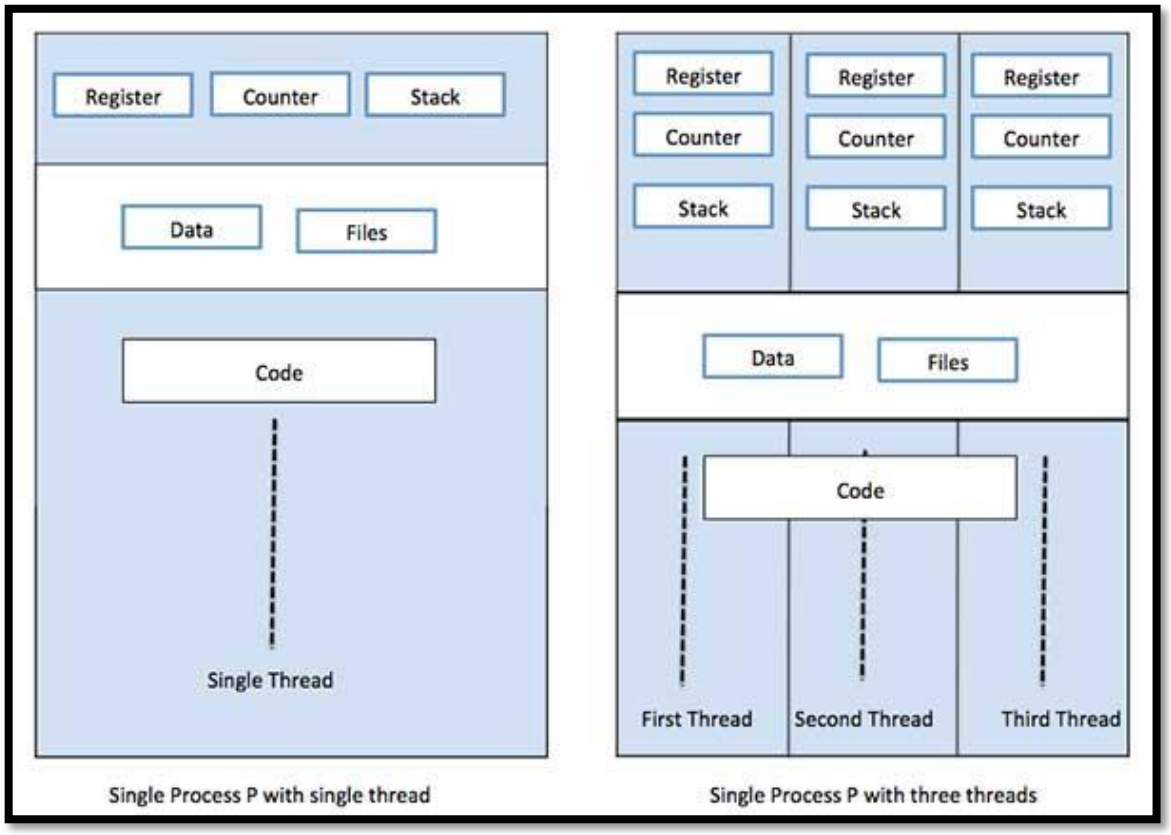


Fig: 2.6

Difference between Process and Thread

S.N.	Process	Thread

1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, 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.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

LECTURE: 4

Types of Thread

Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.

- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

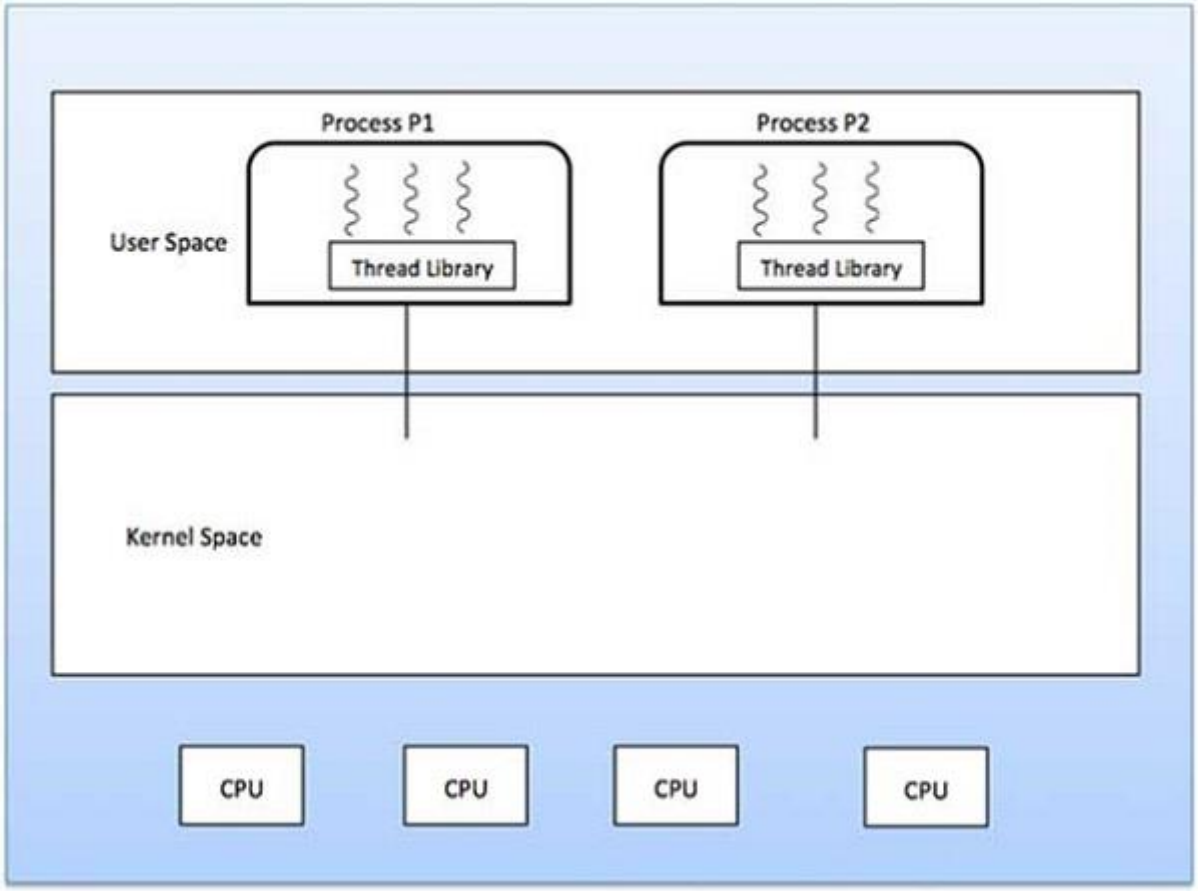


Fig: 2.7

Advantages-

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages-

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

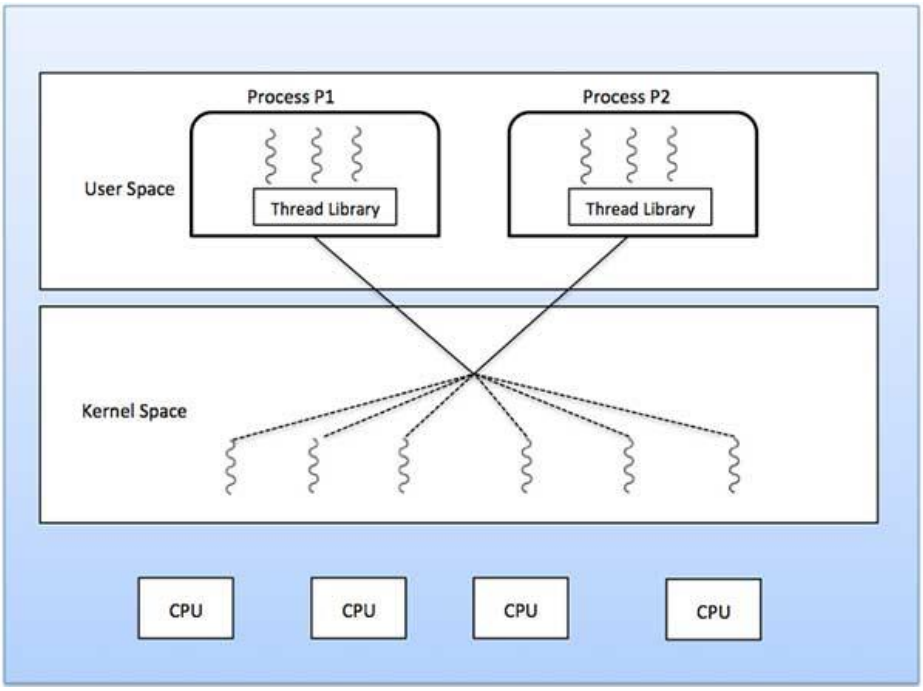


Fig: 2.8

Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

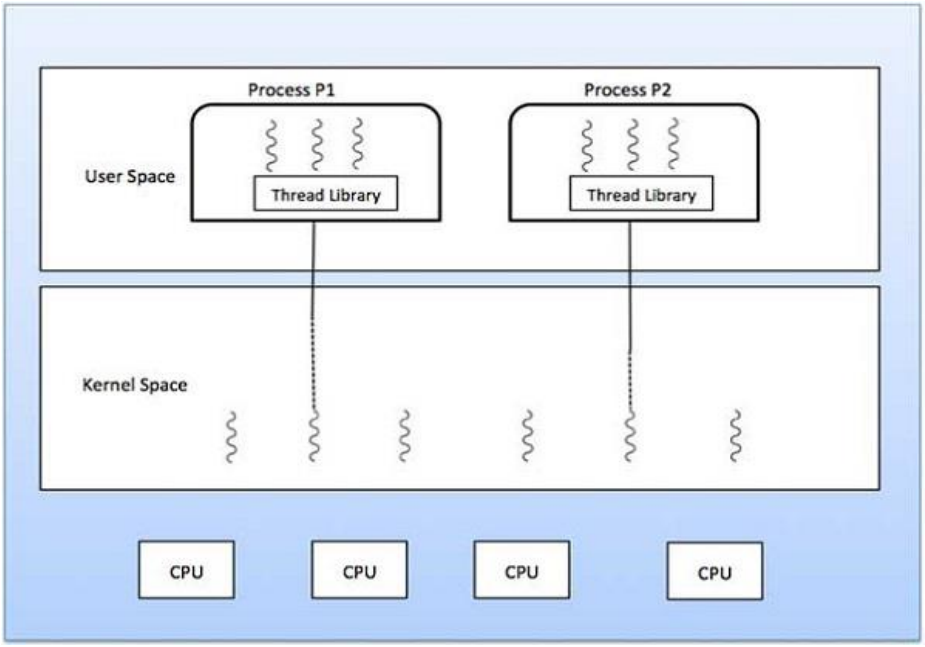


Fig: 2.9

One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.

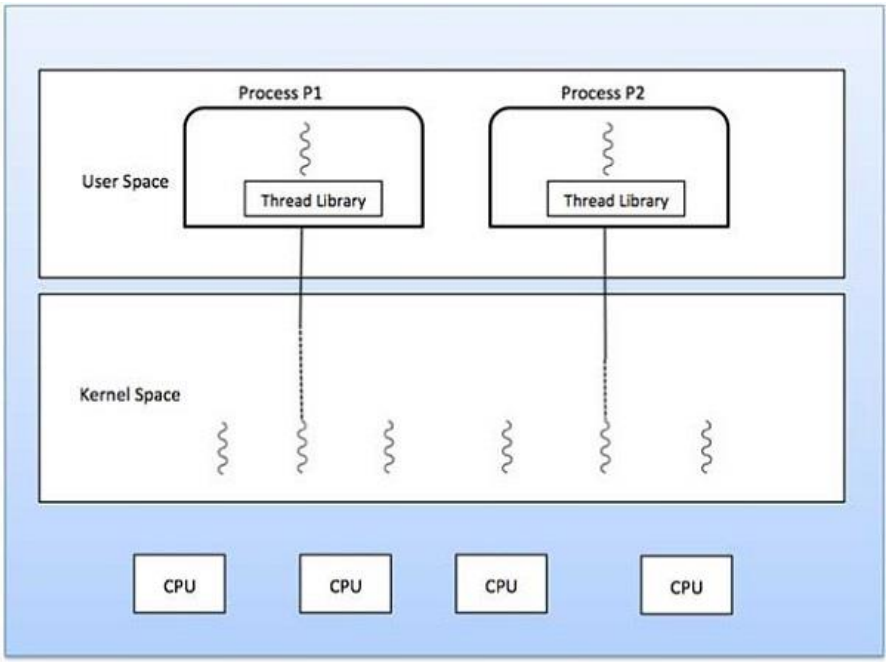


Fig: 2.10

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take	Kernel routines themselves can be

	advantage of multiprocessing.	multithreaded.

LECTURE: 5

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

CPU Scheduling: Dispatcher

Another component involved in the CPU scheduling function is the **Dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context

- Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**. Dispatch Latency can be explained using the below figure:

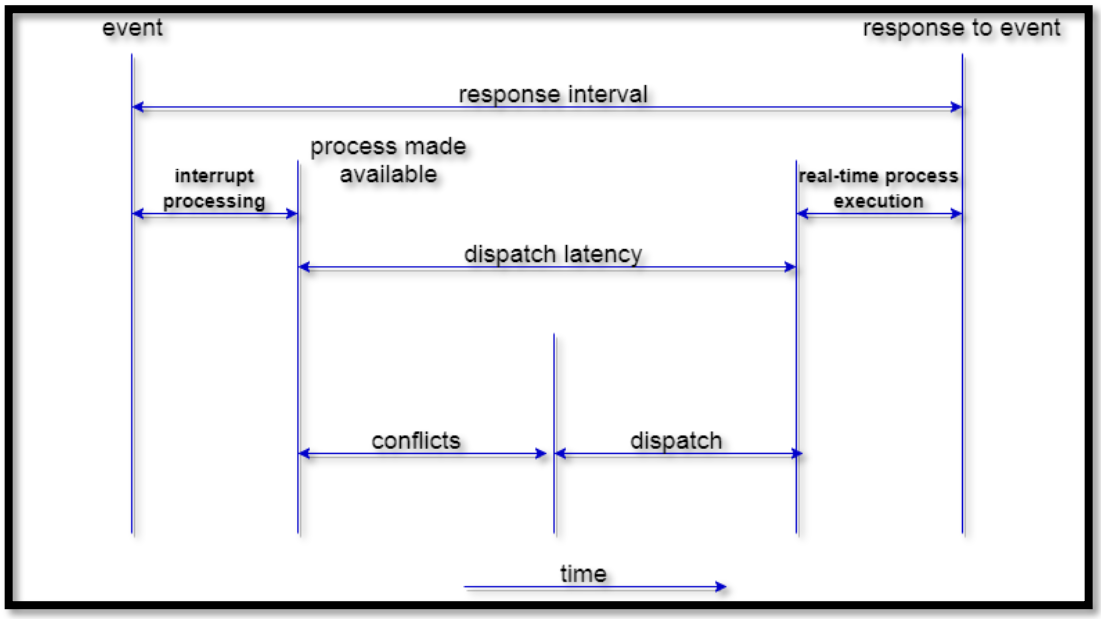


Fig: 2.11

Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

CPU Scheduling: Scheduling Criteria

There are many different criterias to check when considering the **"best"** scheduling algorithm, they are:

CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Pre-emptive scheduler:	**Non pre-emptive scheduler**

▪ Scheduler has ability to move process from running state to other state and let another process run when interrupt occur. Also called pre-emptive scheduling.	▪ Cannot take cup (processor) away from a process. Also called non pre-emptive scheduling.
▪ Pre-emptive scheduling is helpful in multi programming environment by using time slots. Using time slots an process cannot enter into infinite loop and block the whole system.	▪ When we need processor do not halt in middle of something very important non pre-emptive environment is preferable.Processes in kernel mode usually run in non-pre-emptive mode. The process gives up control voluntarily.

LECTURE: 6

Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilisation, computer scientists have defined some algorithms, they are:

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

Scheduling Algorithms-

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling

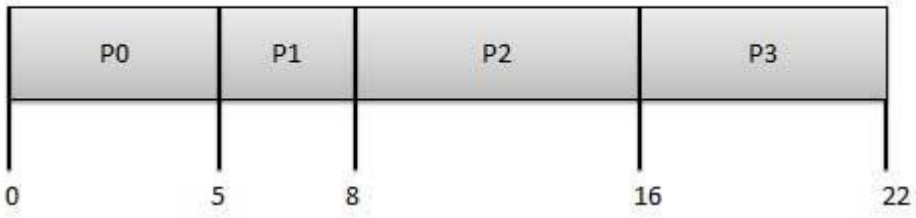
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	**Wait Time : Service Time - Arrival Time**

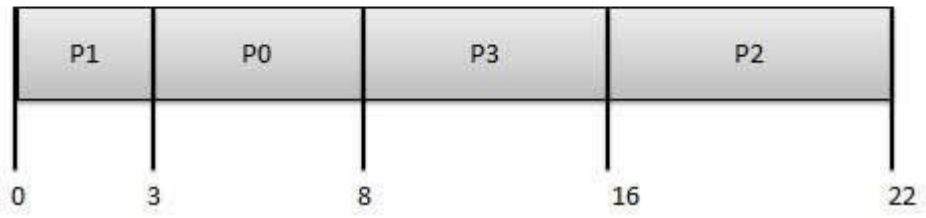
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Wait time of each process is as follows −

Process	**Wait Time : Service Time - Arrival Time**

P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

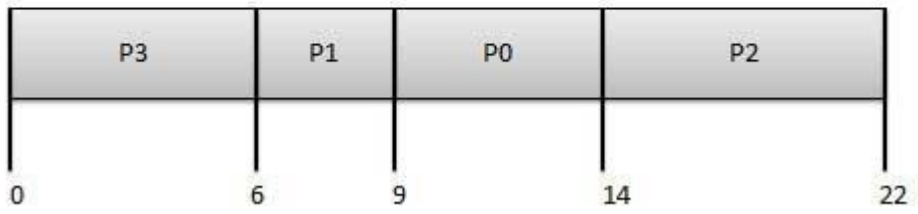
Average Wait Time: (3+0+14+5) / 4 = 5.50

LECTURE: 7

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows −

Process	**Wait Time : Service Time - Arrival Time**

P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

LECTURE: 8

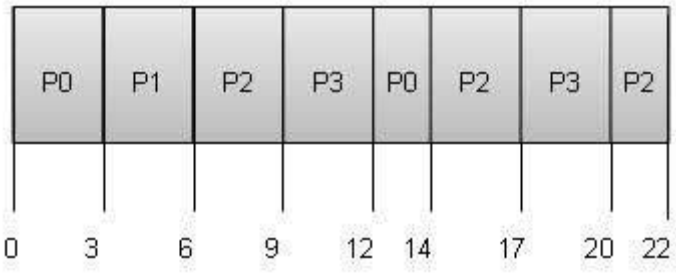
Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Quantum = 3



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

LECTURE: 9

Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

For example: A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. **For example:** The foreground queue may have absolute priority over the background queue.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes

3. Interactive Editing Processes

4. Batch Processes

5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.

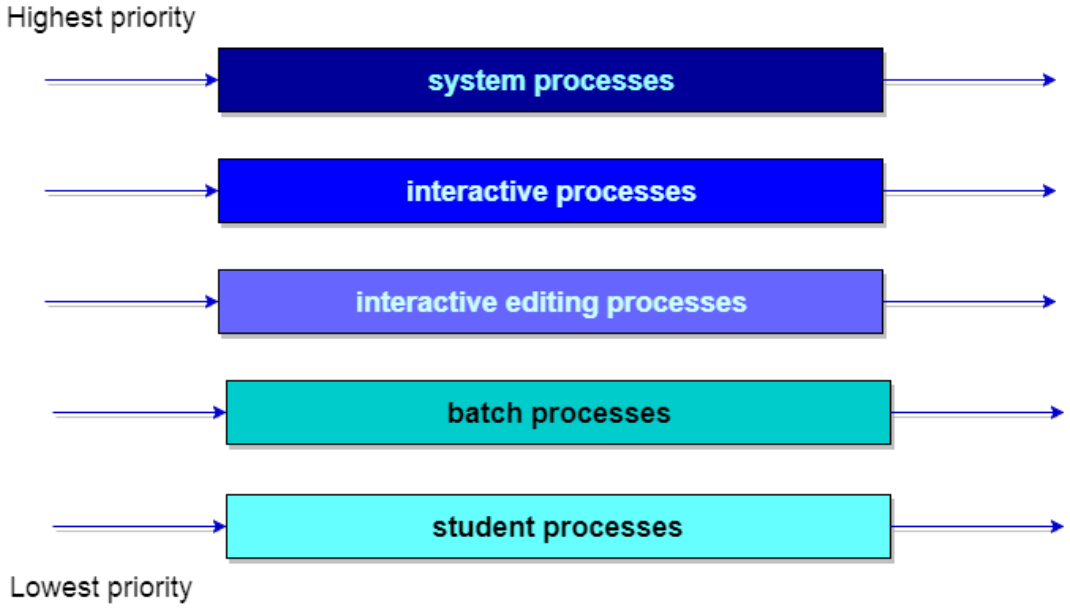


Fig: 2.12

Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

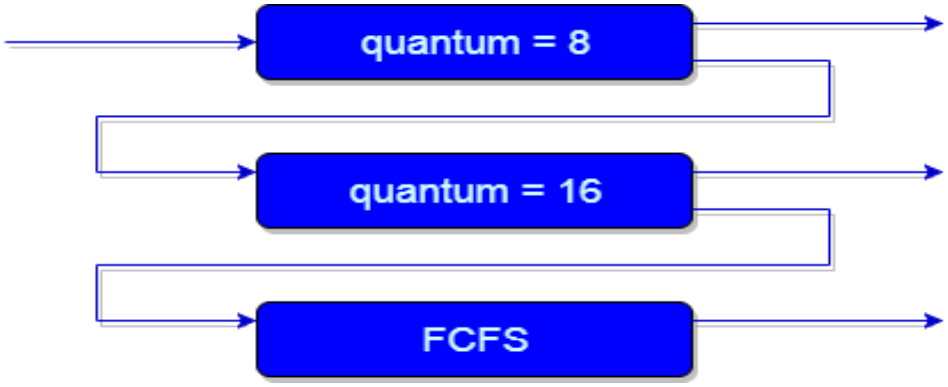


Fig: 2.13

An example of a multilevel feedback queue can be seen in the below figure.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.

Questions :-

1.Some Mutiple Choice Questions :

i) The systems which allows only one process execution at a time, are called
   a) uniprogramming systems
   b) uniprocessing systems
   c) unitasking systems
   d) none of the mentioned

ii) In operating system, each process has its own
   a) address space and global variables
   b) open files
   c) pending alarms, signals and signal handlers
   d) all of the mentioned

iii) In Unix, Which system call creates the new process?
   a) fork
   b) create
   c) new
   d) none of the mentioned

iv) A process can be terminated due to
   a) normal exit
   b) fatal error
   c) killed by another process
   d) all of the mentioned

v) What is the ready state of a process?
   a) when process is scheduled to run after some execution
   b) when process is unable to run until some task has been completed
   c) when process is using the CPU
   d) none of the mentioned

vi) An optimal scheduling algorithm in terms of minimizing the average waiting time of a given set of process is …
   a. FCFS scheduling
   b. Round robin scheduling algorithm
   c. Shortest job first scheduling algorithm
   d. Priority scheduling algorithm

vii) RR scheduling is most suitable for ……..
   a. time shared OS
   b. distributed OS
   c. real time OS
   d. an Ordinary OS

viii) …….. time is defined as the time period for which the execution of the process is stopped for transferring its information to the destination node.
   a. turn around

b. latency
c. freezing
d. execution

ix)) A process stack does not contain
a) Function parameters
b) Local variables
c) Return addresses
d) PID of child process

x)) What is interprocess communication?
a) communication within the process
b) communication between two process
c) communication between two threads of same process
d) none of the mentioned

2.Short Answer Type Questions :

A i)Draw a State Transition diagram of a process.
ii)Write down the advantages and disadvantages of

a)SJF Algorithm and b) Round-Robin Scheduling Algorithm ?

B..Write down short notes on

i) Context Switching

ii) PCB (Process Control Block)

iii) IPC (InterProcess Communication)

C.i)What do you know about interrupt?

ii)What do you mean by a zombie process?

iii)What is the basic difference between pre-emptive and non-pre-emptive scheduling.

1. Consider the Processes listed below .Consider the following Scheduling Algorithms .

a) FCFS b) SJF with Pre-emption c) RR (Time Slice =2 ns)

Process	Arrival Time	Burst Time (ns)
A	0	8
B	1	4
C	2	9
D	3	5

i) Draw a Gantt Chart illustrating the execution time of all the processes. (3*3=9)

ii)Find the average Turn Around Time for each of the Scheduling Algorithm ? (4)

iii)Find Average waiting Time for each of the Scheduling Algorithm ? (3)

2. Consider the Processes listed below .Consider the following Scheduling Algorithms .

a) FCFS b) SJF with Pre-emption c) RR (Time Slice =2 ns)

Process	Arrival Time	Burst Time (ns)
A	0	10
B	1	6
C	2	3
D	3	7
E	4	5

i) Draw a Gantt Chart illustrating the execution time of all the processes. (3*3=9)

ii)Find the average Turn Around Time for each of the Scheduling Algorithm ? (4)

iii)Find Average waiting Time for each of the Scheduling Algorithm ? (3)

Module 3

Process Synchronization

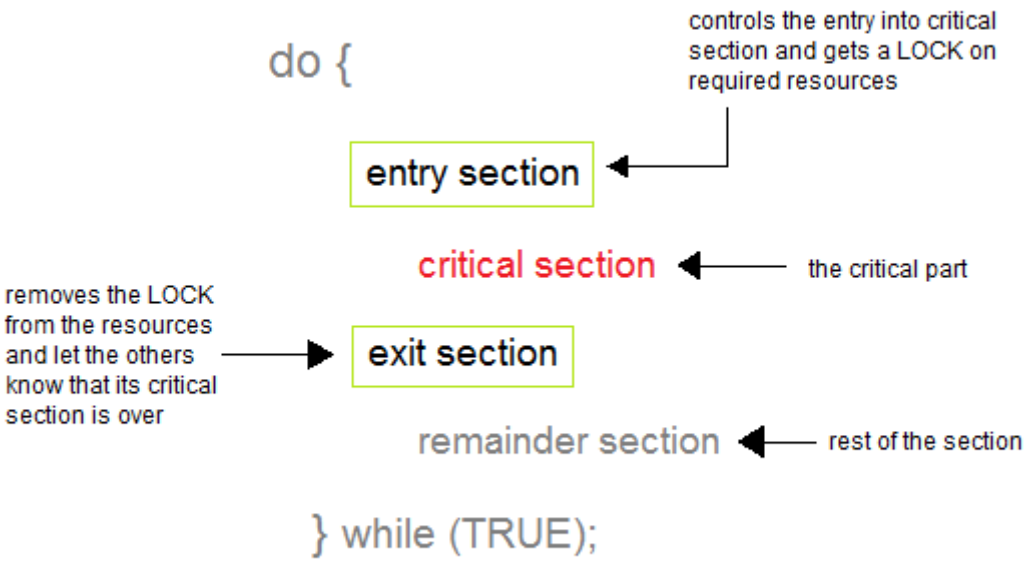
LECTURE 1

Process Synchronization means sharing system resources by processes in a such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

---

1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting

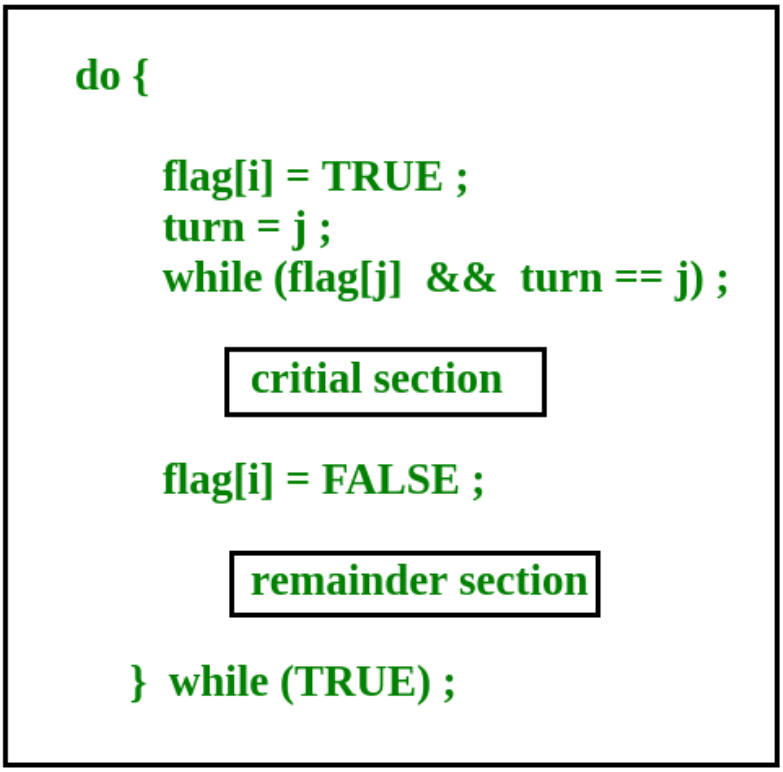
After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

LECTURE 2

Peterson's solution

Peterson's Solution is a classical software based solution to the critical section problem.
In Peterson's solution, we have two shared variables:

- boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section
- int turn : The process whose turn is to enter the critical section.



Peterson's Solution preserves all three conditions :

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not blocks other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages of Peterson's Solution

- It involves Busy waiting
- It is limited to 2 processes.

Synchronization Hardware

Test And Set

TestAndSet is a hardware solution to the synchronization problem. In TestAndSet, we have a shared lock variable which can take either of the two values, 0 or 1.
Before entering into the critical section, a process inquires about the lock. If it is locked, it keeps on waiting till it become free and if it is not locked, it takes the lock and executes the critical section.

In TestAndSet, Mutual exclusion and progress are preserved but bounded waiting cannot be preserved.

Semaphores

A Semaphore is an integer variable, which can be accessed only through two operations *wait()* and *signal()*.
There are two types of semaphores : Binary Semaphores and Counting Semaphores

- Binary Semaphores : They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.
- Counting Semaphores : They can have any value and are not restricted over a certain domain. They can be used to control access a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

Semaphore provides mutual exclusion

```
Semaphore mutex; // initialized to 1
do {
wait (mutex);
// Critical Section
signal (mutex);
// remainder section
} while (TRUE);
```

LECTURE 3

Classical problems of synchronization

The following are some classic problems of synchronization:

- The Producer–Consumer Problem (also called The Bounded Buffer Problem);
- The Readers–Writers Problem;
- The Dining Philosophers Problem.

Bounded buffer producer consumer problem

N buffers- each can hold one item

Semaphore mutex initialized to the value 1

Semaphore full initialized to the value 0

Semaphore empty initialized to the value N

The structure of the producer process

```
do {
// produce an item in nextp
wait (empty);
wait (mutex);
// add the item to the buffer
signal (mutex);
signal (full);
} while (TRUE);
```

The structure of the consumer process

```
do {
wait (full);
wait (mutex);
// remove an item from buffer to nextc
signal (mutex);
signal (empty);
// consume the item in nextc
} while (TRUE);
```

LECTURE 4

The Readers–Writers Problem

A data set is shared among a number of concurrent processes

Readers – only read the data set; they do not perform any updates

Writers – can both read and write

Problem – allow multiple readers to read at the same time

Only one single writer can access the shared data at a time

Semaphore mutex initialized to 1

Semaphore wrt initialized to 1

Integer readcount initialized to 0

The structure of a writer process

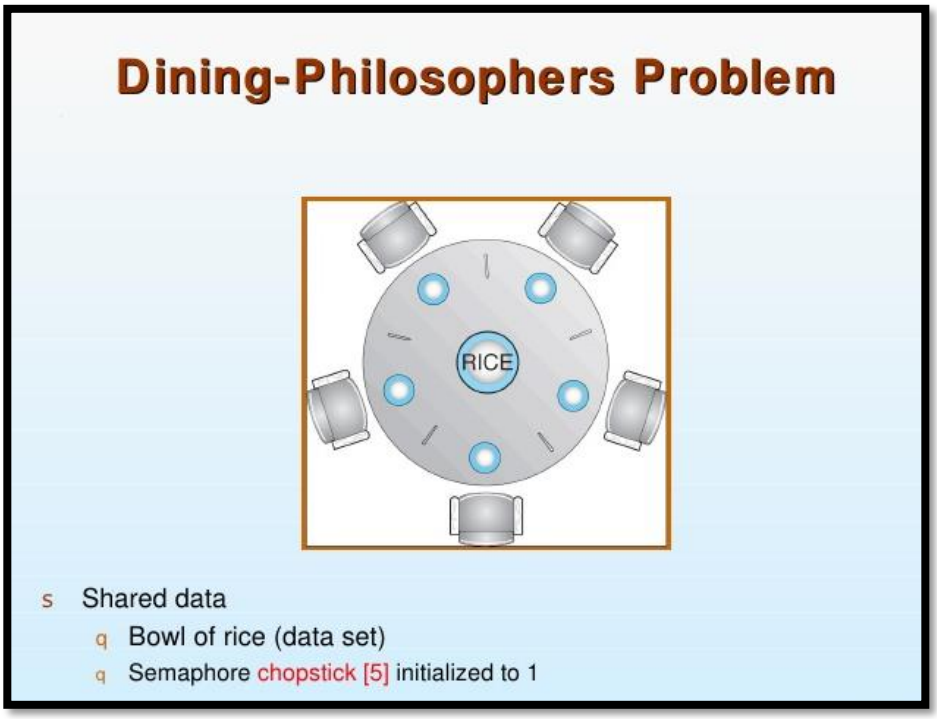
```
do {
wait (wrt) ;
// writing is performed
signal (wrt) ;
} while (TRUE);
```

The structure of a reader process

```
do {
wait (mutex) ;
readcount ++ ;
if (readcount == 1)
wait (wrt) ;
signal (mutex)
// reading is performed
wait (mutex) ;
readcount - - ;
if (readcount == 0)
signal (wrt) ;
signal (mutex) ;
} while (TRUE);
```

LECTURE 5

Dining-Philosophers Problem



Philosophers spend their lives thinking and eating. Don't interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl, need both to eat, then release both when done

In the case of 5 philosophers
Shared data
Bowl of rice (data set)
Semaphore chopstick [5] initialized to 1

The structure of Philosopher i:

```
do {
wait ( chopstick[i] );
wait ( chopStick[ (i + 1) % 5] );
// eat
signal ( chopstick[i] );
signal (chopstick[ (i + 1) % 5] );
// think
} while (TRUE);
```

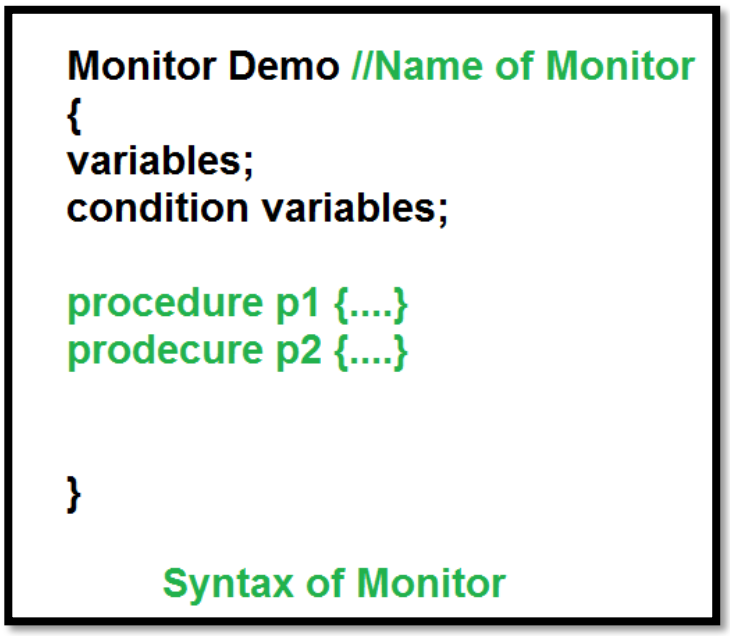
Monitor

Monitor is one of the ways to achieve Process synchronization. Monitor is supported by programming languages to achieve mutual exclusion between processes. For example Java Synchronized methods. Java provides wait() and notify() constructs.

1. It is the collection of condition variables and procedures combined together in a special kind of module or a package.

2. The processes running outside the monitor can't access the internal variable of monitor but can call procedures of the monitor.
3. Only one process at a time can execute code inside monitors.

Syntax of Monitor



Deadlock

LECTURE 6

In an operating system, a deadlock occurs when a process enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

A deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system:

1. ***Mutual exclusion:*** At least one resource must be held in a non-shareable mode. otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time.

2. ***Hold and wait* or *resource holding***: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.

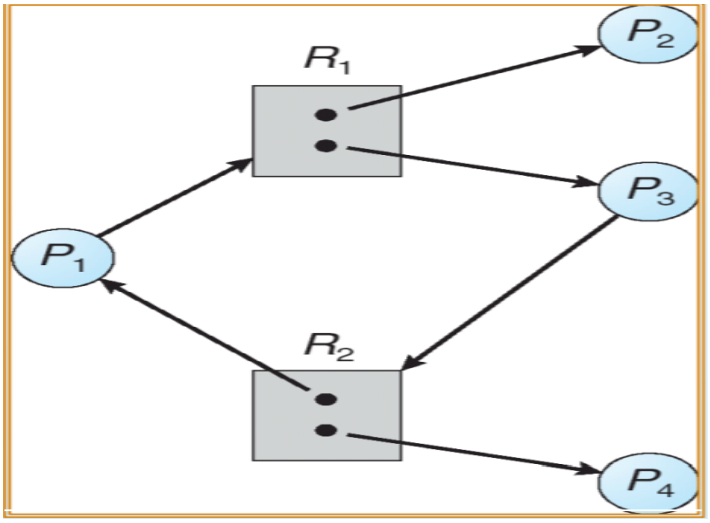
3. ***No preemption***: a resource can be released only voluntarily by the process holding it.

4. ***Circular wait:*** each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, $P = \{P_1, P_2, \ldots, P_N\}$, such that P_1 is waiting for a resource held by P_2, P_2 is waiting for a resource held by P_3 and so on until P_N is waiting for a resource held by P_1.

Resource-Allocation Graph

A set of vertices V and a set of edges E.
V is partitioned into two types:
● P = {P1, P2, …, Pn}, the set consisting of all the processes in the system.
● R = {R1, R2, …, Rm}, the set consisting of all resource types in the system.
■ request edge – directed edge P1 → Rj
■ assignment edge – directed edge Rj → Pi

Resource Allocation Graph With A Cycle But No Deadlock



Deadlock Prevention

Deadlock prevention works by preventing one of the four Coffman conditions from occurring.

- Removing the *mutual exclusion* condition means that no process will have exclusive access to a resource.
- The *hold and wait* or *resource holding* conditions may be removed by requiring processes to request all the resources they will need before starting up (or before embarking upon a particular set of operations). This advance knowledge is frequently difficult to satisfy and, in any case, is an inefficient use of resources. Another way is to require processes to request resources only when it has none. Thus, first they must release all their currently held resources before requesting all the resources they will need from scratch. This too is often impractical. It is so because resources may be allocated and remain unused for long periods. Also, a process requiring a popular resource may have to wait indefinitely, as such a resource may always be allocated to some process, resulting in resource starvation.[12] (These algorithms, such as serializing tokens, are known as the *all-or-none algorithms*.)
- The *no preemption* condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time, or the processing outcome may be inconsistent or thrashing may occur. However, inability to enforce preemption may interfere with a *priority* algorithm. Preemption of a "locked out" resource generally implies a rollback, and is to be avoided, since it is very costly in overhead. Algorithms that allow preemption include lock-free and wait-free algorithms and optimistic concurrency control. If a process holding some resources and requests for some another resource(s) that cannot be immediately allocated to it, the condition may be removed by releasing all the currently being held resources of that process.
- The final condition is the *circular wait* condition. Approaches that avoid circular waits include disabling interrupts during critical sections and using a hierarchy to determine a partial ordering of resources. If no obvious hierarchy exists, even the memory address of resources has been used to determine ordering and resources are requested in the increasing order of the enumeration. Dijkstra's solution can also be used.

LECTURE 7

Deadlock Avoidance

• Main idea

– Carefully allocate the resources such that the system will not run into deadlock

• More specifically

– Require additional information about how resources will be requested. Use this information to determine whether to grant an allocation request or to cause the requesting process to wait.

• Costs

– Run-time overhead of decision making

– Extra information required from applications

Safe and Unsafe States

A system is in a safe state if the system can allocate resources to each process (up to its maximum) in some order and let each of them compete successfully (hence, avoiding a deadlock).

• More formally

– each process declares a maximum need

– Safe state: if there exists a sequence of processes <P1,P2,...,Pn> such that for each Pi in the sequence, 1 <= i <= n, the resources that Pi can still request (i.e., Pi's maximum need - current need) can be satisfied by the currently available resources plus the resources held by all the Pj, j < i.

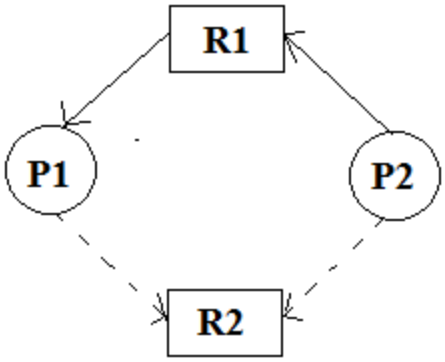
– **Unsafe state**: not safe

As you saw already, most prevention algorithms have poor resource utilization, and hence result in reduced throughputs. Instead, we can try to avoid deadlocks by making use prior knowledge about the usage of resources by processes including resources available, resources allocated, future requests and future releases by processes. Most deadlock avoidance algorithms need every process to tell in advance the maximum number of resources of each type that it may need. Based on all these info we may decide if a process should wait for a resource or not, and thus avoid chances for circular wait.

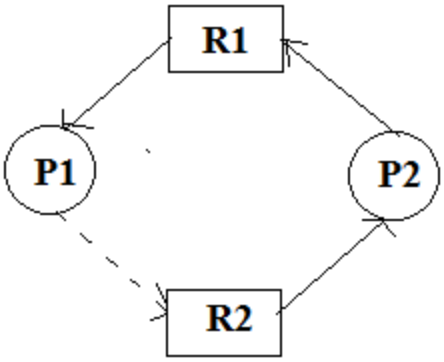
If a system is already in a safe state, we can try to stay away from an unsafe state and avoid deadlock. Deadlocks cannot be avoided in an unsafe state. A system can be considered to be in safe state if it is not in a state of deadlock and can allocate resources upto the maximum available. A safe sequence of processes and allocation of resources ensures a safe state. Deadlock avoidance algorithms try not to allocate resources to a process if it will make the system in an unsafe state. Since resource allocation is not done right away in some cases, deadlock avoidance algorithms also suffer from low resource utilization problem.

A resource allocation graph is generally used to avoid deadlocks. If there are no cycles in the resource allocation graph, then there are no deadlocks. If there are cycles, there may be a deadlock. If there is only one instance of every resource, then a cycle implies a deadlock. Vertices of the resource allocation graph are resources and processes. The resource allocation graph has request edges and assignment edges. An edge from a process to resource is a request edge and an edge from a resource to process is an allocation edge. A claim edge denotes that a request may be made in future and is represented as a dashed line. Based on claim edges we can see if there is a chance for a cycle and then grant requests if the system will again be in a safe state.

Consider the image with claim edges as below:



If R2 is allocated to p2 and if P1 request for R2, there will be a deadlock.



The resource allocation graph is not useful if there are multiple instances for a resource. In such a case, we can use Banker's algorithm. In this algorithm, every process must tell upfront the maximum resource of each type it need, subject to the maximum available instances for each type. Allocation of resources is made only, if the allocation ensures a safe state; else the processes need to wait. The Banker's algorithm can be divided into two parts: Safety algorithm if a system is in a safe state or not. The resource request algorithm make an assumption of allocation and see if the system will be in a safe state. If the new state is unsafe, the resources are not allocated and the data structures are restored to their previous state; in this case the processes must wait for the resource.

Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following **Data structures** are used to implement the Banker's Algorithm:
Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently allocated **'k'** instances of resource type $\mathbf{R_j}$
- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.
Banker's algorithm consist of Safety algorithm and Resource request algorithm

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize:Work=Available
Finish[i] = false; for i=1, 2, 3, 4….n
2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work if no such i exists goto step (4)
3) Work = Work + Allocation
Finish[i] = true
goto step (2)
4) if finish [i] = true for all i
then the system is in a safe state

LECTURE 8

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i[j] = k$ means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1) If $Request_i <= Need_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i <= Available$ Goto step (3); otherwise, P_i must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

Available = Available – Requesti
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i – Request_i$

Example:
Considering a system with five processes P_0 through P_4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

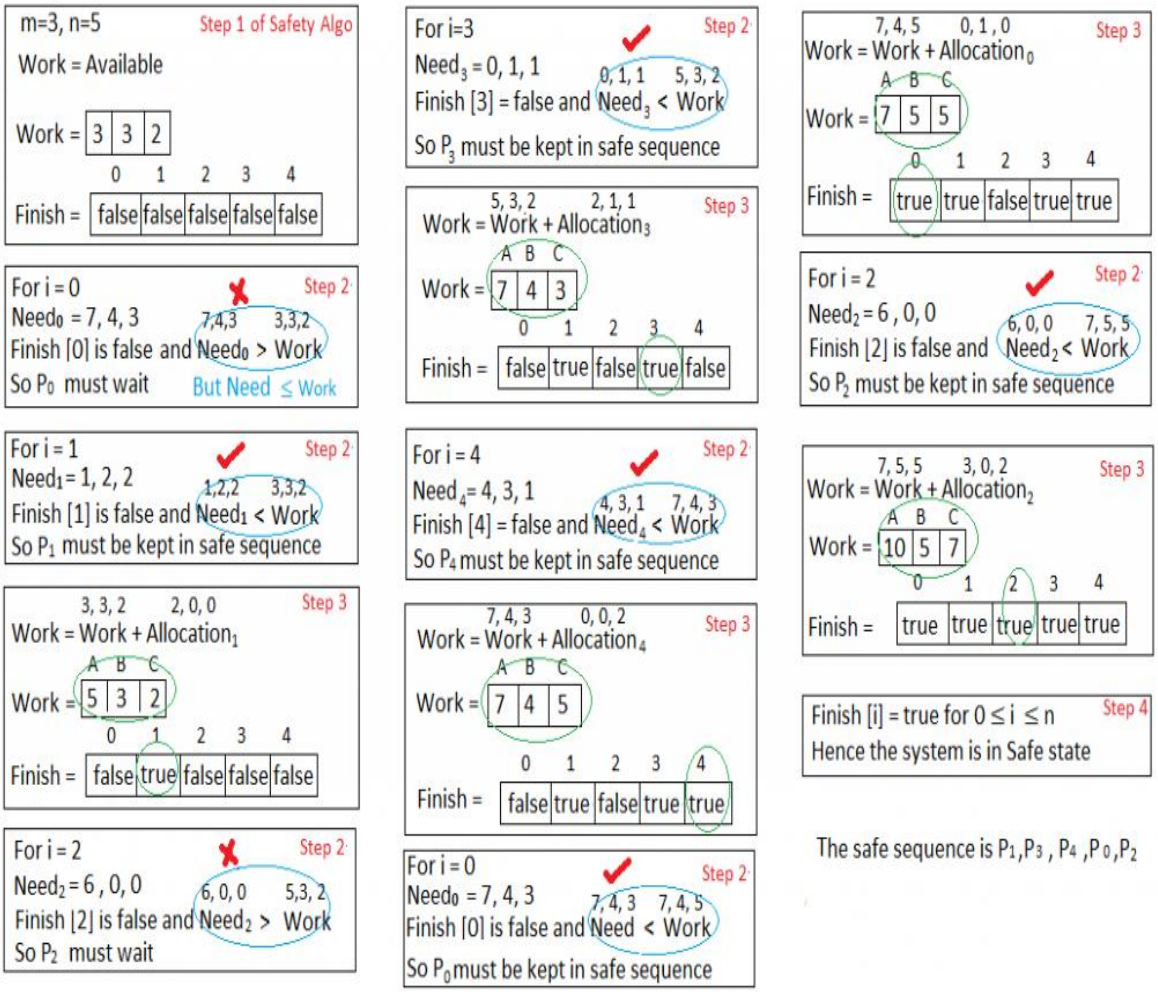
Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Question1. What will be the content of the Need matrix?
Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Question2. Is the system in safe state? If Yes, then what is the safe sequence?
Applying the Safety algorithm on the given system,



Question3. What will happen if process P_1 requests one additional instance of resource type A and two instances of resource type C?

A B C

$Request_1$ = 1, 0, 2

To decide whether the request is granted we use Resource Request algorithm

Step 1

1, 0, 2 1, 2, 2

$Request_1 < Need_1$

Step 2

1, 0, 2 3, 3, 2

$Request_1 < Available$

Step 3

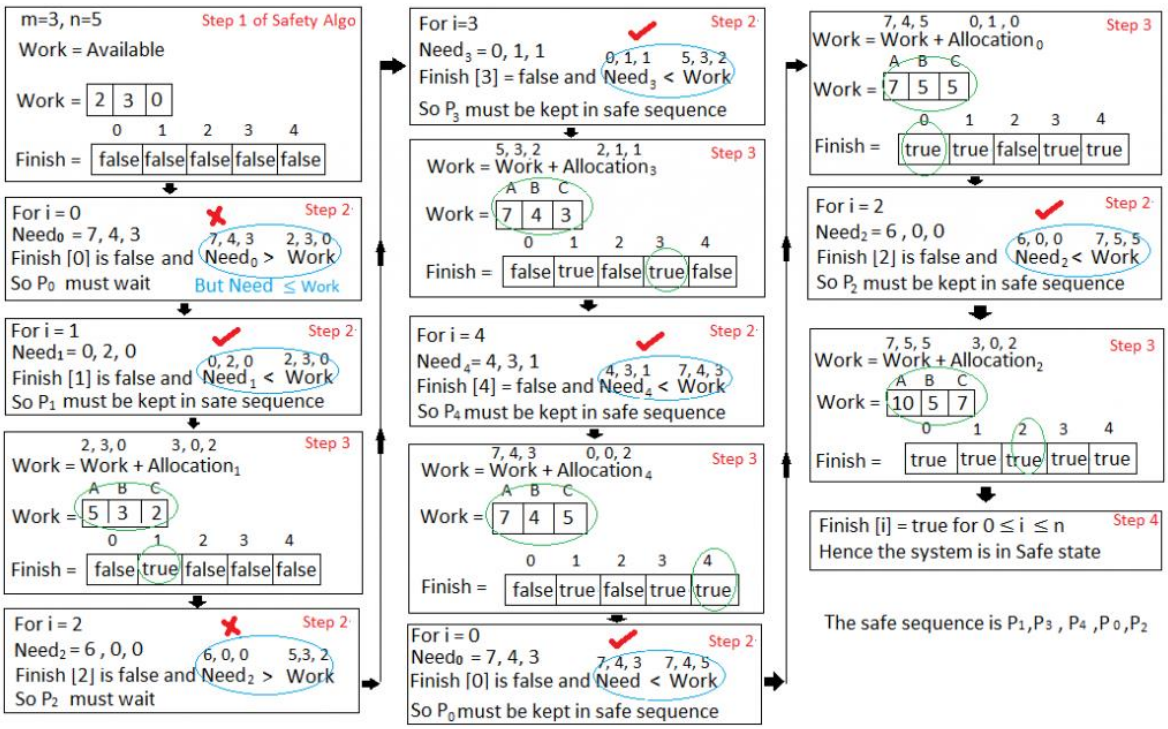
$Available = Available - Request_1$

$Allocation_1 = Allocation_1 + Request_1$

$Need_1 = Need_1 - Request_1$

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



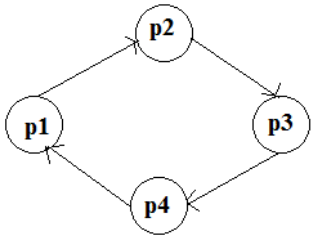
Hence the new system state is safe, so we can immediately grant the request for process $\mathbf{P_1}$.

LECTURE 9

Deadlock Detection

If deadlock prevention and avoidance are not done properly, as deadlock may occur and only things left to do is to detect the recover from the deadlock.

If all resource types has only single instance, then we can use a graph called wait-for-graph, which is a variant of resource allocation graph. Here, vertices represent processes and a directed edge from P1 to P2 indicate that P1 is waiting for a resource held by P2. Like in the case of resource allocation graph, a cycle in a wait-for-graph indicate a deadlock. So the system can maintain a wait-for-graph and check for cycles periodically to detect any deadlocks.



The wait-for-graph is not much useful if there are multiple instances for a resource, as a cycle may not imply a deadlock. In such a case, we can use an algorithm similar to Banker's algorithm to detect deadlock. We can see if further allocations can be made on not based on current allocations.

Single Instance of Each Resource Type

- Maintain wait-for graph
- Nodes are processes.

1. Periodically invoke an algorithm that searches for a cycle in the graph.
2. An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Deadlock Recovery

Once a deadlock is detected, you will have to break the deadlock. It can be done through different ways, including, aborting one or more processes to break the circular wait condition causing the deadlock and preempting resources from one or more processes which are deadlocked.

- **Process Termination**
  - Abort all deadlocked processes:
    - Fast
    - A lot of process work is lost.
  - Abort one deadlocked process at a time and check for deadlocks again:
    - More work to resolve a deadlock.
    - Better in terms of process work.
    - What is a good order to abort processes?

- **Resource Preemption**
  - what is a good way to select a victim
  - How can we rollback and then recover from preemption?
  - How can we protect from starvation

Questions

A. Multiple choice questions.

1. A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which access takes place is called :
   a) data consistency
   b) race condition
   c) aging
   d) starvation
2. For Mutual exclusion to prevail in the system :
a) at least one resource must be held in a non sharable mode
b) the processor must be a uniprocessor rather than a multiprocessor
c) there must be at least one resource in a sharable mode
d) all of the mentioned
3. The segment of code in which the process may change common variables, update tables, write into files is known as :
a) program
b) critical section
c) non – critical section
d) synchronizing
4. The following three conditions must be satisfied to solve the critical section problem :
a) Mutual Exclusion
b) Progress
c) Bounded Waiting
d) All of the mentioned
5. Mutual exclusion implies that :
a) if a process is executing in its critical section, then no other process must be executing in their critical sections
b) if a process is executing in its critical section, then other processes must be executing in their critical sections
c) if a process is executing in its critical section, then all the resources of the system must be blocked until it finishes execution
d) none of the mentioned
6. Bounded waiting implies that there exists a bound on the number of times a process is allowed to enter its critical section :
a) after a process has made a request to enter its critical section and before the request is granted
b) when another process is in its critical section
c) before a process has made a request to enter its critical section
d) none of the mentioned
7. A minimum of _____ variable(s) is/are required to be shared between processes to solve the critical section problem.
a) one
b) two
c) three
d) four

8. In the bakery algorithm to solve the critical section problem :
a) each process is put into a queue and picked up in an ordered manner
b) each process receives a number (may or may not be unique) and the one with the lowest number is served next
c) each process gets a unique number and the one with the highest number is served next
d) each process gets a unique number and the one with the lowest number is served next
9. Semaphore is a/an _____ to solve the critical section problem.
a) hardware for a system
b) special program for a system
c) integer variable
d) none of the mentioned
10. The two atomic operations permissible on semaphores are :
a) wait
b) stop
c) hold
d) none of the mentioned

11. In the bounded buffer problem, there are the empty and full semaphores that :
a) count the number of empty and full buffers
b) count the number of empty and full memory spaces
c) count the number of empty and full queues
d) none of the mentioned

12. A mutex :
a) is a binary mutex
b) must be accessed from only one process
c) can be accessed from multiple processes
d) None of the mentioned

13. For a deadlock to arise, which of the following conditions must hold simultaneously ?
a) Mutual exclusion
b) No preemption
c) Hold and wait
d) All of the mentioned
14. For a Hold and wait condition to prevail :
a) A process must be not be holding a resource, but waiting for one to be freed, and then request to acquire it
b) A process must be holding at least one resource and waiting to acquire additional resources that are being held by other processes
c) A process must hold at least one resource and not be waiting to acquire additional resources
d) None of the mentioned

B. Short answer type question.

1. Define Mutual Exclusion.
2. What is meant by Binary Semaphore?
3. What is meant by Binary Semaphore?
4. What does a solution for Critical Section Problem must satisfy?
5. Discuss in detail semaphores.
6. What is deadlock?
7. What are the necessary condition for deadlock?

C. Long answer type question.

1. Discuss in detail deadlock and its prevention mechanism.
2. Explain the Banker's algorithm for deadlock avoidance.
3. Discuss in detail the methods involved in the detection and recovery of deadlock.
4. What is semaphore? Write a solution for bounded buffer producer consumer problem.
5. Write a solution for Reader’s writer’s problem.
6. Write a solution for Dining philosopher’s problem.
7. What is critical section problem ? How it can be solved?

Module 4
Memory Management
LECTURE 4

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Basic concepts related to Memory Management

Process Address Space

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated −

S.N.	Memory Addresses & Description
1	**Symbolic addresses** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses** At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	**Physical addresses** The loader generates these addresses at the time when a program is loaded into main memory.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Static vs Dynamic Loading

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

Static vs Dynamic Linking

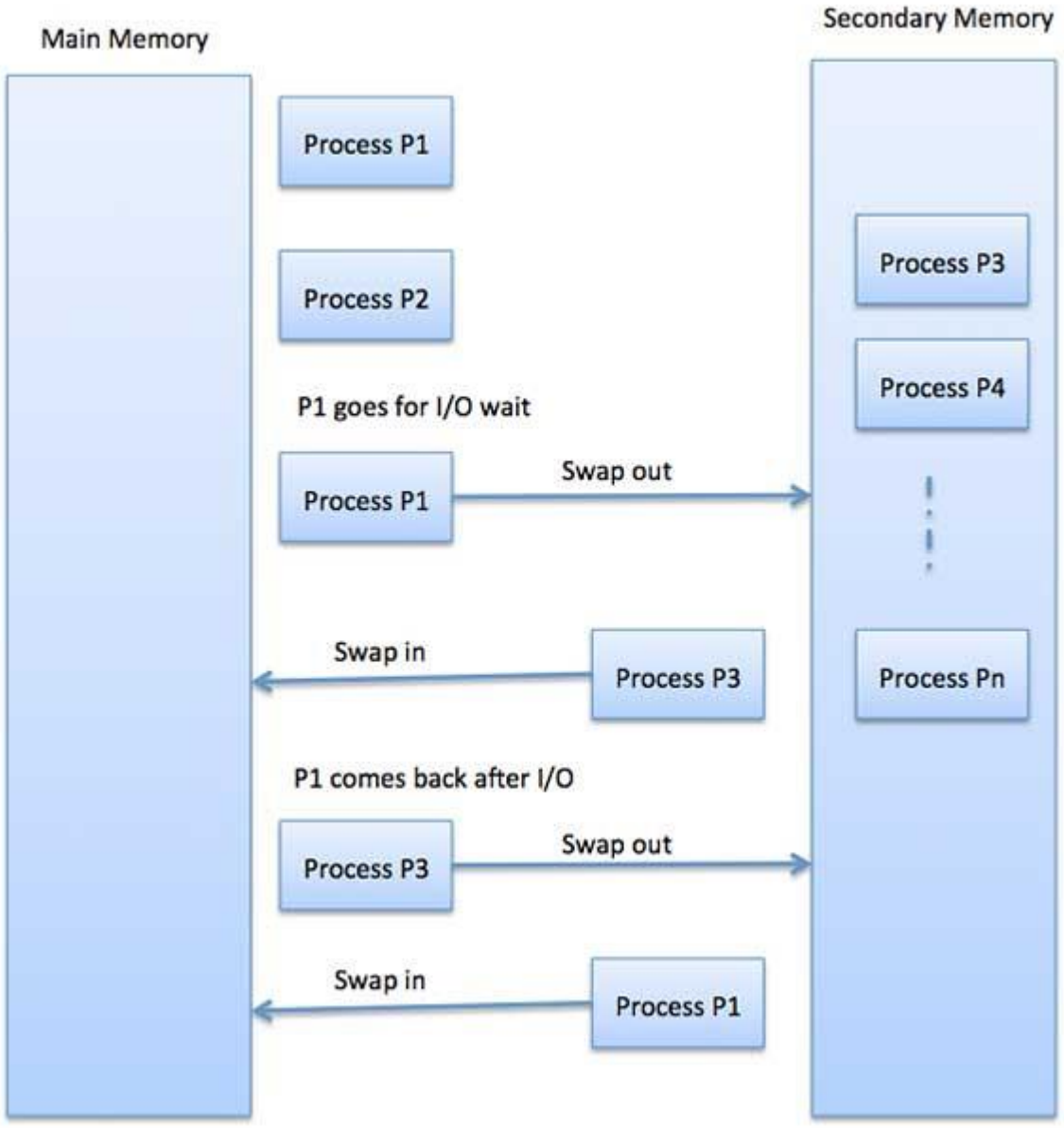
As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

2048KB / 1024KB per second

= 2 seconds

= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Memory Allocation

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

In **Partition Allocation**, when there are more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.
Below are the various partition allocation schemes :

1. **First Fit**: In the first fit, partition is allocated which is first sufficient from the top of Main Memory.

2. **Best Fit** Allocate the process to the partition which is first smallest sufficient partition among the free available partition.

3. **Worst Fit** Allocate the process to the partition which is largest sufficient among the freely available partitions available in the main memory.

4. **Next Fit** Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Contiguous memory allocation

Memory is a large array of bytes, where each byte has its own address. The memory allocation can be classified into two methods contiguous memory allocation and non-contiguous memory allocation. The major difference between Contiguous and Noncontiguous memory allocation is that the **contiguous memory allocation** assigns the consecutive blocks of memory to a process requesting for memory whereas, the **noncontiguous memory allocation** assigns the separate memory blocks at the different location in memory space in a nonconsecutive manner to a process requesting for memory.

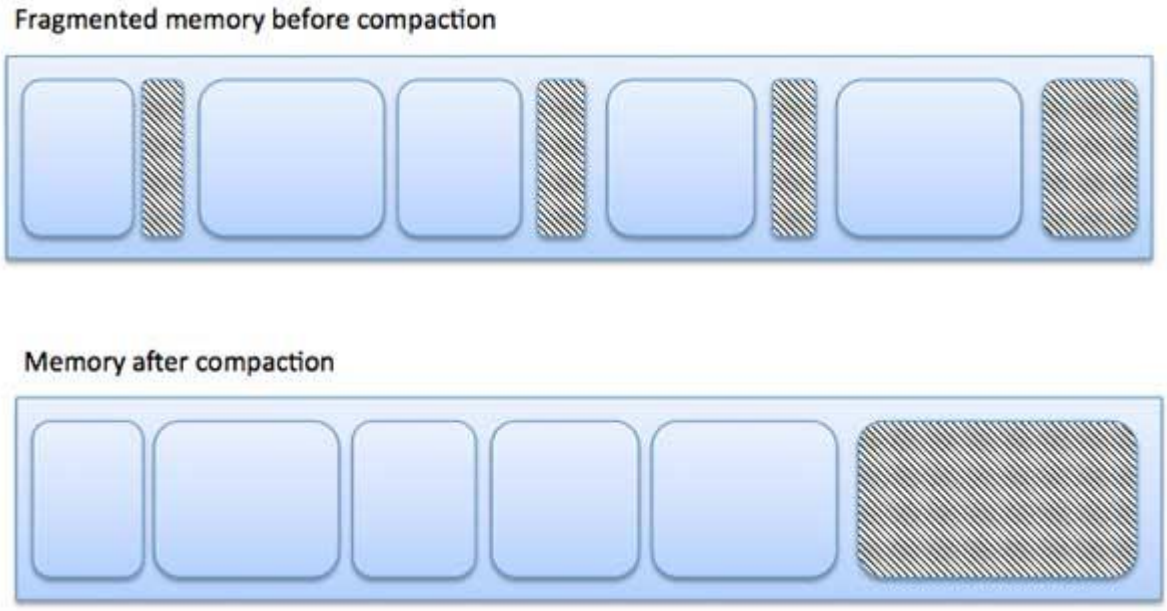
Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

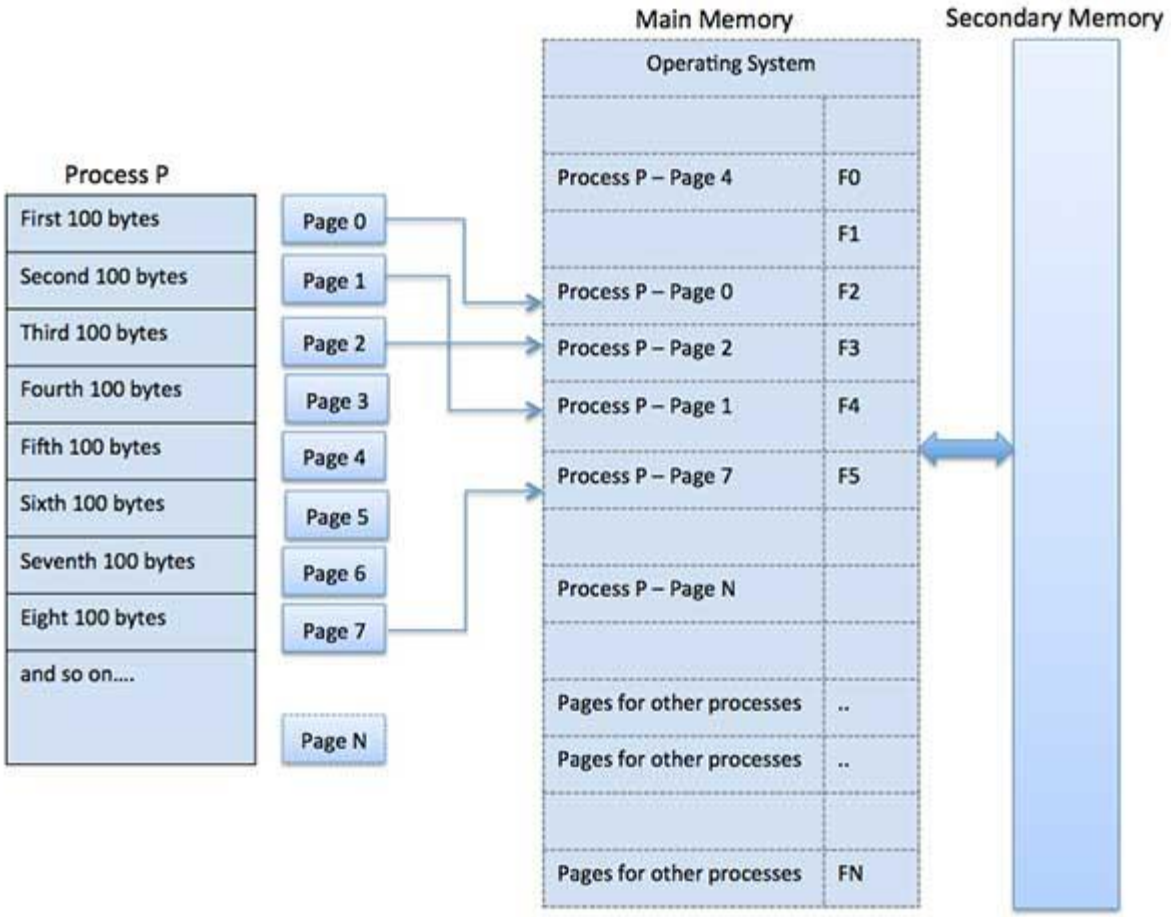
The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

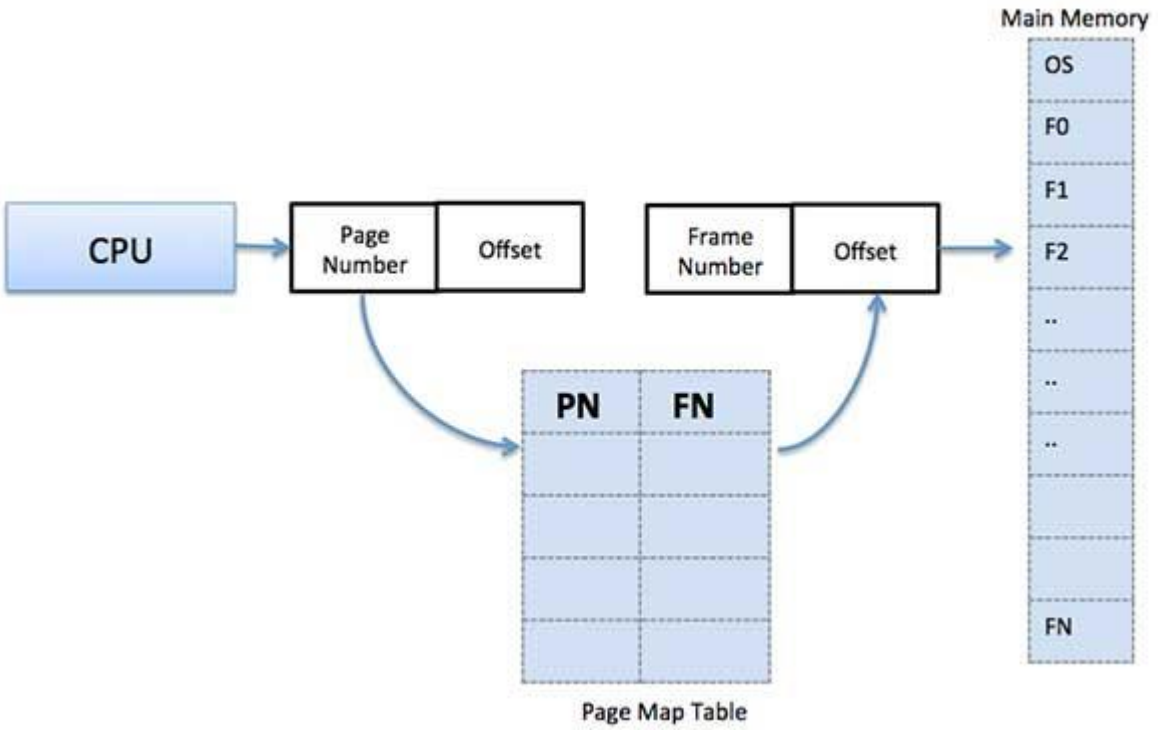
Page address is called **logical address** and represented by **page number**and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

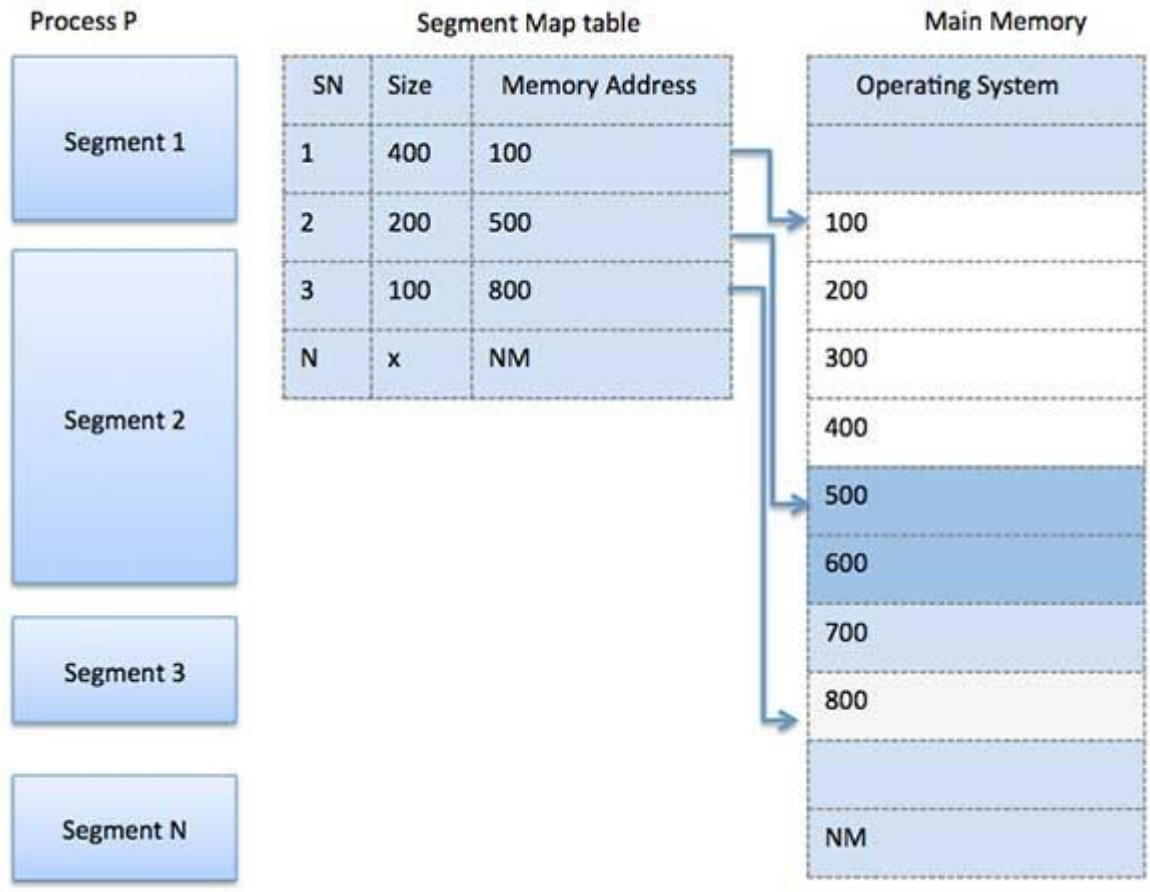
Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



Address translation through segmentation

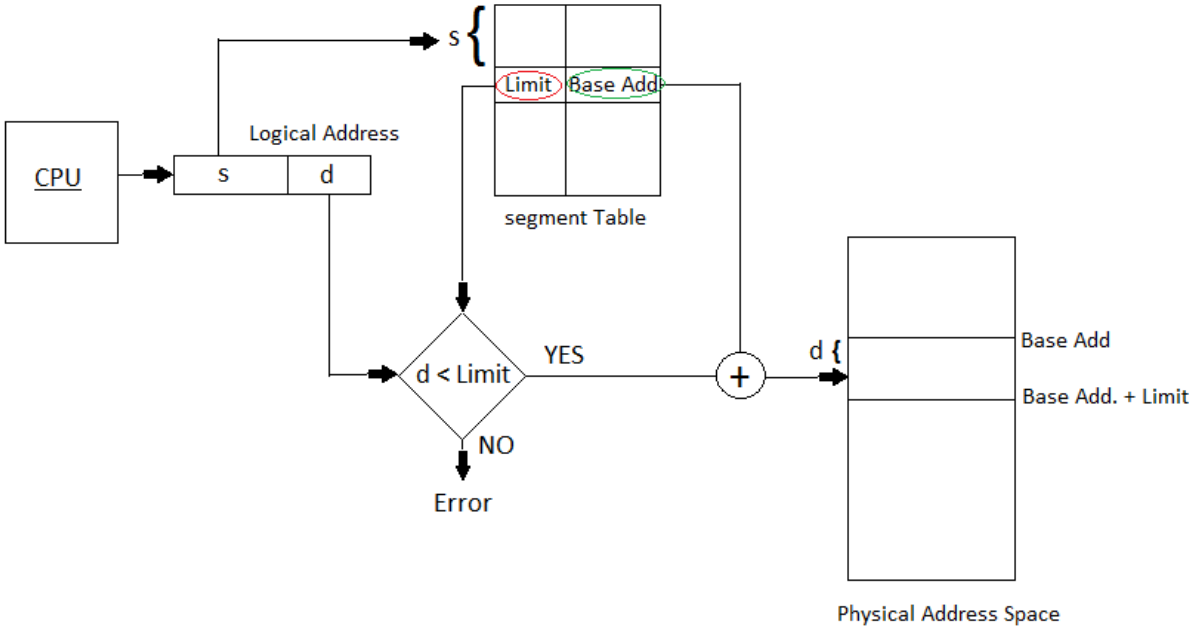


Fig. Address translation through segmentation

Translation lookaside buffer (TLB)

A translation lookaside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

When a virtual memory address is referenced by a program, the search starts in the CPU. First, instruction caches are checked. If the required memory is not in these very fast caches, the system has to look up the memory's physical address. At this point, TLB is checked for a quick reference to the location in physical memory.

When an address is searched in the TLB and not found, the physical memory must be searched with a memory page crawl operation. As virtual memory addresses are translated, values referenced are added to TLB. When a value can be retrieved from TLB, speed is enhanced because the memory address is stored in the TLB on processor. Most processors include TLBs to increase the speed of virtual memory operations through the inherent latency-reducing proximity as well as the high-running frequencies of current CPU's.

TLBs also add the support required for multi-user computers to keep memory separate, by having a user and a supervisor mode as well as using permissions on read and write bits to enable sharing.

TLBs can suffer performance issues from multitasking and code errors. This performance degradation is called a cache thrash. Cache thrash is caused by an ongoing computer activity that fails to progress due to excessive use of resources or conflicts in the caching system.

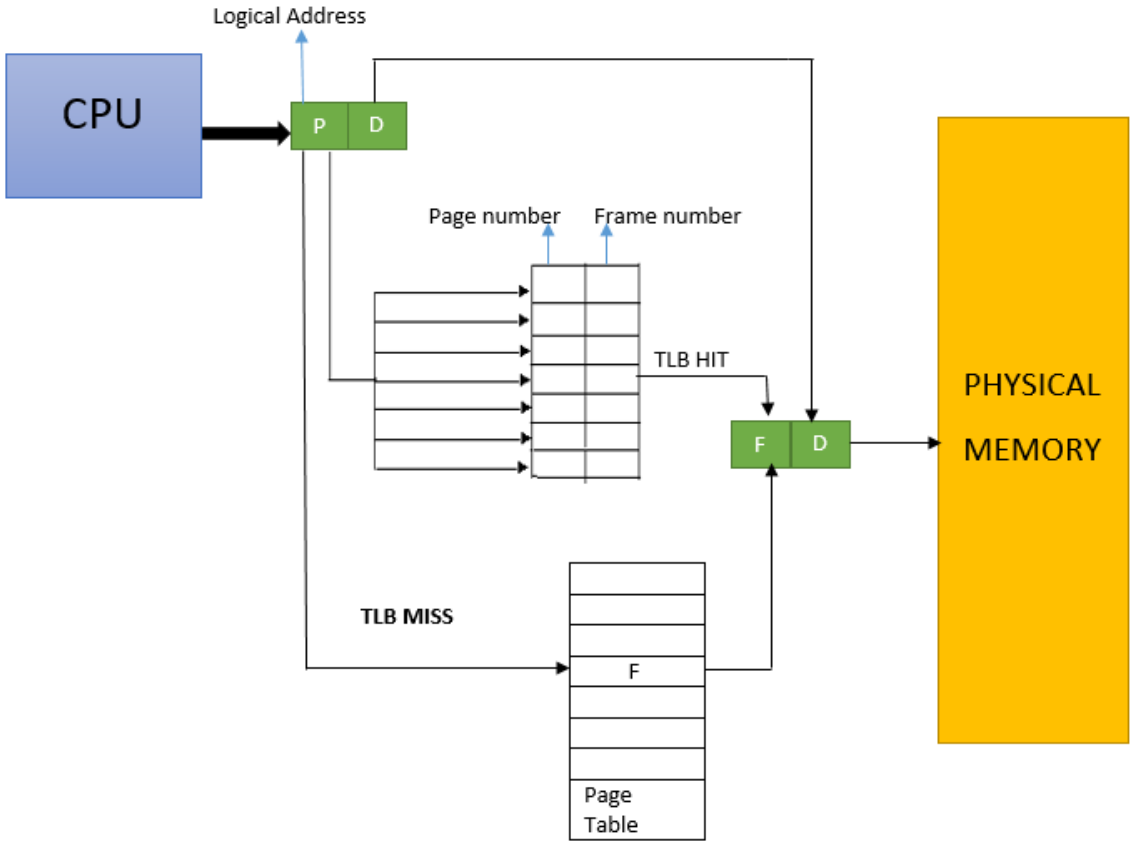


Fig. TLB

Questions

- Multiple choice questions.

1. CPU fetches the instruction from memory according to the value of
   a) program counter
   b) status register
   c) instruction register
   d) program status word
2. Which one of the following is the address generated by CPU?
   a) physical address
   b) absolute address
   c) logical address
   d) none of the mentioned

3. 4. Run time mapping from virtual to physical address is done by
   a) Memory management unit
   b) CPU
   c) PCI
   d) None of the mentioned

4. 5. Memory management technique in which system stores and retrieves data from secondary storage for use in main memory is called
   a) fragmentation
   b) paging
   c) mapping
   d) none of the mentioned
5. What is compaction?
   a) a technique for overcoming internal fragmentation
   b) a paging technique
   c) a technique for overcoming external fragmentation
   d) a technique for overcoming fatal error
6. Physical memory is broken into fixed-sized blocks called _____

   a) frames
   b) pages
   c) backing store

d) none of the mentioned

7. 2. Logical memory is broken into blocks of the same size called _____
   a) frames
   b) pages
   c) backing store
   d) none of the mentioned

8. 3. Every address generated by the CPU is divided into two parts :
   a) frame bit & page number
   b) page number & page offset
   c) page offset & frame bit
   d) frame offset & page offset
9. The size of a page is typically :
   a) varied
   b) power of 2
   c) power of 4
   d) none of the mentioned
10. The operating system maintains a _____ table that keeps track of how many frames have been allocated, how many are there, and how many are available.
    a) page
    b) mapping
    c) frame
    d) memory

- Short answer type question.

1. What is meant by External Fragmentation and Internal Fragmentation?
2. What is meant by Paging? Give its advantages?
3. What is meant by Swapping?
4. What is meant by Memory Compaction?

- Long answer type question.

1. Explain paging technique in detail.
2. Explain Segmentation technique in detail.
3. How TLB can be used in Paging?
4. What do you mean by external and internal fragmentation?
5. What is compaction? Why it is used?

MODULE: 4

VIRTUAL MEMORY

LECTURE: 1

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of main memory such that it occupies different places in main memory at different times during the course of execution.
2. A process may be broken into number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time addres translation and use of page or segment table permits this

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below –

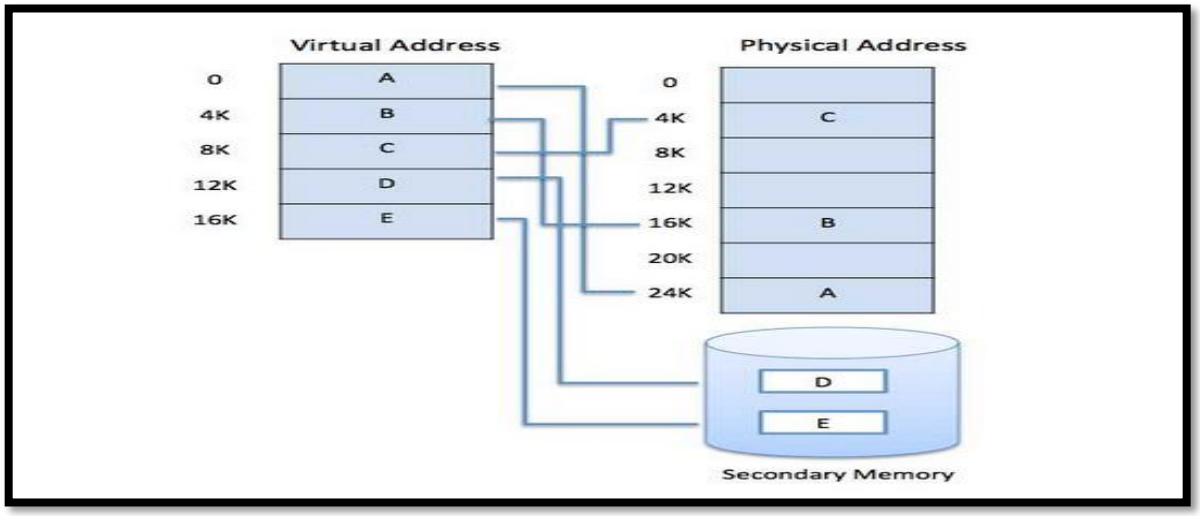


Fig: The MMU's job is to translate virtual addresses into physical addresses

Demand paging:

Virtual memory is commonly implemented by **demand paging.** It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

The **Demand Paging** is also same with the Simple Paging. But the Main Difference is that in the Demand Paging Swapping is used. Means all the Pages will be in and out from the Memory when they are required. When we specify a Process for the Execution then the Processes is stored firstly on the Secondary Memory which is also known as the Hard Disk.

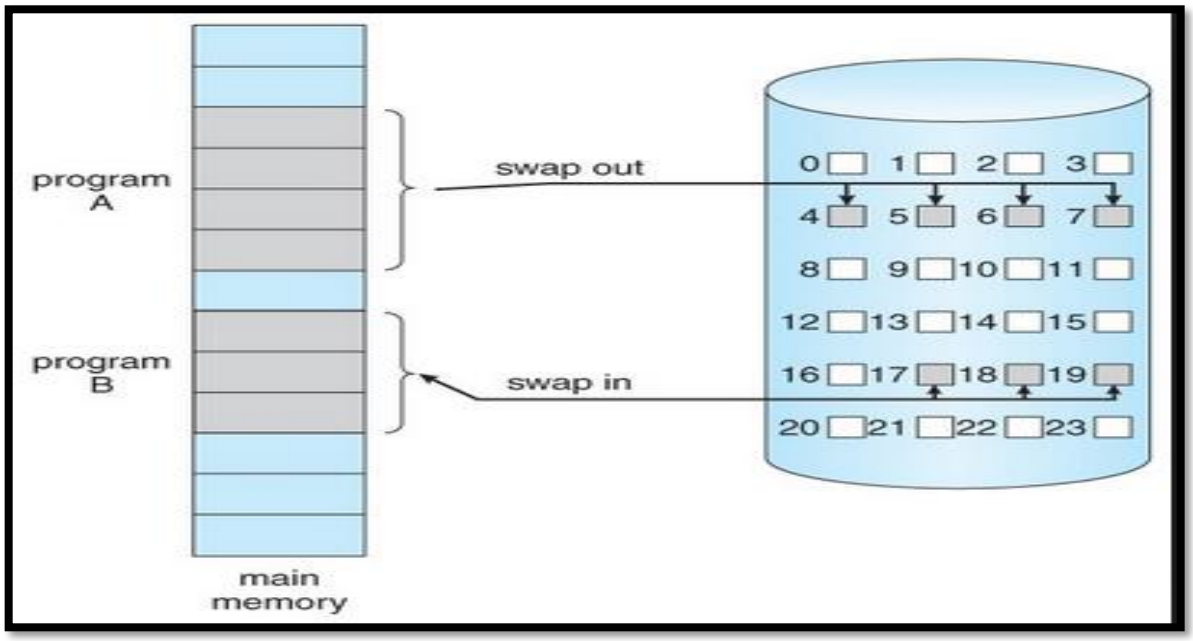


Fig: Swapping

But when they are required then they are Swapped Backed into the Memory and when a Process is not used by the user then they are Temporary Swapped out from the Memory. Means they are Stored on the Disk and after that they are Copied into the Memory.

So Demand Paging is the Concept in which a Process is Copied into the Logical Memory from the Physical Memory when we needs them. A Process can load either Entire, Copied into the Main Memory or the part of single Process is copied into the Memory so that is only the single Part of the Process is copied into the Memory then this is also called as the **Lazy Swapping**.

For Swapping the Process from the Main Memory or from the Physical Memory, a Page Table must be used. The Page Table is used for Storing the Entries which Contains the Page or Process Number and also the offset Number which indicates the address of the Process where a Process is Stored and there will also be the Special or Extra Bit which is also Known as the Flag Bit which indicates whether the Page is Stored into the Physical Memory.

The Page Table Contains two Entries those are used as valid and invalid means whether the Process is Stored into the Page Table. Or Whether the Demand Program is Stored into the Physical Memory So that they can be easily swapped. If the Requested Program is not stored into the Page Table then the Page Table must Contains the Entries as v and I means valid and invalid along the Page Number.

When a user Request for any Operation then the Operating System performs the following instructions:-

1) First of all this will fetch all the instructions from the Physical Memory into the Logical Memory.
2) Decode all the instructions means this will find out which Operation has to be performed on the instructions.
3) Perform Requested Operation.
4) Stores the Result into the Logical Memory and if needed the Results will be Stored into the Physical Memory.

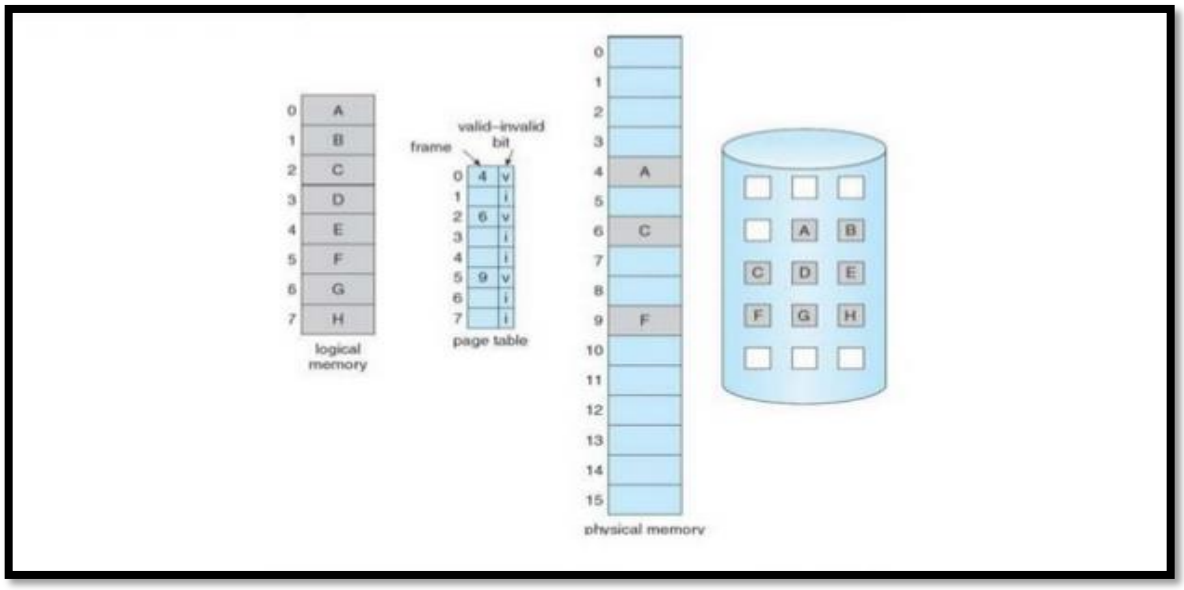


Fig: Demand paging Example

Advantages

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

LECTURE: 2

PAGE FAULTS

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

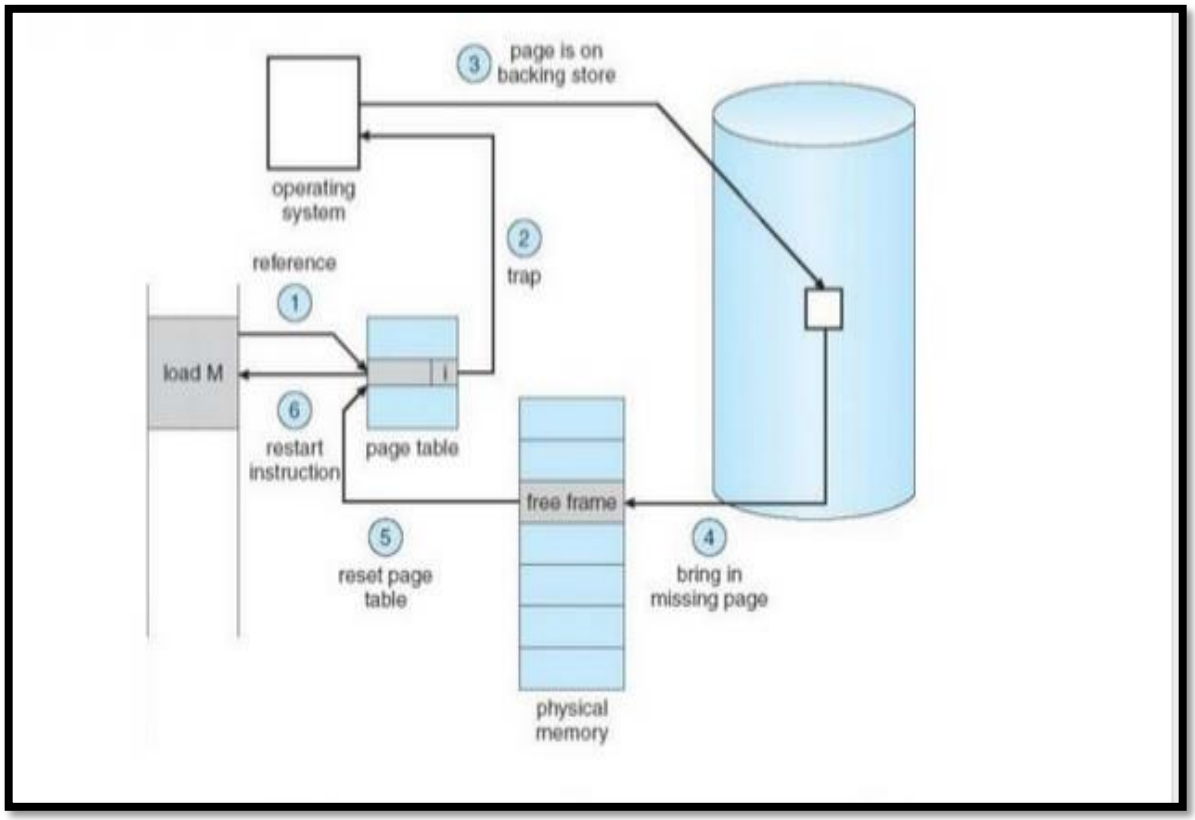


Fig: Page faults

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
5. The page table will updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.

Pure Demand paging:

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

Demand Paging.	**Pure Demand Paging**.
In demand paging, a page is not loaded into main memory until it is needed	In pure demand paging, even a single page is not loaded into memory initially. Hence pure demand paging causes a page fault. Page fault, the situation in which the page is not available whenever a processor needs to execute it.
In Demand paging follows that pages should only be brought into memory if the executing process demands them	while in pure demand paging swapping, where all memory for a process is swapped from secondary storage to main memory during the process startup.

PERFORMANCE OF DEMAND PAGING

LECTURE: 3

1. Effective Access Time (EAT) for a demand-paged memory.
2. Memory Access Time (ma) for most computers now ranges from 10 to 200 nanoseconds.
3. If there is no page fault, then EAT = ma.
4. If there is page fault, then
5. EAT = (1 – p) x (ma) + p x (page-fault time).

p: the probability of a page fault ($0 \leq p \leq 1$),
we expect p to be close to zero (a few page faults).

6. If p=0 then no page faults, but if p=1 then every reference is a fault. If a page fault occurs, we must first read the relevant page from disk, and then access the desired word.
7. We are faced with three major components of the page-fault service time:
a. Service the page-fault interrupt.
b. Read in the page.
c. Restart the process.
8.A typical hard disk has: An average latency of 8 milliseconds. A seek of 15 milliseconds. A transfer time of 1 milliseconds. Total paging time = (8+15+1)= 24 milliseconds, including hardware and software time, but no queuing (wait) time.

Example 1:

Assume an average page-fault service time of 25 milliseconds (10-3), and a Memory Access Time of 100 nanoseconds (10-9). Find the Effective Access Time?

•Solution: Effective Access Time (EAT)
= (1 – p) x (ma) + p x (page fault time)
= (1 – p) x 100 + p x 25,000,000
= 100 – 100 x p + 25,000,000 x p
= 100 + 24,999,900 x p.

•Note: The Effective Access Time is directly proportional to the page-fault rate.

PAGE REPLACEMENT ALGORITHM
LECTURE: 3

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

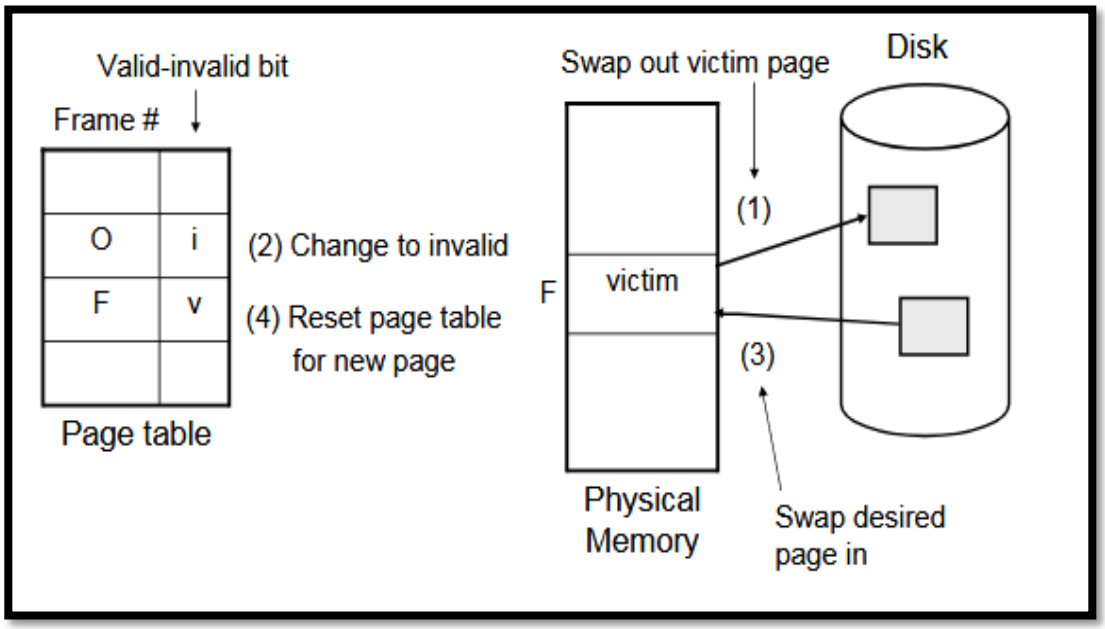


Fig: Page Replacement Algorithm

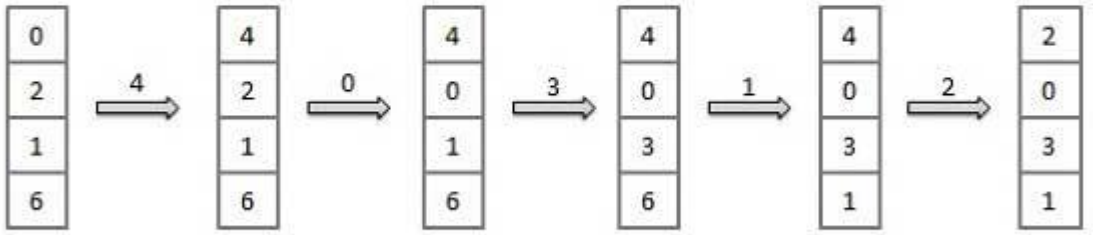
The page-fault service time is now modified to include page replacement:
1. Find the location of the desired page on the disk.
2. Find a free frame:
   –If there is a free frame use it.
   –Otherwise, use a page-replacement algorithm to
   select a victim frame.
   –Write the victim page to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly free frame;
change the page and frame tables.
4. Restart the user process

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



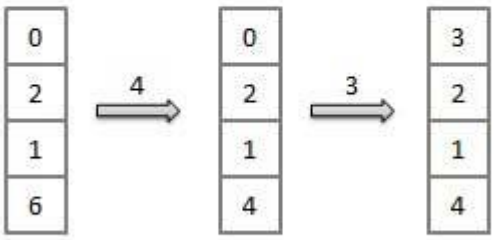
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



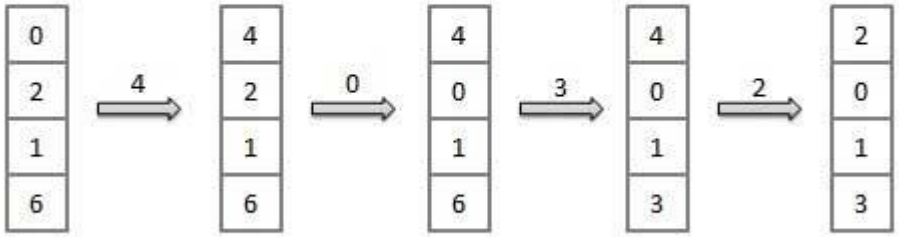
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x

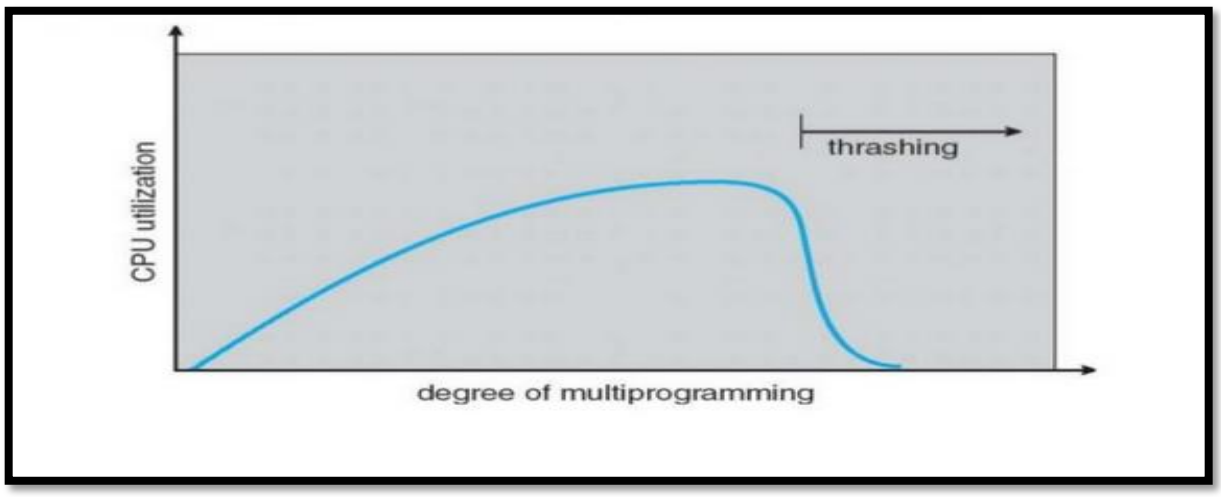


Fault Rate = 8 / 12 = 0.67

THRASHING

LECTURE: 3

Too much of this leads to a condition called Thrashing. The system spends most of its time swapping pages rather than executing instructions.



Causes of Thrashing :

1. **High degree of multiprogramming** : If the number of processes keeps on increasing in the memory than number of frames allocated to each process will be decreased. So, less

number of frames will be available to each process. Due to this, page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.

For example:
Let free frames = 400
Case 1: Number of process = 100
Then, each process will get 4 frames.

Case 2: Number of process = 400
Each process will get 1 frame.
Case 2 is a condition of thrashing, as the number of processes are increased,frames per process are decreased. Hence CPU time will be consumed in just swapping pages.

2. **Lacks of Frames**:If a process has less number of frames then less pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

Recovery of Thrashing :

- Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
- If the system is already in thrashing then instruct the mid term schedular to suspend some of the processes so that we can recover the system from thrashing.

References:

[1] http://www.dauniv.ac.in/downloads/CArch_PPTs/CompArchCh12L01MultProcArch.pdf

[2] https://www.tutorialspoint.com

[3] www.geeksforgeeks.org/operating-system-page-replacement-algorithm

[4] http://www2.latech.edu/~box/os/ch08.pdf

[5] http://www.eecg.toronto.edu/~jacobsen/os/2007s/memory.pdf

MCQ Questions:

1. Because of virtual memory, the memory can be shared among
   a) processes
   b) threads
   c) instructions
   d) none of the mentioned
   2. _____ is the concept in which a process is copied into main memory from the secondary memory according to the requirement.
   a) Paging
   b) Demand paging
   c) Segmentation

d) Swapping
3. The pager concerns with the
a) individual page of a process
b) entire process
c) entire thread
d) first page of a process
4. Swap space exists in
a) primary memory
b) secondary memory
c) cpu
d) none of the mentioned
5. When a program tries to access a page that is mapped in address space but not loaded in physical memory, then
a) segmentation fault occurs
b) fatal error occurs
c) page fault occurs
d) no error occurs
6. Effective access time is directly proportional to
a) page-fault rate
b) hit ratio
c) memory access time
d) none of the mentioned
7. In FIFO page replacement algorithm, when a page must be replaced
a) oldest page is chosen
b) newest page is chosen
c) random page is chosen
d) none of the mentioned
8. Which algorithm chooses the page that has not been used for the longest period of time whenever the page required to be replaced?
a) first in first out algorithm
b) additional reference bit algorithm
c) least recently used algorithm
d) counting based page replacement algorithm
9. A process is thrashing if
a) it is spending more time paging than executing
b) it is spending less time paging than executing
c) page fault occurs
d) swapping cannot take place
10. Working set model for page replacement is based on the assumption of
a) modularity
b) locality
c) globalization
d) random access

Short Answer Type Questions:

2. What is the need of Virtual Memory?
3. Write down the advantages and disadvantages of virtual memory?
4. What is lazy swapper?
5. What is swap space?
6. What is page fault?
7. What do you mean by Pure Demand paging?
8. Differentiate between: Demand Paging VS Pure Demand paging
9. Why page replacement algorithm important?
10. What do mean by Belady's anomaly?
11. What is thrashing?

Assignment:

1. Explain how do you implement virtual memory by demand paging? What is pure demand paging?
2. Explain how a page fault occurs with diagram.
3. What is thrashing? Why thrashing occurs? How do you recover from Thrashing?
4. Consider the following page reference string:

   1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

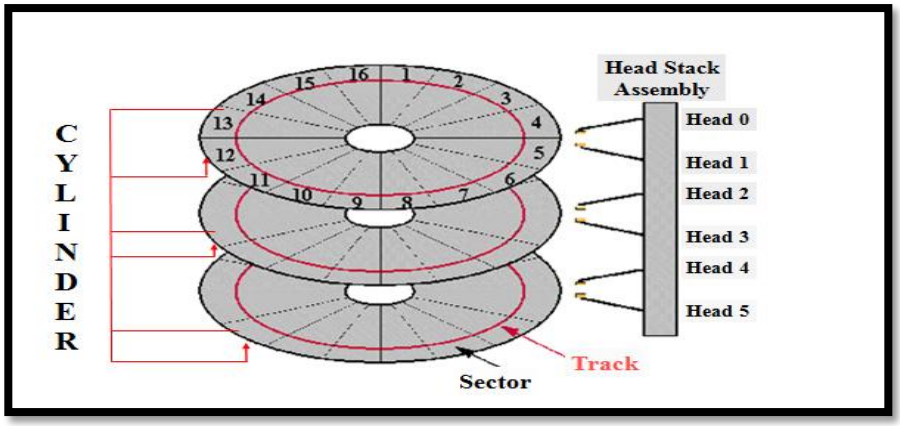
   How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty,so your first unique pages will all cost one fault each.
   a)LRU replacement,
   b)FIFO replacement,
   c)Optimal replacement
5. Write Short Notes on:
   a) Beladys anomaly
   b) Frame allocation strategies
   c) NRU (Not recently Used)
   d) The Clock Page Replacement Algorithm
   e)Thrashing

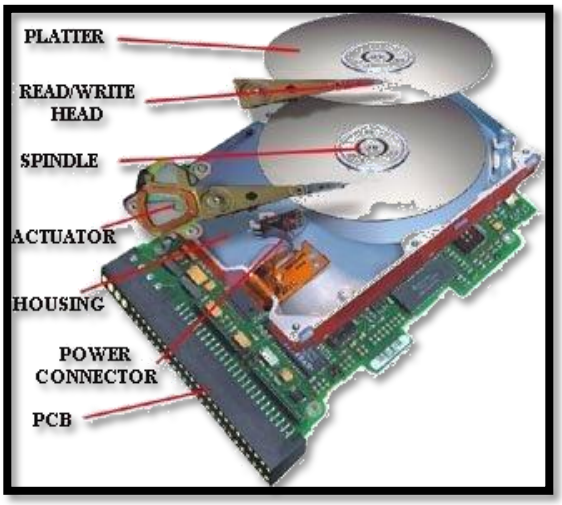
Module 5

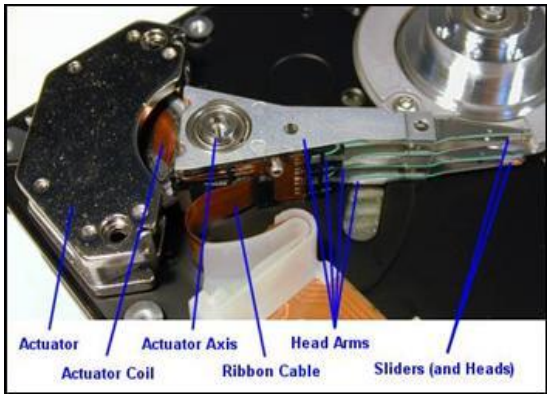
LECTURE 1

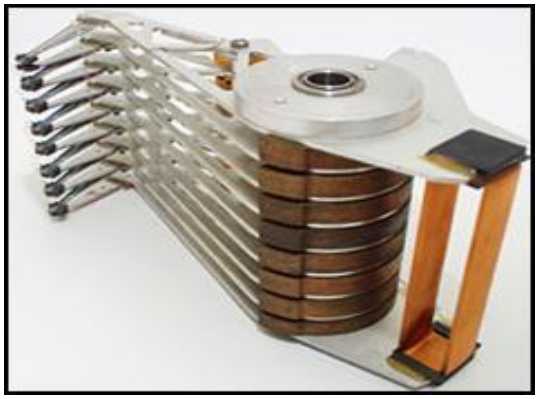
- Hard disks drives are organized as a concentric stack of disks or 'platters'.
- Each platter has 2 surfaces.
- Platter is made from aluminum, ceramic, or class, coated with a magnetic materials such as iron oxide.

- ***Platters***: The shiny rigid disks. Multiple platters increase storage without equivalent increase in cost.
- ***Heads***: The read/write heads of a hard drive. Disk assembly must be sealed &micro-filtered.
- ***Tracks***: Lanes centered around platters.
- ***Sectors*** / ***Clusters***: Each track was divided into sectors. Several sectors form a cluster.
  Cylinders: A grouping of the same tracks vertically through the stack of platters




PLATTER
READ/WRITE HEAD
SPINDLE
ACTUATOR
HOUSING
POWER CONNECTOR
PCB


Actuator
Actuator Coil
Actuator Axis
Ribbon Cable
Head Arms
Sliders (and Heads)



Disk Scheduling Algorithms

INTRODUCTION

Disk scheduling is is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

. PURPOSE

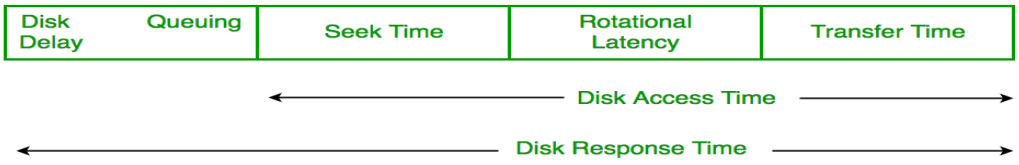
Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

- **Seek Time:**Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:

```
Disk Access Time = Seek Time +
                   Rotational Latency + Transfer Time
```



- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. Average Response time is the response time of the all requests. Variance Response Time is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

TYPES OF DISK SCHEDULING ALGORITHMS

Although there are other algorithms that reduce the seek time of all requests, but we will only concentrate on the following disk scheduling algorithms:

i.First Come-First Serve (FCFS)
ii.Shortest Seek Time First (SSTF)
iii.Elevator (SCAN)
iv.Circular SCAN (C-SCAN)
v.LOOK
vi.C-LOOK

These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why C-LOOK is the best algorithm to use in trying to establish less seek time.
Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

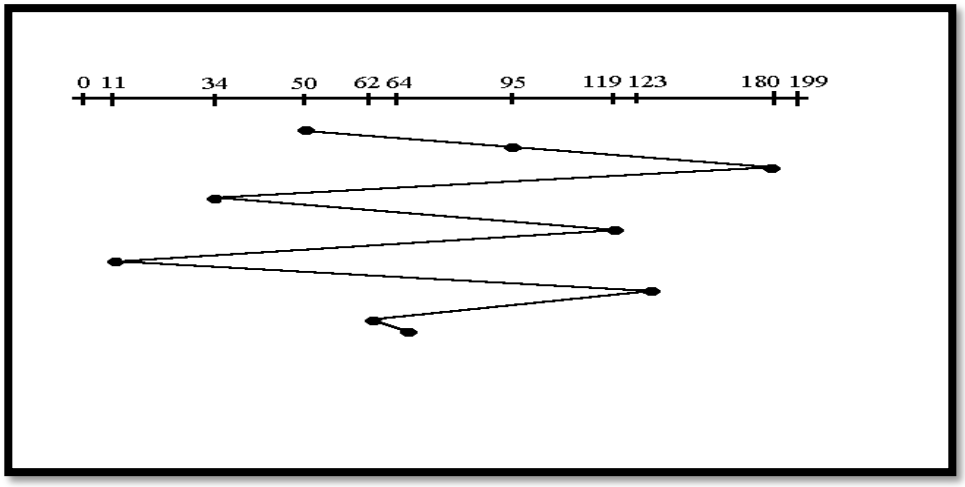


Fig: First Come -First Serve (FCFS)

1. *First Come -First Serve (FCFS)* All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

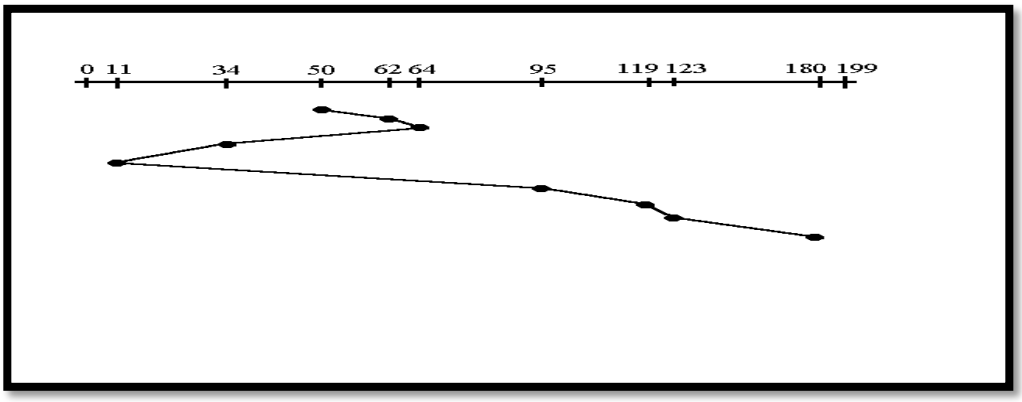


Fig: Shortest Seek Time First (SSTF)

2. Shortest Seek Time First (SSTF) :In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to eachother the other requests will never be handled since the distance will always be greater.

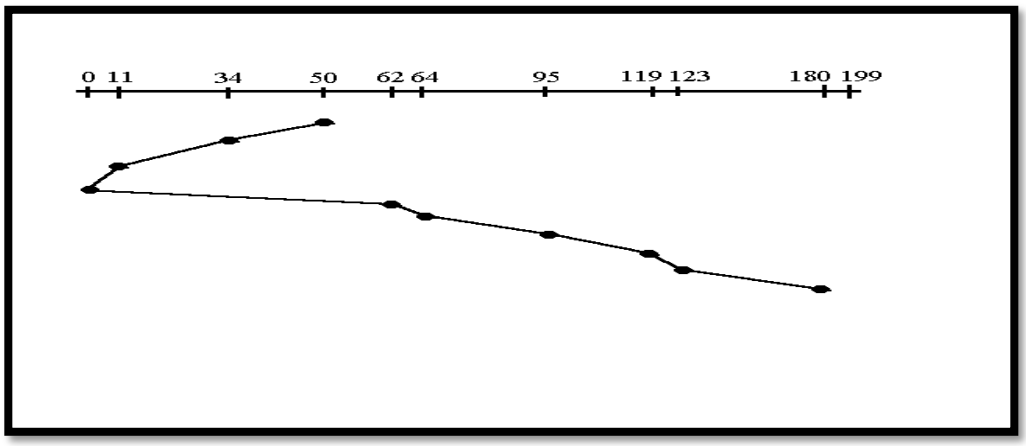


Fig:Elevator (SCAN)

3. Elevator (SCAN) :This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.

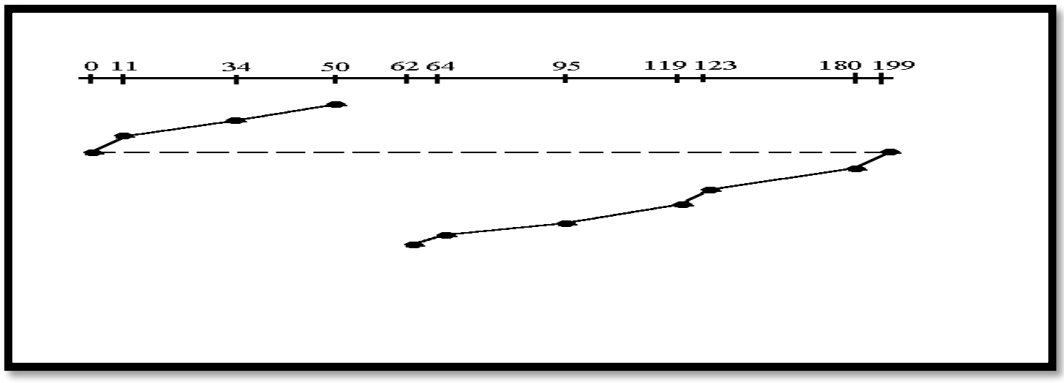


Fig:Circular Scan (C-SCAN)

4. Circular Scan (C-SCAN) :Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the mose sufficient.

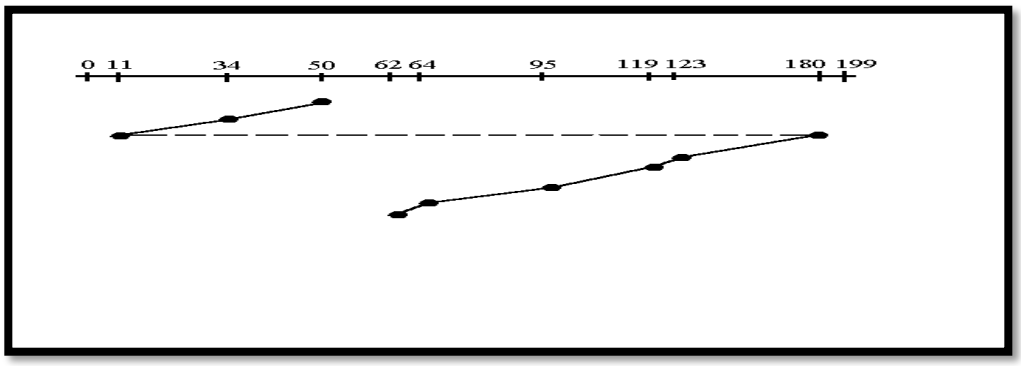


Fig:C-LOOK

*5. C-LOOK :*This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.

From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

FILE CONCEPT

File: File concept, access methods, directory structure, file system structure, UNIX file structure, allocation methods (contiguous, linked, indexed), free-space management (bit vector). [2L]

File system

File system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them.

Most file systems make use of an underlying data storage device such as Hard Disks that offers access to an array of fixed-size blocks which is the smallest logical amount of disk space that can be allocated to hold a file.

File systems typically have directories which associate file names with files, usually by connecting the file name to an index in a file allocation table of some sort, such as the FAT in a DOS file system, or an inode in a Unix-like file system.

File names are simple strings, and per-file Metadata is maintained which is the bookkeeping information, typically associated with each file within a file system.

Metadata could contain file attributes such as file size, data and time of creation or modification of the file, owner of the file, access permissions etc.

Types of File system

File system types can be classified into disk file systems, network file systems and flash file systems.

A disk file system is a file system designed for the storage of files on a data storage device, most commonly a disk drive e.g. FAT, NTFS, etx2, ext3 etc.

A network file system is a file system that acts as a client for a remote file access protocol, providing access to files on a server e.g. NFS, SMB etc.

A flash file system is a file system designed for storing files on flash memory devices.

File system and OS

Operating systems provide a file system, as a file system is an integral part of any modern operating system.

Windows Operating system supports FAT and NTFS File Systems

Linux popularly supports ext2 and ext3 File Systems

Other flavors of Operating Systems may support other File Systems like UFS in many UNIX Operating Systems and HFS in MAC OS X.

All Operating Systems provide a user interface like Command Line (CLI) or File Browser to access and manage File System information.

File Allocation Table (FAT)

The File Allocation Table (FAT) file system was initially developed for DOS Operating System and was later used and supported by all versions of Microsoft Windows.

It was an evolution of Microsoft's earlier operating system MS-DOS and was the predominant File System in Windows versions like 95, 98, ME etc.

All the latest versions of Windows still support FAT file system although it may not be popular.

FAT had various versions like FAT12, FAT16 and FAT32. Successive versions of FAT were named after the number of bits in the table: 12, 16 and 32.

NTFS

NTFS or the NT File System was introduced with the Windows NT operating system.

NTFS allows ACL-based permission control which was the most important feature missing in FAT File System.

Later versions of Windows like Windows 2000, Windows XP, Windows Server 2003, Windows Server 2008, and Windows Vista also use NTFS.

NTFS has several improvements over FAT such as security access control lists (ACL) and file system journaling.

File System in Linux

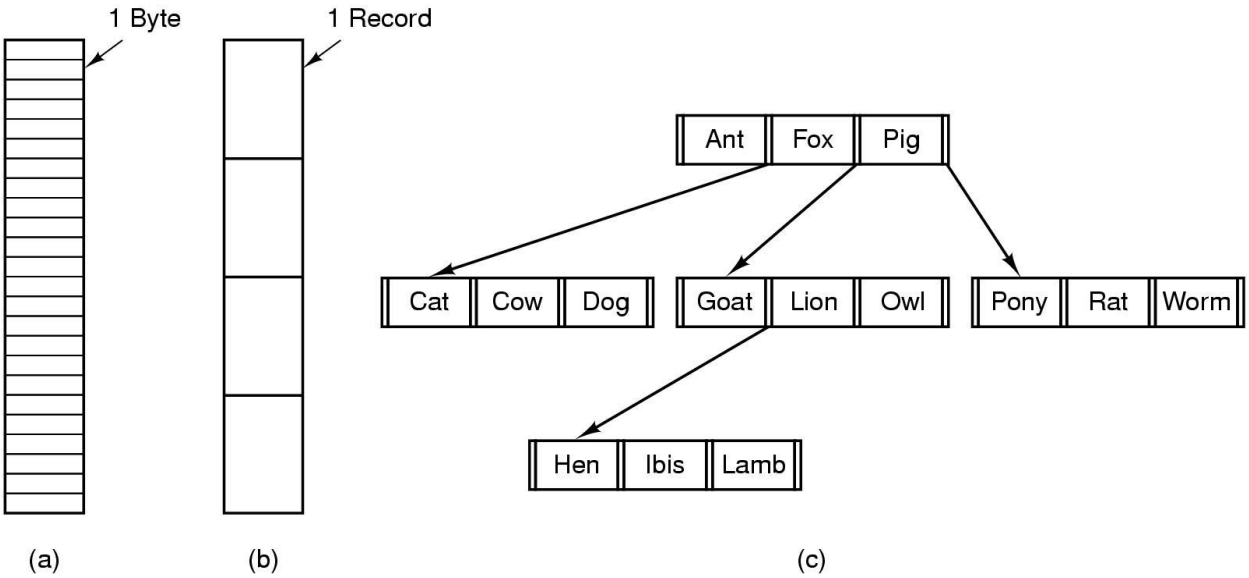
Linux supports many different file systems, but common choices for the system disk include the ext family (such as ext2 and ext3), XFS, JFS and ReiserFS.

The ext3 or third extended file system is a journaled file system and is the default file system for many popular Linux distributions .

It is an upgrade of its predecessor ext2 file system and among other things it has added the journouling feature.

A journaling file system is a file system that logs changes to a journal (usually a circular log in a dedicated area) before committing them to the main file system. Such file systems are less likely to become corrupted in the event of power failure or system crash.

File Structure:



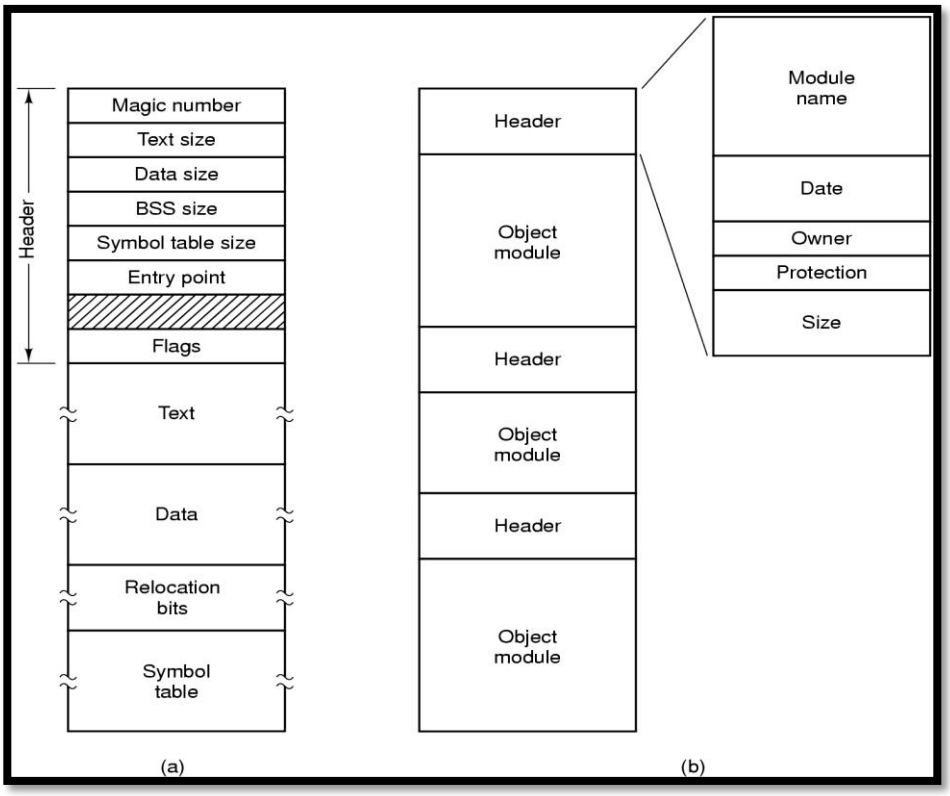
- Three kinds of files

  a.byte sequence

  b.record sequence

  c.tree

File Types



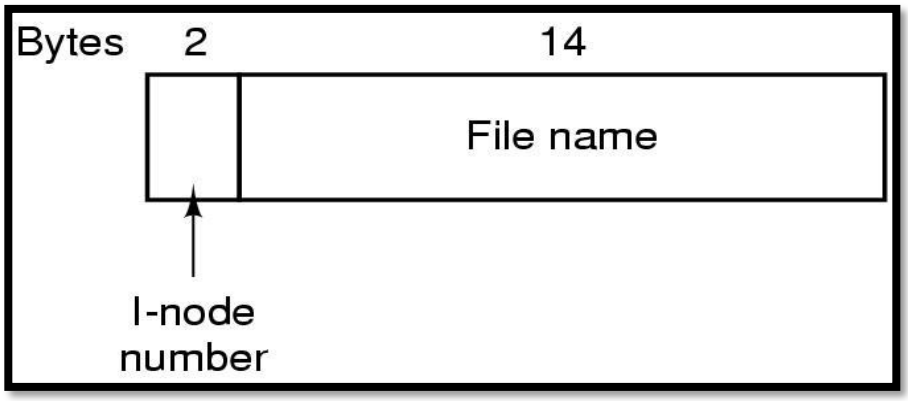
(a) An executable file (b) An archive

File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape
- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

UNIX file structure

The UNIX V7 File System:



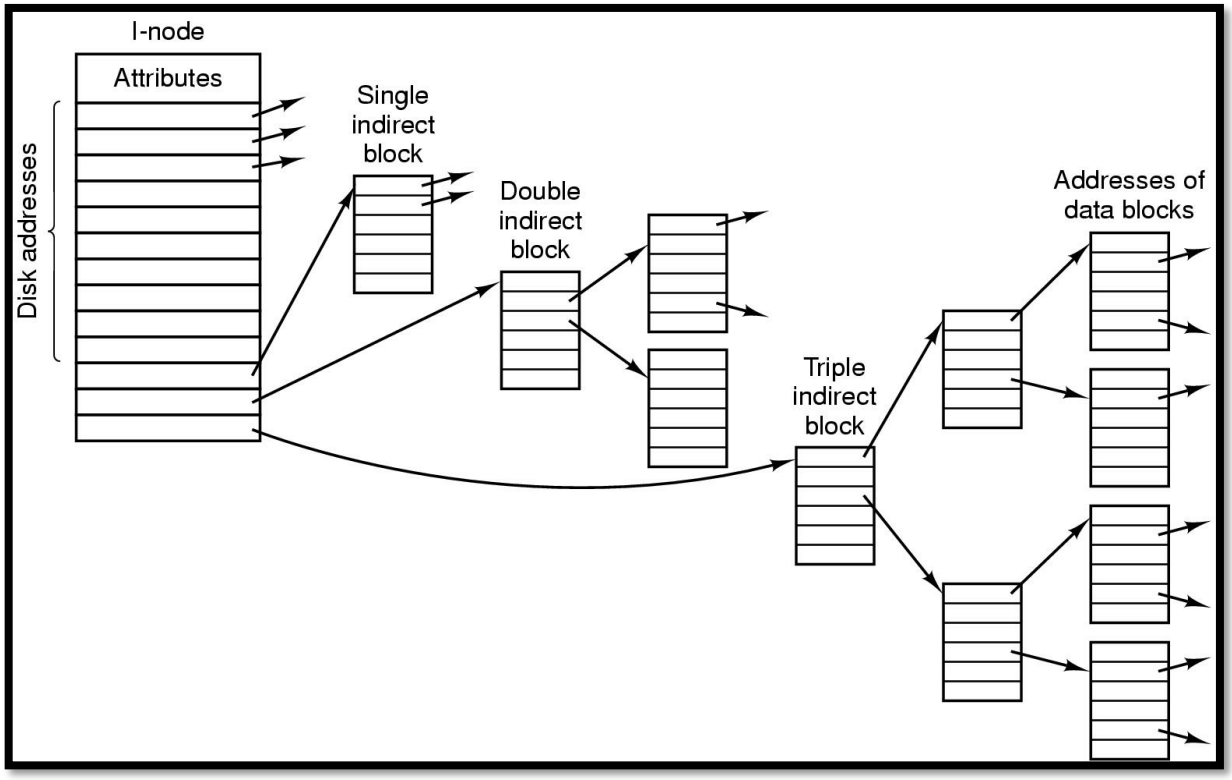


Fig: The UNIX V7 File System

File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

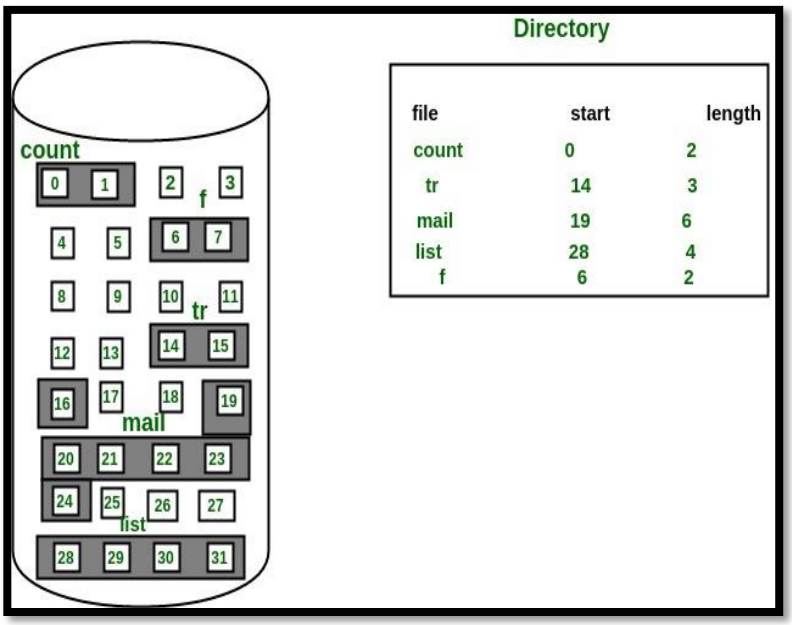
All the three methods have their own advantages and disadvantages as discussed below:

1. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.* This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



Advantages:

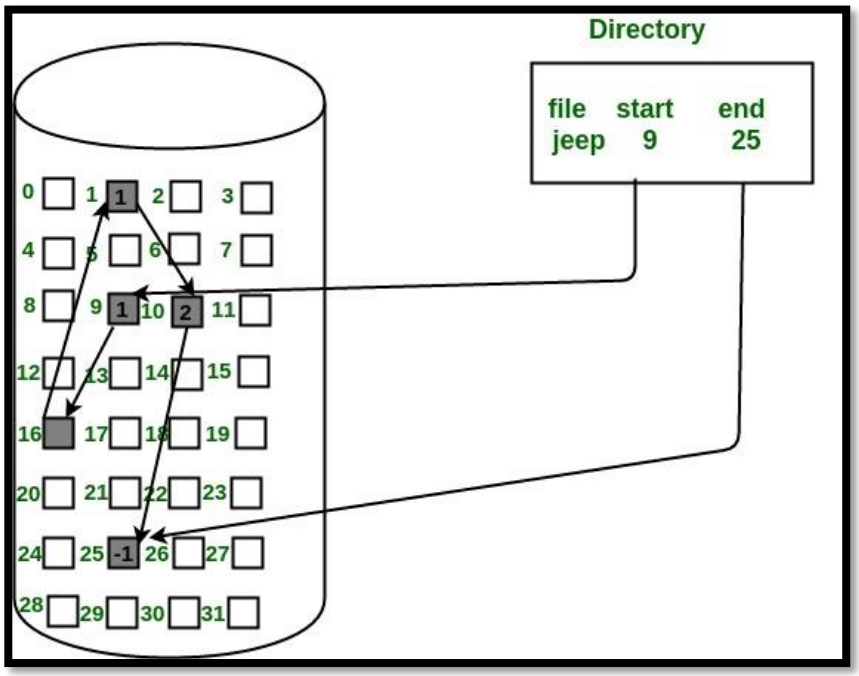
- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation

- In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.
  The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
- *The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.*



Advantages:

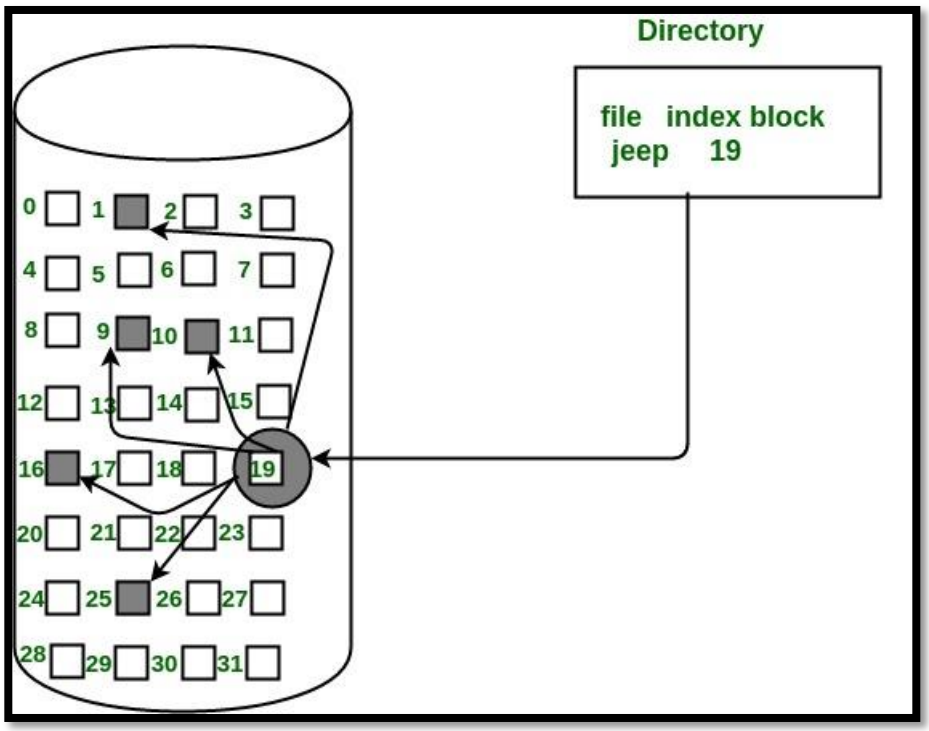
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

3. Indexed Allocation

- In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

FREE-SPACE MANAGEMENT

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks – those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list

1. Bit Vector

- The free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be

  00111100111111000110000011100000 …

- The main **advantage** of this approach is its relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

Example

The Intel family starting with the 80386 and the Motorola family starting with the 68020 (processors that have powered PCs and Macintosh systems, respectively) have instructions that return the offset in a word of the first bit with the value 1. In fact, the Apple Macintosh operating system uses the bit-vector method to allocate disk space.

The calculation of the block number is

(number of bits per word) x (number of 0-value words) + offset of first 1 bit.

Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory (and is written to disk occasionally for recovery needs). Keeping it in main memory is possible for smaller disks, such as on microcomputers, but not for larger ones.

A 1.3-GB disk with 512-byte blocks would need a bit map of over 332 KB to track its free blocks. Clustering the blocks in groups of four reduces this number to over 83 KB per disk.

2. Linked List

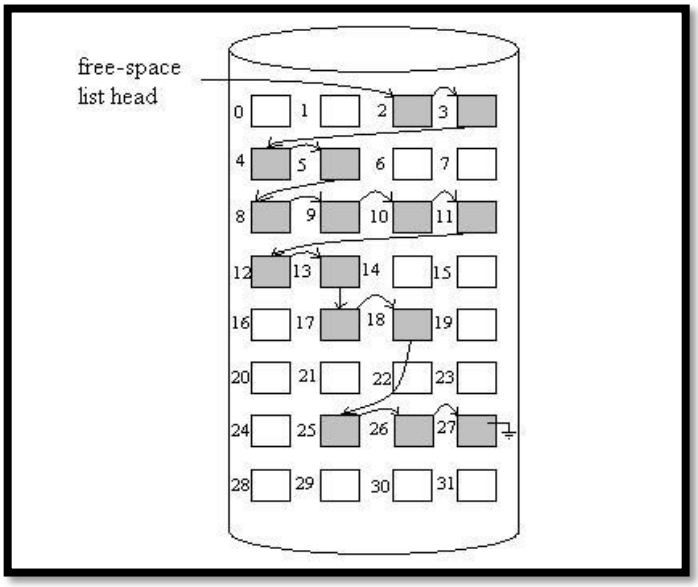
Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.

This first block contains a pointer to the next free disk block, and so on.

In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.

However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.

The FAT method incorporates free-block accounting data structure. No separate method is needed.



3. Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.The first n-1 of these blocks are actually free.

  The last block contains the addresses of another n free blocks, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

4. Counting

- We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.

- Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than one.

I/O MANAGEMENT

LECTURE 1

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

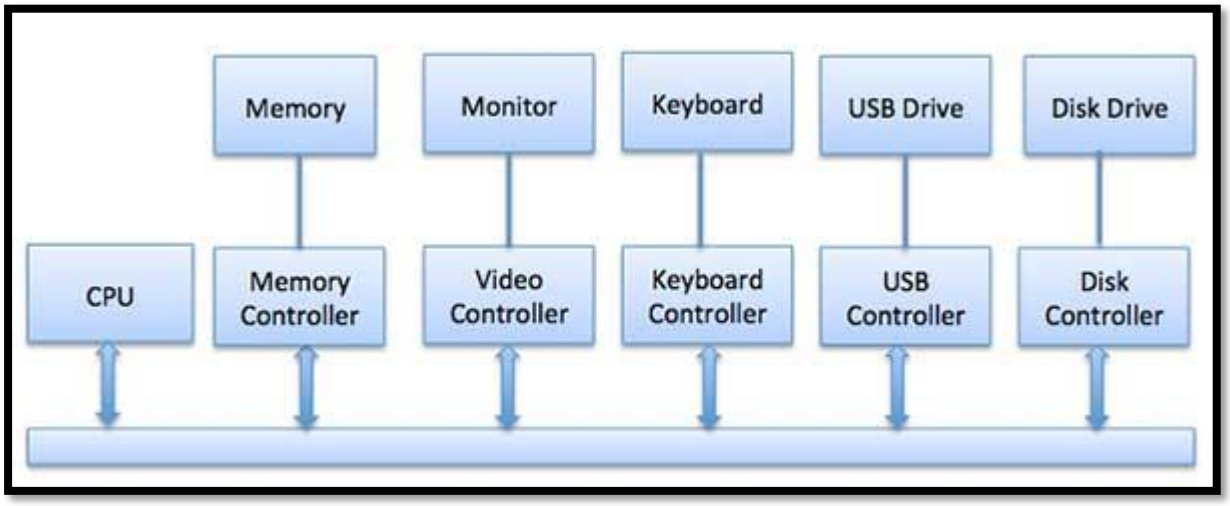
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

- **Synchronous I/O** − In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** − I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

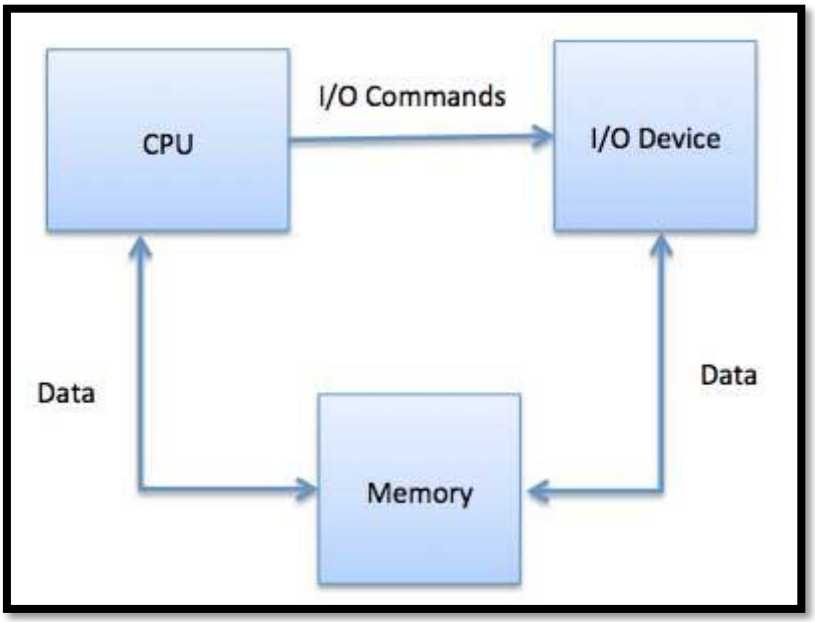
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

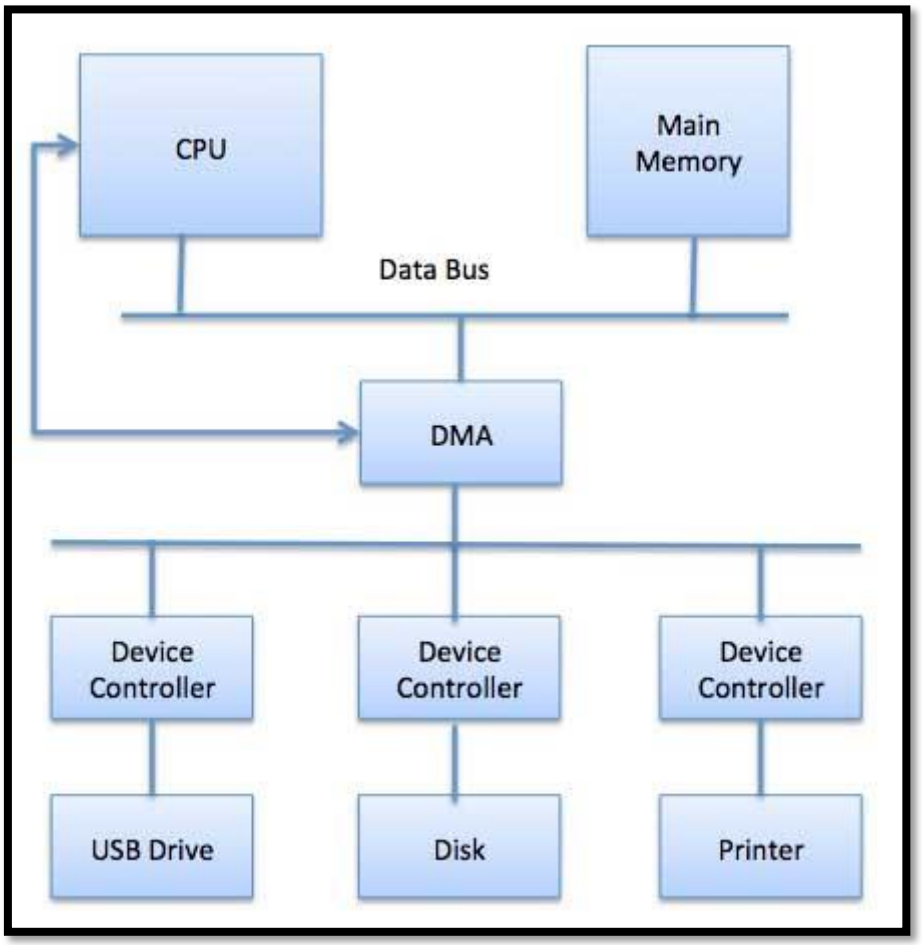
DIRECT MEMORY ACCESS (DMA)

LECTURE 2

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows −

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

MCQ Questions

1) Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing _____ number of requests. (GATE CS 2014
(A) 1
(B) 2
(C) 3
(D) 4

2) Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs? (GATE CS 2004)
(A) 50%
(B) 40%
(C) 25%
(D) 0%

3) Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _____ tracks
(A) 8
(B) 9
(C) 10
(D) 11

4.Write Short notes on:
a) Device Driver
b) DMA
c)Bit Vector
d) Polling I/O vs Interrupt I/O
e) Memory mapped I/O

Web/Video links:

[1] https://www.youtube.com/watch?v=qlH4-oHnBb8

[2] https://www.youtube.com/watch?v=59rEMnKWoS4

[3] https://www.youtube.com/watch?v=KNUJhZCQZ9c

[4] https://www.youtube.com/watch?v=6neHHkI0Z0o

[5] https://www.youtube.com/watch?v=bShqyf-hDfg

[6] https://www.youtube.com/watch?v=2quKyPnUShQ

[7] https://www.youtube.com/watch?v=ujoJ7J_l9cY

[8] https://www.youtube.com/watch?v=ZrMl-7QHg28&list=PLLDC70psjvq5hlT0kfr1sirNuees0NIbG&index=18

[9] https://www.youtube.com/watch?v=ygekh5ehxu4&list=PLLDC70psjvq5hlT0kfr1sirNuees0NIbG&index=17

[10] https://www.youtube.com/watch?v=2i2N_Qo_FyM&list=PLEbnTDJUr_IF_BnzJkkN_J0Tl3iXTL8vq