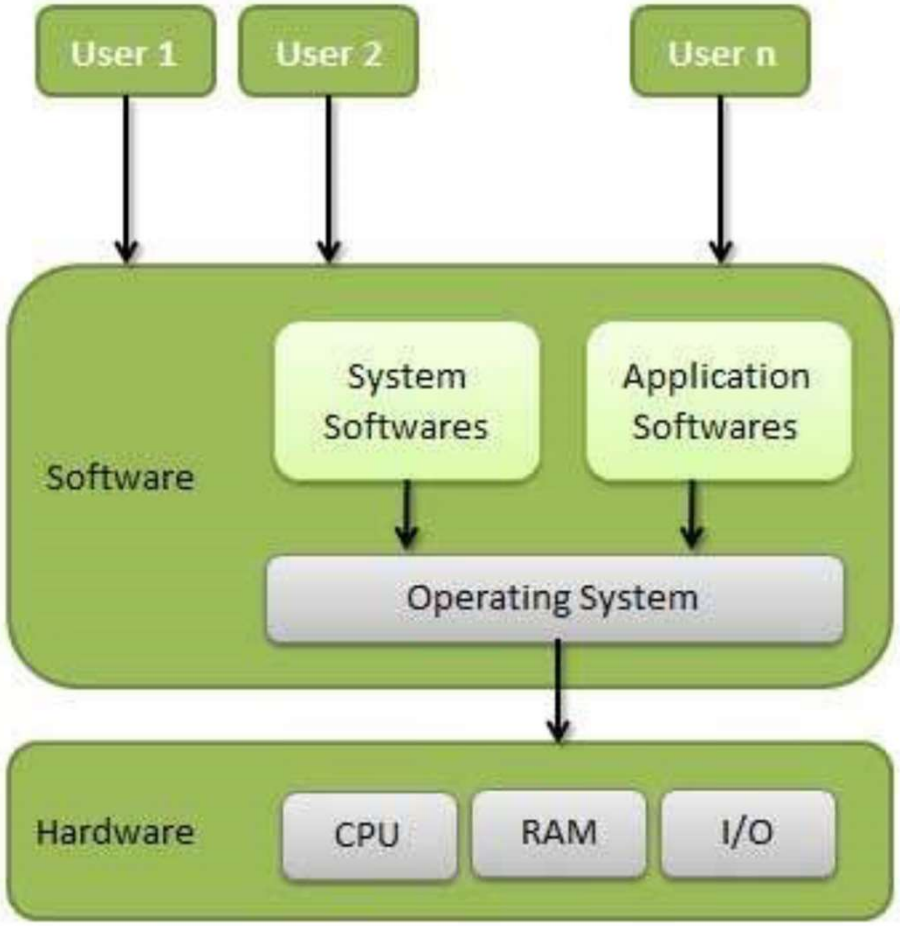
OPERATING SYSTEM

4th Semester CSE (R-23)

OUTLINE

- An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.



DEFINITION

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

MEMORY MANAGEMENT

- Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.
- Main memory provides a fast storage that can be accessed directly by the CPU.
- For a program to be executed, it must in the main memory.

An Operating System does the following activities for memory management :

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

PROCESSOR MANAGEMENT

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling. An Operating System does the following activities for processor management :

- Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

DEVICE MANAGEMENT

An Operating System manages device communication via their respective drivers. It does the following activities for device management :

- Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

FILE MANAGEMENT

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directories.

An Operating System does the following activities for file management –

- Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

OTHER IMPORTANT ACTIVITIES

Following are some of the important activities that an Operating System performs :

- Security : By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- Control over system performance : Recording delays between request for a service and response from the system.
- Job accounting : Keeping track of time and resources used by various jobs and users.

OTHER IMPORTANT ACTIVITIES

- Error detecting aids : Production of dumps, traces, error messages, and other debugging and error detecting aids.
- Coordination between other software and users : Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

TYPES OF OPERATING SYSTEM

A lot of operating system has been designed as per the necessity of users. We will summarize some of them here.

Batch system:

☐ Early computers were enormous machine run from a console. The users did not interact directly with the computer systems. To speed up processing, operators batched jobs together with similar needs and ran them through the computer as a group. The operator would sort programs into batches with similar requirements and as the system became available would run each batch. The output from each job would be sent back to the programmer.

TYPES OF OPERATING SYSTEM

- In the execution environment the CPU is often idle, as the speed of mechanical IO devices are much slower than those of electronic devices. The difference in speed makes the CPU wait for a long time always.
- Later on the introduction of disk technology allowed the operating system to keep all jobs on a disk and to perform better.

MULTI PROGRAMMED SYSTEMS:

- The most important aspect of job scheduling is the ability to multiprogramming. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.
- The Operating System keeps several jobs in memory simultaneously. These set of jobs are the part of jobs reside in the job pool of the disk. These jobs can be executed one by one. The Operating System can assign another job to the CPU whenever the running process needs any IO, so that the CPU can be used in a optimal way.

MULTI PROGRAMMED SYSTEMS:

- Multiprogramming is the first instance where the Operating System must make decisions for the users. The jobs reside in the job pool can be chosen by the Operating System through different scheduling criteria and scheduling algorithms.

TIME SHARING SYSTEMS:

- Time sharing (multi tasking) is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the user can interact with each program while it is running.
- An interactive computer system provides direct communication between the user and the system. So the response time of it should be short typically with in 1 second. A time shared operating system allows many users to share the computer simultaneously. It uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.

TIME SHARING SYSTEMS:

- Here several jobs may be kept simultaneously in memory, so the system may have memory management and protection. To obtain a reasonable response time jobs may have to be swapped in and out of main memory.
- A common method virtual memory is implemented to achieve this goal.

MULTIPROCESSOR SYSTEM:

Multiprocessor system also known as parallel system or tightly coupled system have more than one processor. In close communication, sharing the computer bus, the clock and sometimes memory and peripheral devices.

Advantage:

- Increased throughput: by increasing the number of processors, we hope to get more work done in less time, whereas it also incurs some overhead to maintain the processors.

MULTIPROCESSOR SYSTEM:

- Economy of scale: we can save money by sharing some system resources.
- Increased reliability: if functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.
- The ability to continue service providing proportional to the level of surviving hardware is called graceful degradation. Systems designed for graceful degradation are also fault tolerant systems.

MULTIPROCESSOR SYSTEM:

- The most common multiple processor systems now use symmetric multiprocessing (SMP), in which each processor runs an identical copy of the operating system and these copies communicate with one another as needed. Some systems use asymmetric multiprocessing, in which each processor is assigned a specific task. A master processor controls the system, other processors either look to the master for instruction or have predefined task. This scheme defines a master slave relationship.

DISTRIBUTED SYSTEMS:

A network is a communication path between two or more systems. Distributed systems depend on network for their functionality. By being able to communicate, distributed systems are able to share computational tasks and provide a rich set of features to users. As a part of this centralized systems today act as server systems to satisfy requests generated by client systems.

- Server systems can be broadly categorized as compute server and file server

DISTRIBUTED SYSTEMS:

Compute server system: 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 system: provides a file system interface where clients can create, update, delete and read files.

The computer network used in the distributed operating system does not share a memory or a clock instead, each processor has its own local memory. The processors communicate with each other through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled system.

REAL TIME SYSTEM:

A real time system is used when rigid time requirements have been placed on the operations of a processor or the flow of data, thus it is often used as a control device in a dedicated application.

- Sensors being data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. For e.g. scientific experiments, medical imaging systems etc.
- A real time system has well defined fixed time constraints, hence processing must be done within the defined constraints or the system fails.

REAL TIME SYSTEM:

Real time system comes in two flavors :

- Hard real time system guarantees that the critical task be completed on time. This goal requires that all delays in the system be bounded from the retrieval of stored data to the time that it takes the operating system to finish any request made of it.
- Soft real time system is less restrictive compared to hard real time system, where a critical real time task gets priority over other tasks and retain that priority until it completes.

REAL TIME SYSTEM:

- The soft real time systems are useful however in several areas including multimedia, virtual reality and advanced scientific projects.
- Virtual memory almost never found in real time systems.

THE COMPUTER SYSTEM STRUCTURE RELATED TO OPERATING SYSTEM

- A modern general-purpose system consists of a CPU and a number of device controllers that are connected through a common bus, which provides access to share memory. Each device control is in charge of a specific type of device.
- To start a program or a computer system we need a bootstrap program, which is stored in read only memory (ROM) or in EEPROM as firmware. It initializes all aspects of the system from CPU registers to device controller to the memory contents. After execution of the bootstrap program the OS starts executing the first process "init" and wait for some event to occur.

THE COMPUTER SYSTEM STRUCTURE RELATED TO OPERATING SYSTEM

- The occurrence of an event is usually signaled by an interrupt from either the hardware or software.
- Software may trigger an interrupt by executing a special operation called a system call or monitor call. When the CPU is interrupted it stops what it is doing and immediately transfers execution to a fixed location, which actually contains the starting address where the service routine for the interrupt is located.
- The information about the interrupts may be stored in a table of pointer which is generally stored in low memory (generally the first hundred location).

THE COMPUTER SYSTEM STRUCTURE RELATED TO OPERATING SYSTEM

- These locations hold the address of the interrupt service routines for the various devices. This array or interrupt vector of addresses is then indexed by a unique device number given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.
- A system call is invoked in a variety of ways, depending on the functionality provided by the processor. In all forms it is the method used by a process to request action by the operating system.

THE COMPUTER SYSTEM STRUCTURE RELATED TO OPERATING SYSTEM

- IO structure: A device controller is in charge of a specific type of device or may be attached to more than one device. The device controller is responsible for moving the data between the peripheral devices that it controls and its local storage buffer.
- IO interrupts: To start an IO operation, the CPU loads the appropriate register with in the device controller, in turn the device controller examines the registers to find the action, that is READ or WRITE.
- For a read operation it starts transfer of data from the device to its local buffer and after completion it informs the CPU. The information passing can be done by triggering an interrupt.

OPERATING SYSTEM - SERVICES

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system:

- ☐ Program execution
- ☐ File System manipulation
- ☐ Error Detection
- ☐ Protection
- ☐ I/O operations
- ☐ Communication
- ☐ Resource Allocation

OPERATING SYSTEM - SERVICES

- Program execution : Operating systems handle many kinds of activities from user programs to system programs. Following are the major activities of an operating system with respect to program management:

- ☐ Loads a program into memory.
- ☐ Executes the program.
- ☐ Handles program's execution.
- ☐ Provides a mechanism for process synchronization.
- ☐ Provides a mechanism for process communication.
- ☐ Provides a mechanism for deadlock handling.

OPERATING SYSTEM - SERVICES

I/O Operation: An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

- An Operating System manages the communication between user and device drivers.
- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

OPERATING SYSTEM - SERVICES

File system manipulation : A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long term storage purpose.

- A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directories.
- Following are the major activities of an operating system with respect to file management:

OPERATING SYSTEM - SERVICES

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

COMMUNICATION

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication:

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

ERROR HANDLING

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling:

- ☐ The OS constantly checks for possible errors.
- ☐ The OS takes an appropriate action to ensure correct and consistent computing.

RESOURCE MANAGEMENT

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management:

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

PROTECTION

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection:

- ☐ The OS ensures that all access to system resources is controlled.
- ☐ The OS ensures that external I/O devices are protected from invalid access attempts.
- ☐ The OS provides authentication features for each user by means of passwords

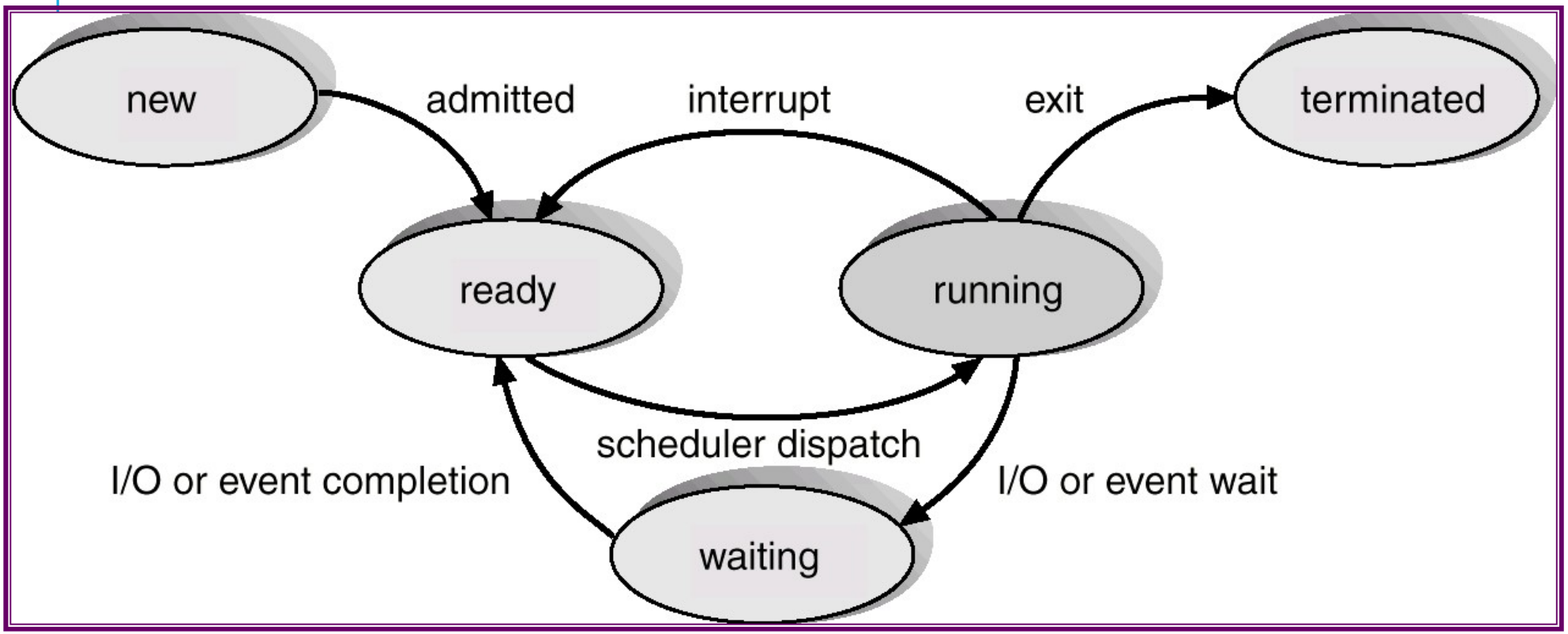
PROCESS

- A process can be thought of as a program in execution. It needs some resources to continue its work such as CPU time, memory, files and I/O devices.
- We can say that the program itself is not a process. A program is a passive entity such as the contents of a file stored on disk, where as a process is an active entity with a program counter specifying the next instruction to execute and a set of associated resources attached to it.

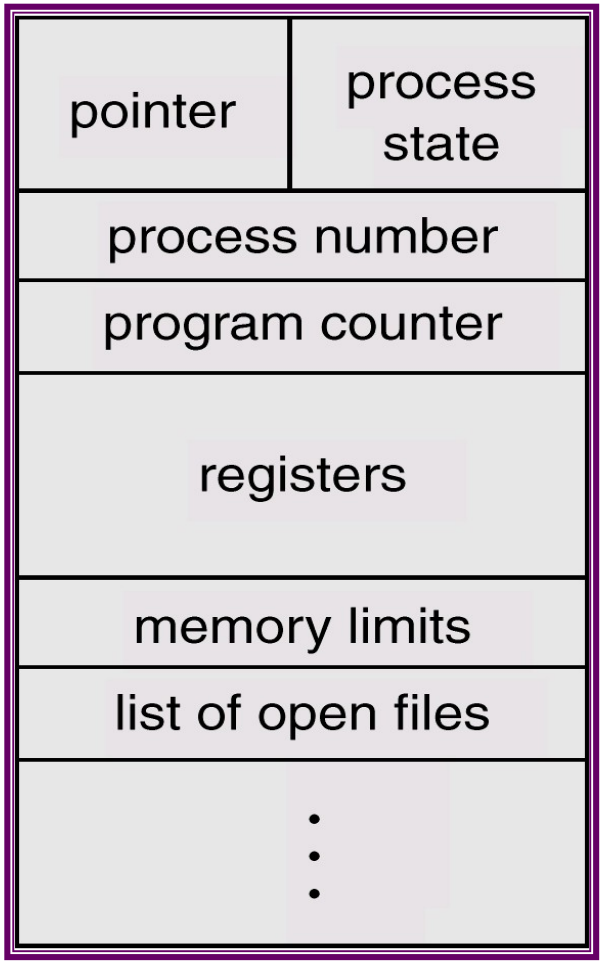
PROCESS STATE:

- The state of a process is defined in parts by the current activity of the process, such as
  - New: The process is being created.
  - Running: instructions are being executed.
  - Waiting: the process is waiting for some event to occur.
  - Ready: waiting to be assigned to some process.
  - Terminated: the process has finished execution
- Only one process can be running on any processor at any instance, although many processes may be ready and waiting.

PROCESS STATE:



PROCESS CONTROL BLOCK:



☐ Each process is represented in the OS by a process control block. The PCB contains all information of a specific process associated with it such as:

- Process state: the state may be new, ready, running, waiting, or halted.
- Program counter: the counter indicates the address of the next instruction to be executed.

PROCESS CONTROL BLOCK:

- CPU registers: they are many types such as: index registers, stack pointers, general purpose registers etc..
- CPU scheduling information: process priority number, pointers to scheduling queues etc.
- Memory management information: such as base and limit registers, page table and segment table information etc.
- Accounting information: includes the amount of CPU and real time used, time limits, job or process nos. etc.
- I/O status information: the information includes the list of I/O devices allocated to the process, a list of open files etc.

SCHEDULERS:

A process migrates between various scheduling queues throughout its life time. The selection of process for the scheduling purpose is done by the scheduler.

In a batch system often more processes are submitted then can be executed immediately. These processes are spooled to a mass storage for later execution.

The scheduler may be categorized into three categories:

- long term scheduler
- short term scheduler
- medium term scheduler

SCHEDULERS:

long term scheduler : The long-term scheduler selects process from the spooled processes kept in mass storage for later execution and loads them into memory for execution.

It controls the degree of multiprogramming by looking the rate of creation and termination rates of processes thus, it may be invoked only when a process is leaving the system after its completion.

As it has lot of time to select a process it should careful selection according to its nature, as the process may be I/O bound (spends more time doing I/O) or CPU bound (spends more time for computation) so a long-term scheduler should select a good process mix of i/o bound and CPU bound to balance the load of CPU.

SCHEDULERS:

Short Term Schedulers:

- After submission of the process in memory the short term scheduler will take over the responsibility.
- By the help of dispatcher, the short-term scheduler choose a process from the ready queue to give the control of CPU by using some scheduling algorithm.

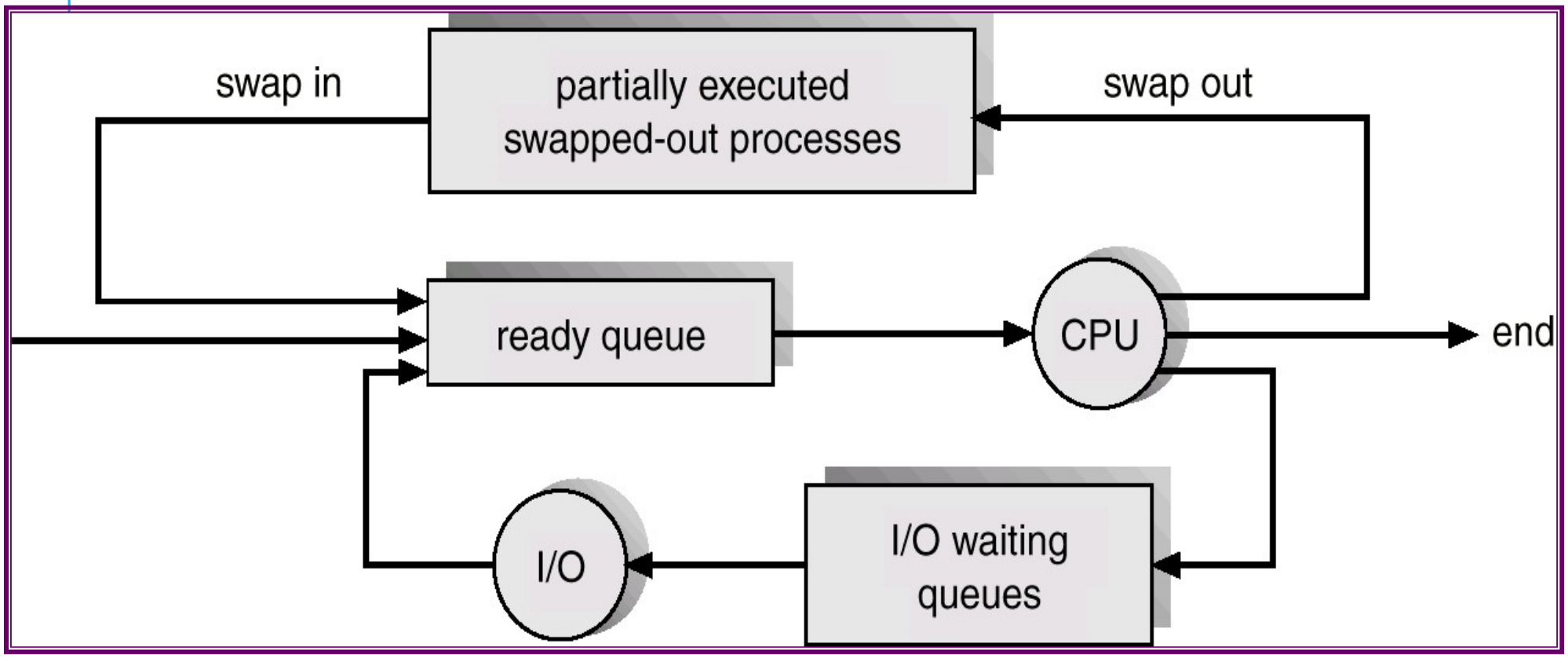
SCHEDULERS:

Medium term scheduler :

On some systems such as a time-shared system may introduce an intermediary level of scheduler called medium-term scheduler.

When there is lack of main memory at the time of execution due to the virtual memory implementation that is when the required memory for execution is larger than the available memory then some process lies in the waiting queue may be swapped out to make room for a currently executing process. This job is done by the medium-term scheduler.

SCHEDULERS:


swap in
partially executed swapped-out processes
swap out
ready queue
CPU
end
I/O
I/O waiting queues

CONTEXT SWITCH:

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as context switch.

- The information about the context is stored inside the PCB of a process.
- Context switch is pure overhead because the system does no useful work while switching.
- Context switch times are highly dependent on hardware support.

COMMUNICATION BETWEEN PROCESSES (INTER PROCESS COMMUNICATION):

A concurrent process can be of two types:

- Independent: a process is independent if it cannot affect or be affected by other processes executing in the system.
- Co-operative: a process is said to be co-operating if it can affect or be affected by other processes.

The inter process communication (IPC) in Co-operating processes can be done through two models:

- ☐ Shared memory
- ☐ Message passing.

COMMUNICATION BETWEEN PROCESSES (INTER PROCESS COMMUNICATION):

Shared Memory: Inter process communication using shared memory requires communicating processes to establish a shared memory region.

- A shared memory region resides in the address space of the process creating the shared memory segment. Other processes that wish to communicate using this shared memory segment must attach themselves to the address space of the creator's address space.
- They can then exchange information by reading and writing data in the shared area. The form and the locations determined by these processes are not under the control of the operating system. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

COMMUNICATION BETWEEN PROCESSES (INTER PROCESS COMMUNICATION):

Message Passing:

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is useful in distributed environment, where the communication processes may reside on different computers connected by a network.

A message passing facility provides at least two operations

- Send
- Receive

COMMUNICATION BETWEEN PROCESSES (INTER PROCESS COMMUNICATION):

Message sent by processes may either fixed or variable size. If the message is of fixed size the system level implementation becomes easy, this restriction however makes the task of programming more difficult. Whereas for a variable size message system level implementation is difficult with programming task simpler.

In order to send and receive message from each other, the process must have a communication link existing between them. The link can be implemented in many ways. Logically a link can be implemented using the following methods:

- ☐ Direct or indirect communication
- ☐ Synchronous or asynchronous communication
- ☐ Automatic or explicit buffering.

DIRECT COMMUNICATION:

In this method each process which wants to communicate must name explicitly the sender or receiver of the communication, and the send () and receive () primitives are defined as follows

- Send (P, message): send a message to process P
- Receive (Q, message): receive a message from process Q

A communication link in this method will have the following properties:

- Automatic link establishment take place between every pair of processes which wants to communicate and the processes are required to know only each other's identity for communication.

DIRECT COMMUNICATION:

- Only two processes are associated with a link
- There will be only one link between each pair of processes.

The discussed mechanism of direct communication exhibits symmetry in addressing, that is both the sender and the receiver process must name other to communicate whereas a variant of this scheme employs asymmetry in addressing where the send and receive primitives may be defined as:

- Send (P, message): send a message to process P
- Receive (ID, message): receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

DIRECT COMMUNICATION:

Disadvantage of symmetric and asymmetric schemes are the limited modularity of the resulted process definition, that is a process name changing may require examining all other process definition and all the references to the old name must be found, so that they can be modified to the new name.

INDIRECT COMMUNICATION:

- In this scheme the sending and receiving of message is done using a mail box (also called as ports). A mailbox abstractly can be viewed as an object, where the processes place their message and removes the messages from it.
- Each mailbox will have a unique identification and a process can communicate with some other processes through number of different mailboxes.
- Communication between two processes is possible only if they have a shared mailbox. In this scheme the send and receive primitives are defined as:
  - Send (B, message): send a message to mailbox B
  - Receive (B, message): receive a message from mailbox B

INDIRECT COMMUNICATION:

The communication link in this scheme has the following properties:

- A link can be established between a pair of processes only if they have a shared mail box.
- More than two processes can be associated with a link.
- There can be a number of different links between each pair of communicating processes and each link corresponds to one mailbox.
- A mailbox can be owned by a process or by the operating system.

INDIRECT COMMUNICATION:

- If it is owned by a process then the process that creates the mail box can receive the message from that mail box and others can send messages to this mail box. A mail box disappears when the process that created the mailbox terminates and the other processes those tries to send message to the mailbox must be notified about the absence of the mailbox.
- If a mailbox is owned by the operating system, it has its own existence. It is independent and is not attached to any process.

INDIRECT COMMUNICATION:

Operating system provides a mechanism using which a process.

- Can create a new mailbox
- Can send and receive messages through the mailbox.
- Can destroy a mailbox

The process who creates the mailbox is the owner of the mailbox by default. The ownership and receive privilege can be passed to other processes through some system calls.

SYNCHRONIZATION:

Message passing may be either blocking or nonblocking also known as synchronous or asynchronous

- Blocking send: the sending process is blocked until the message is received by the receiving process or by the mailbox.
- Non blocking send: the sending process sends the message and resumes its operation.
- Blocking receive: the receiver blocks until a message is available.
- Non blocking receive: the receiver retrieves either a valid message or a null

BUFFERING:

Whether communication is direct or indirect, messages resides in a temporary queue called buffer. Basically, such queues are implemented in three ways:

- Zero capacity: here the link cannot have any message waiting in it, that is the sender must block until the receiver receives the message.
- Bounded Buffer: the queue has finite length n, thus can keep up to n messages, hence if the link is full the sender must block until space is available in the queue.
- Unbounded buffer: the queues length is potentially infinite; thus, any number of messages can wait in it, hence is known as automatic buffering.

PROCESS SCHEDULING

CPU scheduling is the basis of multi-programmed operating system. The computer can be made productive by switching the CPU among processes. The objective of multiprogramming is to have some process running at all times.

- On a system when a process is executing it's I/O the CPU sits idle so if we use this time productively then we can increase the CPU utilization.
- The solution is to keep several processes in memory at one time. When one process is busy with the I/O the control of the CPU is given to another process in the memory that is ready to execute.

PROCESS SCHEDULING

- A process execution generally alternates between the CPU execution and I/O wait cycle.
- Process execution always starts with a CPU burst and is followed by an I/O burst and so on and always completes with a CPU burst with a system request to terminate execution at the completion of a process execution.

CPU scheduler: 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 time scheduler by the help of some scheduling algorithm.

SCHEDULING:

Generally, the scheduling can be done in two ways.

- Non-Preemptive: Here once the CPU has been allocated to a process, the process keeps the CPU until it release the CPU by terminating itself after completion of execution or by switching to the waiting state.
- Preemptive: here the control of CPU can be taken any time due to some reason as a process may be preempted if a higher priority process arrives or due to a system call interruption etc.

SCHEDULING:

- In a time-sharing system a time slice is used, so after expiration of the time slot the process may be preempted. Here all processes are got the same time slot for its execution.
- The preemptive scheduling has greater performance on non-preemptive, but in some cases an inconsistent result may occur due to the preemptive scheduling, hence the non-preemptive scheduling may be used for better result.

SCHEDULING:

- CPU-scheduling decisions may take place under the following four circumstances.
  - When a process switches from the running state to waiting state (as a result of an i/o request or invocation of wait for the termination of one of the child process)
  - When a process switches from the running state to ready state (when an interrupt occurs)

SCHEDULING:

- When a process switches from the waiting state to ready state (at completion of i/o)
- When a process terminates.

When scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is non-preemptive as it does not have any choice, however for situations 2 and 3 we have some choice, hence can be preemptive.

DISPATCHER:

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler which includes certain functions as:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program.

The time taken by the dispatcher to stop one process and start another is known as dispatch latency.

SCHEDULING CRITERIA:

The criteria used for comparison of different available algorithms can make a substantial difference in the determination of the best algorithm to implement.

CPU utilization: We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).

Throughput: the number of processes completed its execution per unit time is known as throughput.

SCHEDULING CRITERIA:

Turnaround time: the interval from the time of submission of a process to the time of completion is known as turnaround time of a process.

Turnaround time = waiting time + execution time (CPU burst)

Waiting time: the CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of periods spent waiting in the ready queue.

Response time: in an interactive system, turn around time may not be the best criterion. Often a process can produce some output fairly early and can continue computing new results while the previous results are being output of the user.

SCHEDULING CRITERIA:

Thus, another measure is the time from the submission of a request until the first response is produced. This measure is called the response time.

We can choose an algorithm as an optimal one if it has

Minimum: Turnaround Time
Waiting Time
Response Time

Maximum: CPU Utilization
Throughput

SCHEDULING ALGORITHMS:

CPU scheduling deals with the problem of deciding which of the process in the ready queue is to be allocated to the CPU.

First come first served algorithm:

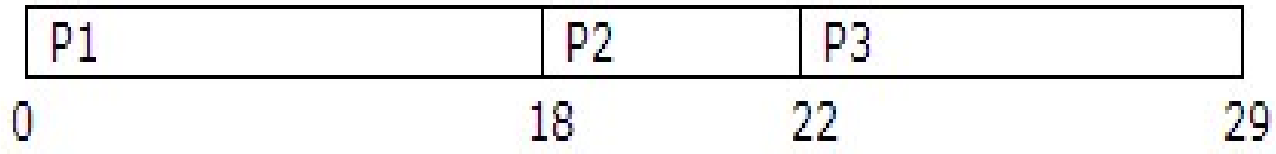
- This is the simplest algorithm and in this, the process which requests the CPU first is allocated the CPU first. This algorithm is easily managed with a FIFO queue
- When a process enters the ready queue, its PCB is linked onto the tail of the queue and when the CPU is free it is allocated to the process at the head of the queue.
- The average waiting time under the FCFS policy is quite long.

SCHEDULING ALGORITHMS:

Ex: consider a set of processes that arrives at time zero with CPU bursts as given

Process Name	CPU Burst
P1	18
P2	4
P3	7

Then, the Gantt chart produces by these processes is



As the processes arrived at time zero, the processes has their waiting times as

P1 0 – 0 = 0 P2 18 – 0 = 18 P3 22 – 0 = 22

Now if we calculate the average waiting time for these processes in FCFS then

AWT = (0 + 18 + 22) / 3 = 40 / 3 = 13.3

SCHEDULING ALGORITHMS:

The performance of this algorithm is also not so good as if a CPU bound process holds the CPU, then many I/O bound processes may be waiting at that time in the ready queue.

There may be a Convoy effect that is all other processes wait for the one big process to get off the CPU.

SCHEDULING ALGORITHMS:

Shortest job first algorithm:

In the shortest job first algorithm when the CPU is available, it is assigned to the process that has the smallest next CPU burst

- If two processes has the same length then FCFS can be used to break the tie
- The SJF scheduling algorithm is provably optimal that is it gives a minimum average waiting time for a given set of processes.
- This approach can be of two types
  - Shortest job first (Non-Preemptive)
  - Shortest remaining time first (Preemptive)

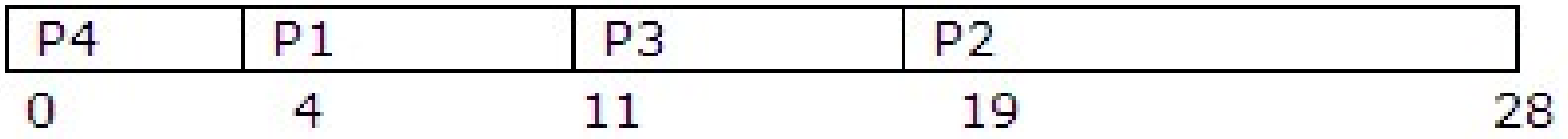
SCHEDULING ALGORITHMS:

Ex: given a set of processes with their corresponding CPU bursts. Assume that arrival time for all is zero.

Process Name	CPU Burst
P1	7
P2	9
P3	8
P4	4

As in the shortest job first the process with the shortest CPU burst will be taken first

So the Gantt chart will be drawn as:



SCHEDULING ALGORITHMS:

So the waiting times for the processes are:

P1 = 4 - 0 = 4

P2 = 19 - 0 = 19

P3 = 11- 0 = 11

P4 = 0

Hence the average waiting time = (4 + 19 + 11 + 0) / 4 = 8.5

SCHEDULING ALGORITHMS:

Shortest remaining time first

- The preemptive SJF or the SRTF algorithm will preempt the currently executing process, if it encounters any process having the lowest CPU burst compared to the current one.
- It will calculate the shortest remaining time and process it first.

For eg: Given a set of processes with their arrival time and CPU bursts

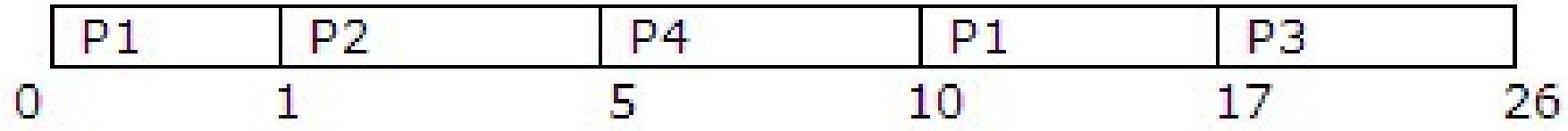
SCHEDULING ALGORITHMS:

Process Name	CPU Burst	Arrival Time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

At time 0 there is only one process that is P1 so the CPU will take this, but after 1 ms P2 arrives. So, the CPU will compare the CPU burst of process P2 with the remaining time of process P1, the process P1 is preempted and the control of CPU is given to P2. This procedure continues till the completion of all processes.

SCHEDULING ALGORITHMS:

According to the procedure the Gantt chart may be drawn as



So, the waiting times for the set of processes are

P1 = 10-1 = 9 ms

P2 = 1-1 = 0 ms

P3 = 17-2 = 15 ms

P4 = 5-3 = 2 ms

SCHEDULING ALGORITHMS:

The average waiting time with SRTF for the given set of processes is:

$9 + 0 + 15 + 2 / 4 = 6.5$ ms

If we will calculate the same with FCFS then we will get an average waiting time of 8.75 ms. So, the SRTF may be taken as a optimal solution for this problem. But in this algorithm the real difficulty arises to knowing the next CPU burst. So, one solution for this may be by predicting the next CPU burst by the past history.

- One approach is to make estimate based on the past behavior and run the process with the shortest estimated time.

SCHEDULING ALGORITHMS:

Priority Scheduling:

- In priority scheduling approach a priority number is associated with each process and the CPU is allocated to the process with the highest process.
- There is no general argument on whether 0 is the highest or lowest priority. It totally depends up on the designer of the OS.
- Priority scheduling can also be done in two ways:
  - Preemptive
  - No Preemptive
- There are also different categories of priorities
  - Static priority
  - Dynamic priority

SCHEDULING ALGORITHMS:

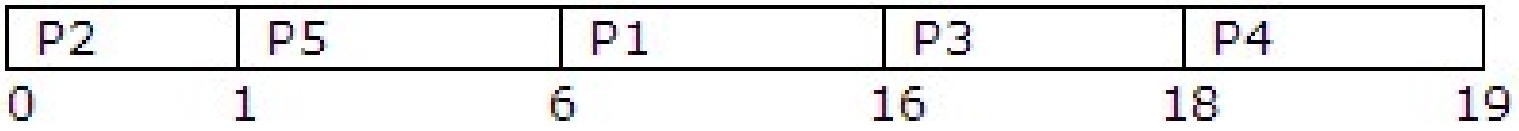
- The process with Static priority does not change its priority value with in its life span where as the Dynamic priority is responsive to change that is after some duration the priorities may be updated to a more appropriate value. This approach may incur some overhead but the overhead is hopefully justified by the increased responsiveness of the system.
- Purchased priorities: A member of the user community needs special treatment due to the importance or rush of the job. They need to pay some extra payment for a special priority over others.
- Deadline scheduling: Certain jobs are scheduled to be completed by a specific time or deadline.

SCHEDULING ALGORITHMS:

For a given set of processes with their priorities to calculate the average waiting time under non-preemptive that is the arrival time for all the processes are 0.

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

As the arrival times for all processes are 0 the Gantt chart can be drawn according to their priorities.



SCHEDULING ALGORITHMS:

So, the waiting times for the set of processes are

P1 = 6-0 = 6 ms

P2 = 0 ms

P3 = 16-0 = 16 ms

P4 = 18-0 = 18 ms

P5 = 1-0 = 1 ms

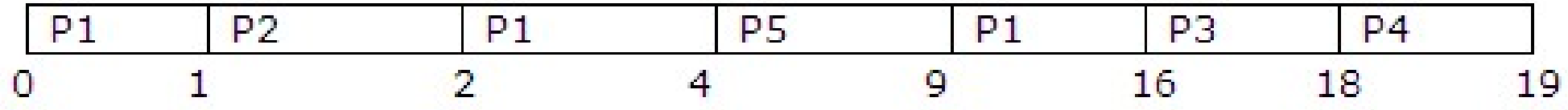
The average waiting time is 6+16+18+1/5 = 8.2 ms

SCHEDULING ALGORITHMS:

If the arrival times of the processes are different then we have to implement the preemptive scheduling.

So, the Gantt chart may be drawn as

Process Name	CPU Burst	Priority	Arrival Time
P1	10	3	0
P2	1	1	1
P3	2	4	2
P4	1	5	3
P5	5	2	4



SCHEDULING ALGORITHMS:

So, the waiting times for the set of processes are

P1 = 0+(2-1)+(9-4) = 6 ms

P2 = (1-1) = 0 ms

P3 = 16-2 = 14 ms

P4 = 18-3 = 15 ms

P5 = 4-4 = 0 ms

The average waiting time is 6+0+14+15+0 / 5 = 7 ms

SCHEDULING ALGORITHMS:

Priorities can be defined internally or externally.

Internally defined properties are

- Time limits
- Memory managements
- Number of open files
- The ratio of average I/O burst and average CPU burst.

Internally with the help of above criteria the properties of a process can be fixed.

SCHEDULING ALGORITHMS:

External properties: The criteria to fix the priority of a process outside the operating system.

- Importance of a process
- The type and amount of fund paid to use
- The department sponsoring the work.

- A major problem in this approach is indefinite blocking of a process due to its lower priority. this is known as Starvation.
- A solution to this problem is aging. Aging is a technique for gradually increasing the priority of processes that wait in the system for a longer period.

SCHEDULING ALGORITHMS:

Round Robin Scheduling: This algorithm is especially designed for time sharing systems. It is similar to FCFS but preemption is added to enable the system to switch between processes.

- In this approach a small amount of time, called a time quantum or time slice is defined.
- In this approach the ready queue used for the scheduling is a circular queue consists of a head and tail.
- A time quantum generally ranges from 10 to 100 ms.

SCHEDULING ALGORITHMS:

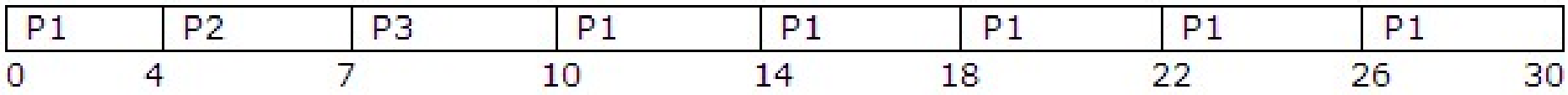
- Whenever a process is given to the CPU two things can happen that is
  - It may have a CPU burst of less than one time quantum.
  - It may have a CPU burst greater than one time quantum.
- The average waiting time under Round Robin algorithm is often long.

For example, a set of given processes with their CPU burst is:

Time quantum = 4

The Gantt chart may be drawn as

Process Name	CPU Burst
P1	7
P2	9
P3	8



SCHEDULING ALGORITHMS:

So, the Average waiting times for the set of processes are

= (10 - 4) + 4 + 7 / 3 = 5.66

- If there are n processes and q is the time quantum then no process will have to wait more then (n-1)q time units.
- The performance of a round robin algorithm totally depends up on the time quantum.
- If the time quantum is extremely large then degrades to FCFS.
- If the time quantum is extremely small the overhead of context switching increases.

SCHEDULING ALGORITHMS:

Multi level queue scheduling:

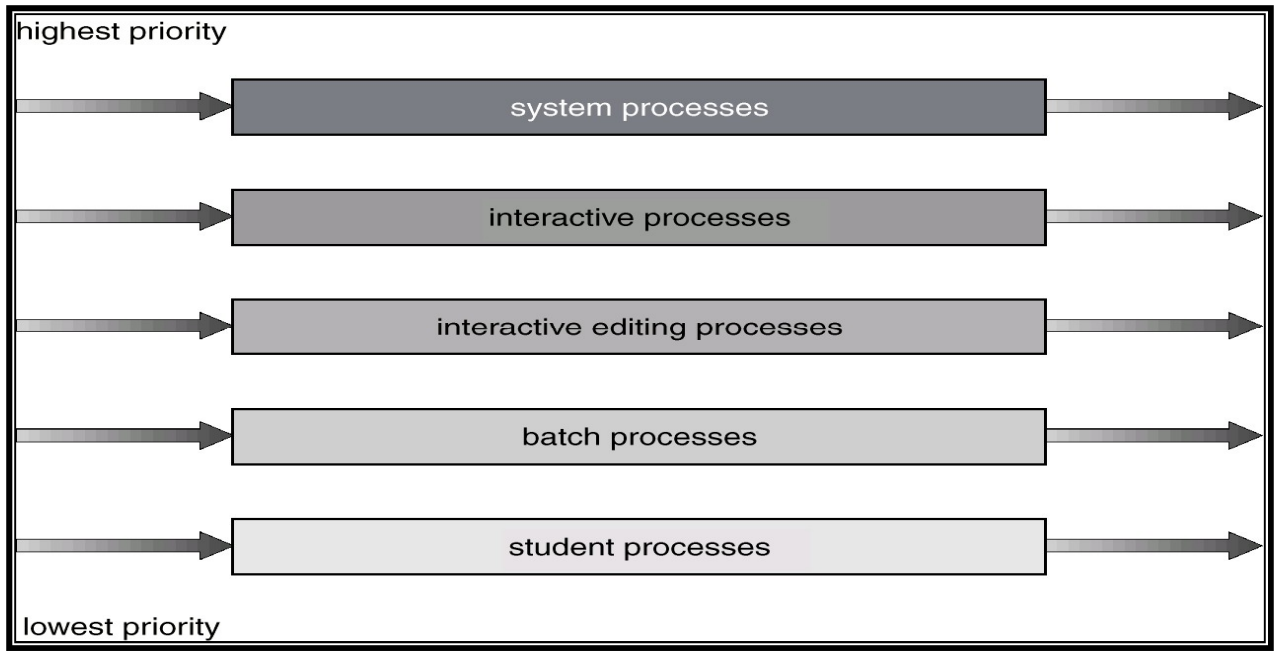
- This is used in the systems where the processes can be easily classified into different groups.

  For example, common division is made between foreground (interactive) processes and background (batch) processes. These processes may have different response time requirements and so may have different scheduling needs.

- A multilevel queue scheduling algorithm partitions the ready queue into several queues.

SCHEDULING ALGORITHMS:

- Separate algorithms may be implemented in separate queue for example a round robin algorithm may be implemented for foreground processes where as a FCFS may be used for a batch process.
- Here the queues also have to schedule according to the priority. Some of the multilevel queue is as:



SCHEDULING ALGORITHMS:

- There are two possibilities are there to run the queues that is a higher priority queue will be run first. If a system process is arrived at the time when the batch process queue is executing then that would be preempted for the system process.
- Another approach is to use a time slice that is here we can give 80% of the CPU time to the foreground processes and 20% of the CPU time can be given to the background processes. Here the queues are also scheduled as well as the processes also scheduled by different scheduling algorithms.

MULTILEVEL FEEDBACK QUEUE:

In the multilevel feedback queue scheduling algorithm, the processes are allowed to move between the queues. The processes are separated according to their CPU burst that is if a process uses too much CPU time, it will be moved to a lower priority queue and the I/O bound and interactive processes get the highest priority queue.

In addition, a process waits for a long period of time in a lower priority queue may be moved to a higher priority queue. This form of aging prevents starvation.

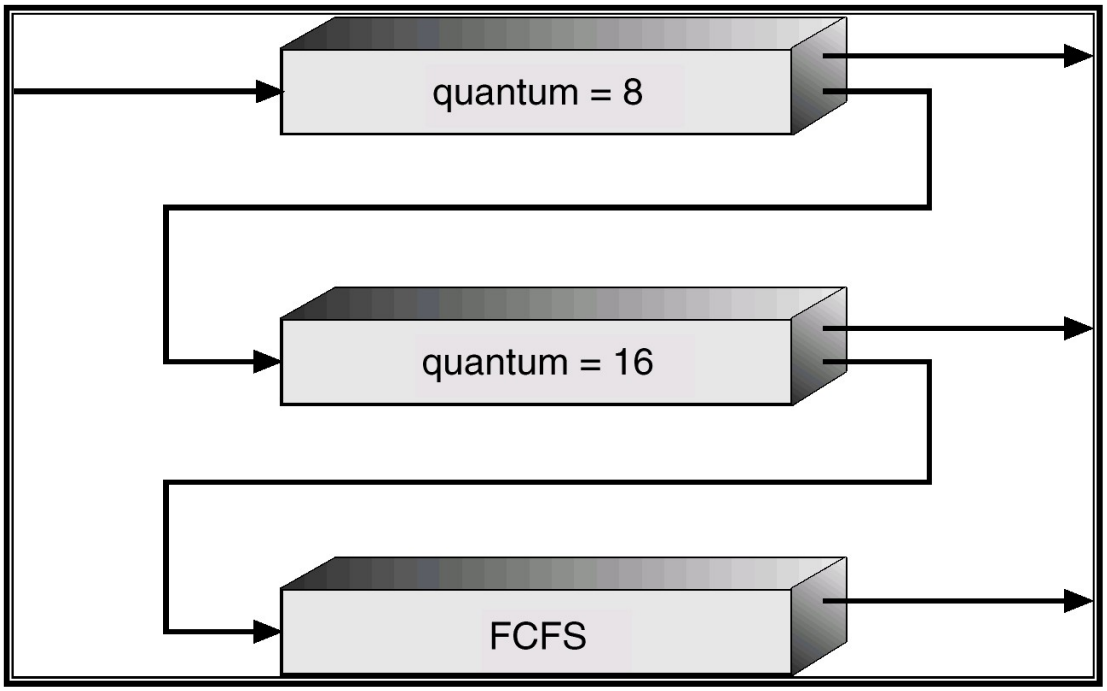
MULTILEVEL FEEDBACK QUEUE:

The general approach is, when a process enters the ready queue is put in queue 0 and a given a time quantum of 8 ms, if it does not finish within the time, it may be moved to queue 1 and given a time quantum of 16 ms and if it still doesn't finish then the process moved to the last queue and processed in FCFS manner.

MULTILEVEL FEEDBACK QUEUE:

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

- The number of queues
- The scheduling algorithm for each queue.
- The method used to determine when a process should upgrade or degrade to a highest or lowest priority queue.



THREAD SCHEDULING

The threads can be categorized into two types:

- User thread
- Kernel thread

- The kernel level threads are being scheduled by the operating system where as user level threads are managed by a thread library and kernel is unaware of them. To run on CPU user level threads must ultimately be mapped to an associated kernel level thread.

THREAD SCHEDULING

- By the help of a scheme Process Contention Scope the thread library schedules the user level threads to run on a available light weight process. To decide which kernel thread to schedule on to a CPU the kernel uses System Contention Scope.
- Process Contention Scope is done according to priority. The scheduler selects the runnable thread with the highest priority to run. User level thread properties are set by the programmers where as the kernel level threads are scheduled with the help of appropriate scheduling algorithm.

MULTIPROCESSOR SCHEDULING

If multiple CPUs are available, load sharing becomes possible however the scheduling problems becomes correspondingly more complex.

Approaches to multiple processor scheduling:

- Asymmetric
- Symmetric

- In asymmetric multiprocessing one processor will be the master or server and executes all system processes for ex. I/O and system calls etc.

MULTIPROCESSOR SCHEDULING

- In symmetric multiprocessing each processor is self scheduling. All processes may be in a single ready queue or each processor has a private ready queue.
- When a process is given to a CPU for execution it is also recorded in the cache for any future reference due to which any further request for the same process can be satisfied with the help of the cache itself, but if we are using a multiprocessor and the process is migrated to another processor of the system then it has to repopulate all data.

MULTIPROCESSOR SCHEDULING

- To avoid this type of overhead most symmetric multiprocessing systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor.
- This is known as processor affinity that is a process has an affinity for the processor on which it is currently running.

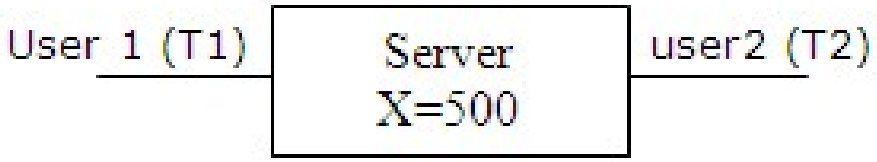
PROCESS SYNCHRONIZATION

- The concurrent processing is very much popular now a days for its efficiency that is with the help of cooperating processes we can share a common value or resource to decrease the time and cost. So, in a concurrent programming two or more processes execute in parallel. So concurrent access to shared data may result in data inconsistency.
- For example, in a server-client paradigm the value in the server may be used by different users.

PROCESS SYNCHRONIZATION

- Assume a transaction in a server with its two users.

Transaction 1	transaction 2
read(x)	read(x)
x=x-15	x=x-100
write x	write x



- If in this transaction the primary value of x is 500 and before committing one transaction both will access the value then there will be an inconsistent data

PROCESS SYNCHRONIZATION

Critical section: Each process has a segment of code, called a critical section in which the process may be changing common variables, updating a table, writing a file and so on.

- To avoid the critical section problem, we may implement mutual exclusion.
- The critical section is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. This section of code implementing this request is the entry section.

PROCESS SYNCHRONIZATION

- The critical section may be followed by an exit section; the reminder code is the reminder section.

```
do
    {
    entry section
        critical section
    exit section
        reminder section
    } while true
```

PROCESS SYNCHRONIZATION

A solution to the critical section problem must satisfy the following requirements.

- Mutual exclusion: if process Pi is executing in its critical section, then no other process can be executing in their critical section.
- Progress: if no process is in its critical section and some processes wish to enter their critical section, then they should not be in their reminder section.
- Bounded waiting: Other processes can enter their critical section for a limited time after a process has made a request to enter its critical section.

PROCESS SYNCHRONIZATION

`Peterson's solution:

- The Peterson's solution is a classic software-based solution to the critical section. It provides a good algorithmic description of solving the critical section problem and also removes some of the complexities involved in designing software that addresses the requirements of mutual exclusion.
- For example, let's take two processes named larry and jim then the algorithm may be written as

PROCESS SYNCHRONIZATION

`Peterson's solution:

Algorithm-1

Larry	Jim

While (turn!= larry)
do no operation
{
Critical section
}
 	
Turn=Jim
	While (turn!= Jim)
	do no operation
	{
	Critical section
	}
	
	Turn=Larry

PROCESS SYNCHRONIZATION

- By using this algorithm, we can see that the mutual exclusion is satisfying for both the process that is if the turn=Larry then only the process can enter into its critical section otherwise it has to wait. After completion of its task Larry sets the turn to jim before leaving
- Disadvantage: strict alternation is required that is if there is no work for the 2nd process then also the 1st process has to wait until the turn will be modified by the 2nd process.

PROCESS SYNCHRONIZATION

Algorithm-2 : To overcome the drawback of the 1st algorithm the 2nd algorithm is derived that is a process is set to false if it has no work so the strict alternation will be removed that is after execution of the 1st process the process will set its flag to false and if it wants the control again then it has to check whether the 2nd process is in its critical section or not

algorithm works properly but if a context switch occurs after verifying the flag and before setting its own flag then again it fails to satisfy the mutual exclusion, that is both the process enter into its critical section and hence a data inconsistency may arise.

PROCESS SYNCHRONIZATION

Larry	Jim

While (Jim flag=true)
do no operation
larry flag=true
{
Critical section
}
larry flag=false
	While (Larry flag=true)
	do no operation
	Jim flag=true
	{
	Critical section
	}
	Jim flag=false

PROCESS SYNCHRONIZATION

Algorithm-3 : For a solution to the 2nd algorithm, we set the flag of a process before verifying the flag of the other process

Larry	Jim
larry flag=true	Jim flag=true
While (Jim flag=true)	While (Larry flag=true)
do no operation	do no operation
{	{
Critical section	Critical section
}	}
larry flag=false	Jim flag=false

PROCESS SYNCHRONIZATION

- In this algorithm again there may be a difficulty if the context switch will occur after setting its own flag and before verifying. In this situation both the process flag will be set to true and both cannot enter into their critical section and creates a dead lock.

PROCESS SYNCHRONIZATION

Algorithm-4 : This algorithm removes all demerits of the previously discussed algorithms.

Larry	Jim
larry flag=true	Jim flag=true
turn=jim	turn=larry
While (Jim flag=true) and (turn==jim)	While (Larry flag=true) and (turn==larry)
do no operation	do no operation
{ Critical section }	{ Critical section }
larry flag=false	Jim flag=false

PROCESS SYNCHRONIZATION

Advantages: This algorithm maintains mutual exclusion at any line of context switch and it's simple to implement.

Disadvantages:

- If number of users is more than, a huge amount of time is required to check status of different flag rather executing critical section, which requires very less time to execute.
- The solution is good if the number of users is less, say 2 or 3.

PROCESS SYNCHRONIZATION

Semaphores: A semaphore S is an integer variable, that apart from initialization is accessed only through two standard atomic operations wait() and signal(). These operations are originally termed P to test and V to increment.

Wait operation: P(S)

```
While(S<=0)
    Do no operation
S=S-1
```

Signal operation(S)

```
S=S+1
```

Initial S = 1

PROCESS SYNCHRONIZATION

Note:

- When one process modifies the semaphore value no other process can simultaneously modify that same semaphore value.
- The testing of the integer value of S(S<=0), and its possible modification (S=S-1) must also be executed without interruption.

We may use the Semaphore to synchronize two processes P1 and P2, such that if we want P2 must execute after P1 then we can initialize the P2 synchronize value 0 that is it can execute only after the P1 execute the signal operation and set it to 1.

PROCESS SYNCHRONIZATION

Semaphore uses the spinlock to get its turn that is if a process is executing in its critical section, then other processes have to check in a continuous loop to know the occurrence of the signal operation. In some systems to overcome the busy waiting we can use two different states that is

- Block and
- Wake up

It will check the condition if it is less then or equals to zero then put the process into queue and the CPU scheduler wake up a process from the queue after the execution of the signal operation V(S).

PROCESS SYNCHRONIZATION

P1	P2	P3	P4	P5

P(S)
{
Critical section
}
V(S)
	P(S)
	{
	Critical section
	}
	V(S)
		P(S)
		{
		Critical section
		}
		V(S)
			P(S)
			{
			Critical section
			}
			V(S)
				P(S)
				{
				Critical section
				}
				V(S)

CLASSIC PROBLEMS OF SYNCHRONIZATION:

A lot of classical problems can be solved with the help of semaphores for example:

- Bounded buffer problem
- Dinning philosopher's problem
- Reader's writer's problem

- Bounded buffer problem: The bounded buffer synchronization problem can be best established through the producer and consumer that is if the producer and consumer will access a shared data then there may be a data inconsistency problem so this can be resolved by the semaphore as:

CLASSIC PROBLEMS OF SYNCHRONIZATION:

Producer code	Consumer code

P(mutex)
{
Critical section
}
V(mutex)
	P(mutex)
	{
	Critical section
	}
	V(mutex)

But if we communicate through buffers that is in case of a bounded buffer let us assume that a pool consists of n number of buffers each capable of holding one item. The mutex semaphores provides mutual exclusion for access to the buffer pool and is initialized to the value 1. The empty and full semaphores count the no of empty and full buffers. The semaphore empty is initialized to the value n and semaphore full is initialized to 0.

CLASSIC PROBLEMS OF SYNCHRONIZATION:

The code may be written as:

Producer code	Consumer code
P(mutex)	P(mutex)
P(empty)	P(full)
{	{
Critical section	Critical section
}	}
V(full)	V(empty)
V(mutex)	V(mutex)

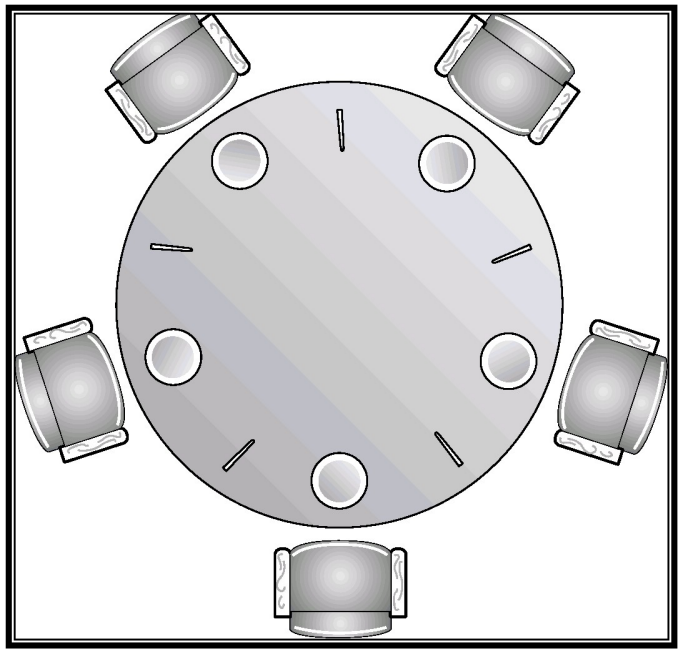
CLASSIC PROBLEMS OF SYNCHRONIZATION:

Empty=1 Full=1 Mutex=1

When it enters it sets the empty value to 0 and the mutex value 0 so that a consumer cannot enter its critical section that is it will check if the buffers are empty then only the producer can produce, after satisfaction of the conditions it will enter into the critical section and produce. After completion of its execution, it will set the full to 1 by V (full) and again set the mutex to 1. Now if again a producer tries to execute the process it will check for the conditions but the empty is set to 0 so it cannot enter where as if a consumer tries it will satisfy all its condition and can proceed with its execution.

CLASSIC PROBLEMS OF SYNCHRONIZATION:

Dinning philosopher's problem: Consider 5 philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs each belonging to one philosopher. In the centre of the table there is a bowl of rice and the table is laid with five single chopsticks. When a philosopher thinks she doesn't interact with her colleagues. From time to time a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her.



CLASSIC PROBLEMS OF SYNCHRONIZATION:

A philosopher may pick up only one chopstick at a time as she cannot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished, she puts down her chopsticks and starts thinking again.

This problem of synchronization can be better solved by the use of a semaphore that is a philosopher tries to grab a chopstick by executing a wait () operation on that semaphore and she releases her chopsticks by executing the signal operation.

CLASSIC PROBLEMS OF SYNCHRONIZATION:

The code for 5 philosophers can be written as:

P0	P1	P2	P3	P4
Think	Think	Think	Think	Think
P(S4)	P(S0)	P(S1)	P(S2)	P(S3)
P(S0)	P(S1)	P(S2)	P(S3)	P(S4)
{	{	{	{	{
eat	eat	eat	eat	eat
}	}	}	}	}
V(S0)	V(S0)	V(S1)	V(S2)	V(S3)
V(S4)	V(S1)	V(S2)	V(S3)	V(S4)

CLASSIC PROBLEMS OF SYNCHRONIZATION:

Generalization of the algorithm:

When we generalize the equation then

Let's P (i) = 0 where (I = 0 to 4)

Then the neighbor of P0 is P (S (i+4) mod 5)

That is if I = 0 then,

the neighbor is (0+4) mod 5 = 4 that is P0 and P4

CLASSIC PROBLEMS OF SYNCHRONIZATION:

Now the generalized equation using semaphore could be written as:

Thinking

P(Si)

P(S(i+4) mod 5)

{

Eating

}

V(Si)

V(S(i+4) mod 5)

Thinking

CLASSIC PROBLEMS OF SYNCHRONIZATION:

We can also design the algorithm using a mutex like:

```
Thinking
        P (mutex)
        P (Si)
        P (S (i+4) mod 5)
    {
    Eating
    }
        V (Si)
        V (S(i+4) mod 5)
        V (mutex)
Thinking
```