Syllabus
Chapter:-03

Process concept, process control block, Types of scheduler, context switch, threads, multithreading model, goals of scheduling and different scheduling algorithms, examples from WINDOWS 2000 & LINUX.

3.0 Introduction

– This chapter introduces the concept of process along with operations performed on process as creation and termination.

– It also focuses on various concepts related to process as process states and PCB.

– It explains the concept of scheduler along with its types as long term, short term and middle term and various types of queues.

– It also introduces the concept of thread with various threading models.

– It also discuss the concept of different CPU scheduling algorithms as FCFS, SJF, Priority, round robin, multilevel and multilevel feedback queue.

– At the end of this chapters numerical based on CPU scheduling asked in exam is solved.

3.1 Process

– Process is a program under execution.

– Process is considered as an instance of a program.

– Generally user writers their program in specific language such as C, C++, Java etc, which is a set of instruction.

– The program is a passive entity, which need to be compiled and translated into executable code.

– When this executable code runs, it considered as process.

– Process is consist of multiple threads, where thread is nothing but light weight process.

– Program is a passive entity but process is an active entity when it loads into memory.

– Several processes may be associated with same program eg. Opening up several instances of the same program.

– In computer system number of processes are running at the same time.

– The process memory is divided into 4 section as shown in Fig. below.

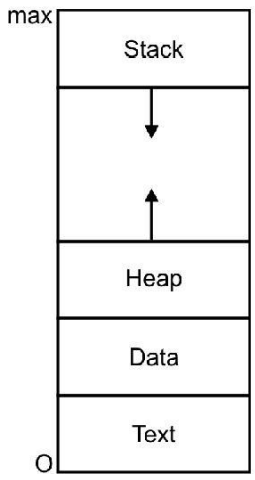


Fig. Process in memory

– Four sections are:

Text section:

This section consist of source code. It also includes the current activity as represented by value of program counter and the content of processor's register.

Stack section:

This section consist of temporary data such as function, parameters, return addresses and local variables. Stack is used for local variables, space on the stack is reserved for local variables when they are declared. Space is freed up when variables go out of scope.

Data Section:

This section stores global and static variables, which needs to be allocated and initialized prior to executing main program.

Heap section:

It is used for dynamic memory allocation and is managed via calls to new, delete malloc, free etc.

3.2 Process states

1. Explain the different states of process.

– During the process execution, it changes it's state from one state to another state.

– The state of process is defined in part by the current activity of that process.

– State of process represents it's current status.

– Each process may be in one of the following states, shown in fig below.

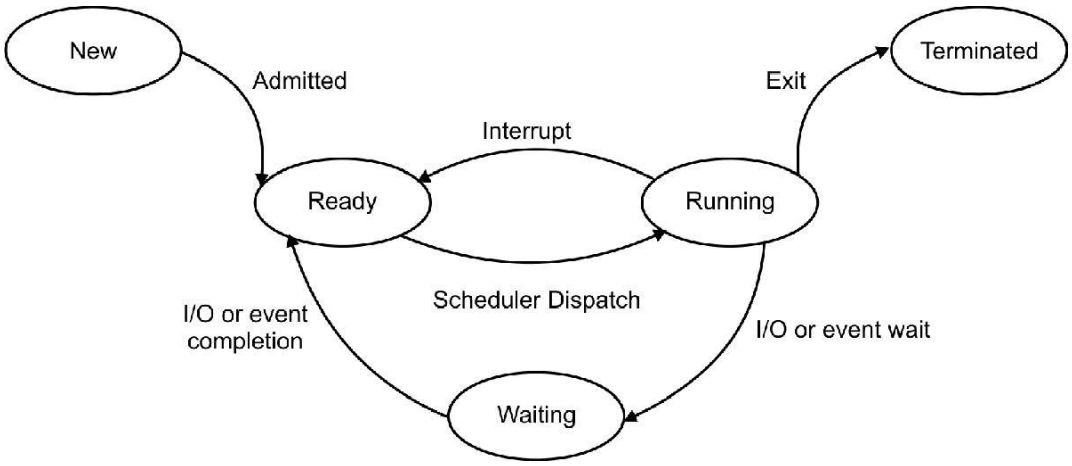


Fig. Process states

– New:

This is the first state of process in which process belongs when it is created.

– Ready:

This is the second state, in which process belongs when it is ready to execute. Scheduler pick up the process from ready queue and submit it to CPU for execution.

– Running:

This is the state where actual execution of process is carried out. From running state process may go to one of following state.

1. Terminated state: If process is successfully executed.
2. Ready state: If interrupt occurs
3. Waiting state: When any I/O event occurs.

– Waiting:

This is the state where process is waiting for some event to occur such as I/O completion or reception of a signal.

– Terminated:

Process enters into this state when it finishes its execution.

3.3 Process control Block

1. State and Explain the attributes of PCB
2. Write short note on PCB
3. Explain process control Block in details.

– The operating system is responsible for process management.

– To manage process OS needs some information related to process, which is stored in data structure, called process control block.

– Each process in operating system is represented by process control block.

– PCB is also called Task control block.

– PCB contains many pieces of information associated

With specific process shown in Fig. below.

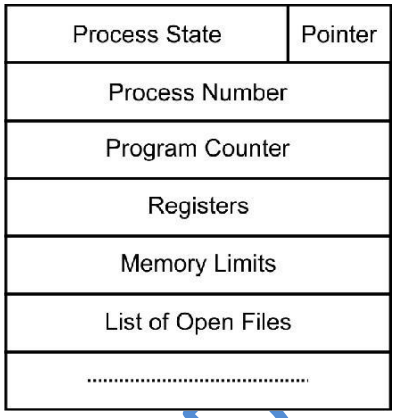


Fig. Process control Block

– Process state:

The state may be new, ready, running, waiting, and halted.

– Program counter

The counter indicates the address of next instruction to be executed for this process.

– CPU registers:

The registers vary in number and type depending on computer architecture. It includes accumulators, index registers, stack pointers and general purpose registers.

– Process number

It represents identifier of the process.

– Memory management information

It include information as value of base and limit registers, the page tables or segment table based on memory system.

– Pointer:

In include pointer to next PCB that is pointer to PCB of the next process to run.

– I/O status information:

It include the list of I/O devices allocated to process a list of open files.

3.4 Process scheduling:

Q. Explain the concept of process scheduling in O.S.

– Process scheduling is the technique adopted by OS which governs the order in which process need to be executed.

– In order to achieve multiprogramming environment in OS we requires that some process is always running, which helps in maximizing CPU utilization.

– To achieve this CPU need to switch from one process to another process.

– To keep CPU busy we need some entity that assign Job's to CPU and helps in decreasing idle times of CPU and increasing CPU utilization.

– The entity which helps to achieve this is called scheduler which select proper process from queue and submit it to CPU for execution.

3.4.1 Scheduling Queue:

– Scheduling Queue is the data structure which keeps set of processes.

– Scheduling Queue refers to queue of processor or devices.

– When the process enters into system then this process is put into queue.

– Operating system maintains different types of queues which are as follows:

1. Job queue
2. Ready queue
3. I/O device queue.

1. Job Queue:

   Initially when processes enters into system they are always put into queue called as Job queue. It consists of all processes in the system.

2. Ready Queue

   The processes that are residing in main memory and are ready and waiting to execute are kept on list called ready queue.

– This queue is generally stored as linked list.

– A ready queue header contains pointers to first and final PCB in the list.

3. Device queue:

   The list of processes waiting for a particular I/O device is called device Queue Each device has its own device queue.

– A common representation of process scheduling is a queuing diagram as shown in fig. below.

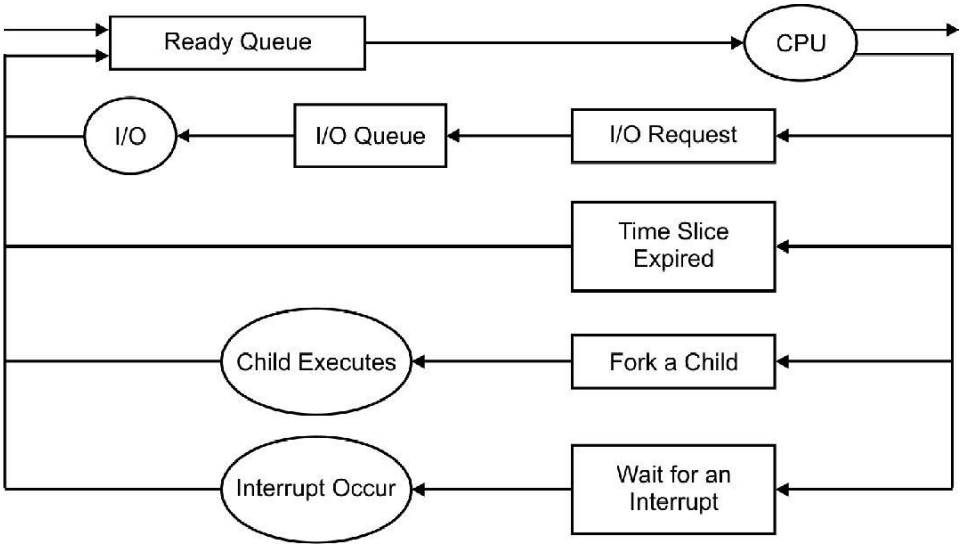


Fig. Queuing diagram representation of process

In above fig rectangular box represents a queue as ready queue and device queue. The circle represent the resources that serve the queue and arrow indicate flow of processes in the system.

3.4.2 Schedulers:

Q. What is scheduler? Explain different types of scheduler.

Q. State the purpose and functioning of short term; medium term and long term scheduler.

Q. What are the three levels of process scheduling.

Scheduler

– Scheduler is an entity responsible for arranging the processes in proper order and then submitting it to CPU for execution.

– Scheduler arranges the processes into an approximate sequence.

– As process migrates among various scheduling queues throughout it's life time, this task of migration is carried out by scheduler..

– Scheduler is an intermediate level of scheduling.

– There are different types of scheduler which carry out this activity, as mention below.

(1) Long term scheduler

(2) Short term scheduler

(3) Medium term scheduler.

1. Long term scheduler or Job scheduler

– This type of scheduler is responsible for selecting the process from Job queue to ready queue.

– Simply this scheduler pick the process from secondary memory and load it into primary memory.

– It works like a gate keeper.

– This scheduler also called admission scheduler.

– It is very important in the large system like super computer.

– it also controls the degree of multiprogramming.

– Unix system does not make use of long term scheduler.

– Long term scheduler executes less frequently.

2. Short term scheduler: Or CPU scheduler

– It select the process from ready queue that are ready to execute and submit the process to CPU for execution.

– This scheduler temporarily remove process from main memory and submit it to CPU.

– Short term scheduler makes scheduling decision more frequently.

– This scheduler can be a preemptive that means it is capable of forcibly removing process from CPU when it decide to allocate CPU to another process.

3. Medium term scheduler:

– Some of the operating system such as time sharing system may introduces an additional intermediate level of scheduling called short term scheduler as shown in Fig. below.

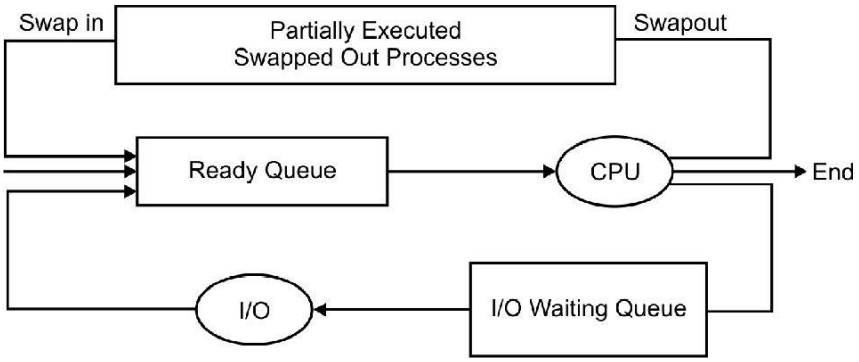


Fig. Medium term scheduling to queuing diagram.

– Medium term scheduler removes process from memory and thus reduces the degree of multiprogramming.

– This scheduler is available in all system which is responsible for swapping in and out operation, which means loading process into main memory from secondary memory and taken out process from main memory and store it into secondary memory.

– This scheduler is also responsible for swapping out a process which is not active for some time or process which has low priority.

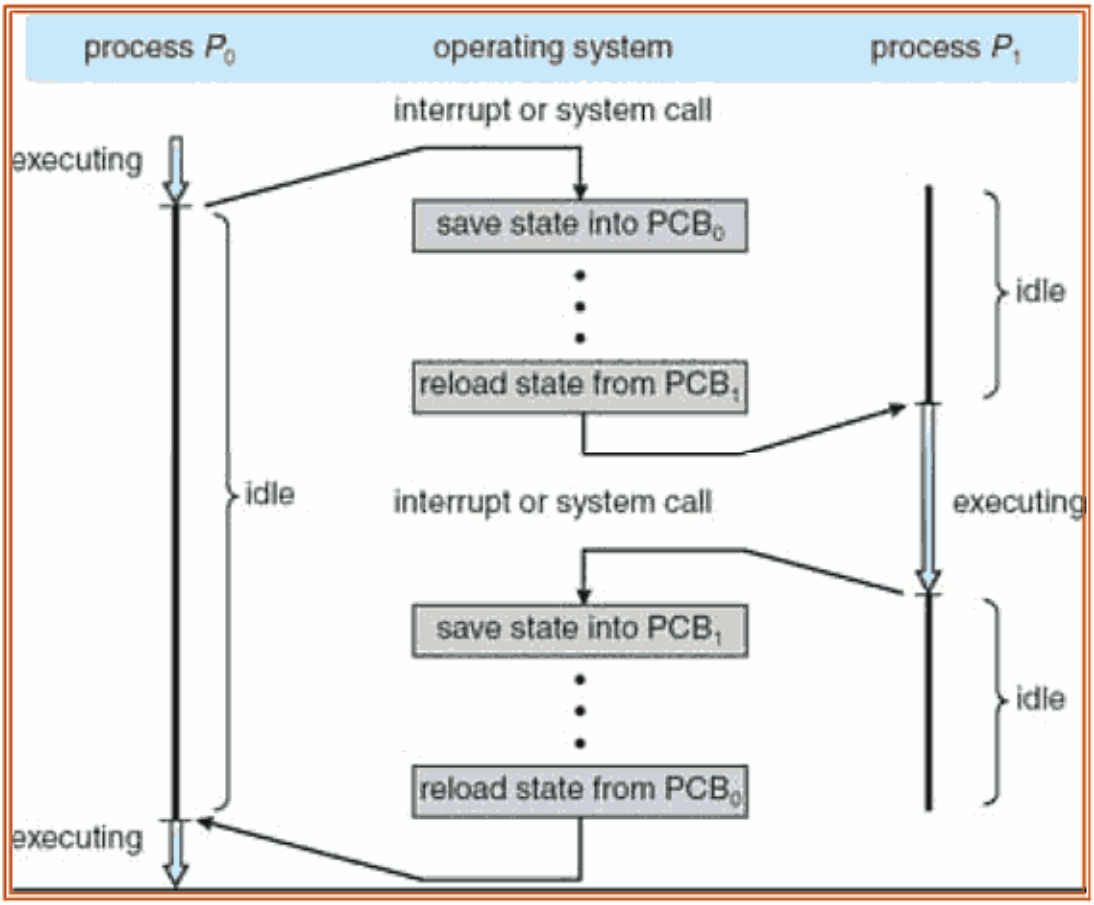
3.4.3. Context Switching

Q. Write a note on context switching. .

Q. Explain the concept of process switching.

In case of multiprogramming, the CPU switches between various processes several times, this task of switching of CPU between various processes is called context switching.

– When CPU switches from one process to another, then system must save the state of old process and load the saved state again when CPU switch back to old process.



– Context switching is done because of following reasons.

1. To maintain degree of multiprogramming and multitasking.
2. In interrupt handling.
3. In user and kernel mode switching.

5. Operations on processes:

1. Explain different operations on processes.

– In most of systems processes scan execute concurrently and they may be created and deleted dynamically.

– Thus these systems must provide a mechanism for process creation and termination, as given below.

1. Process creation:

– The process may create several new processes, via create process system call during course of execution.

– The creating process is called parent process and created process is called children of creating process.

– Each of these newly created process may in turn create other processes forming a tree of processes.

– Child process may obtain it's resource directly from O.S.

– Most of OS like Unix and windows identify processes according to unique process id.

– When one process creates new process, then two possibility exist in terms of execution.

1. The parent continues to execute concurrently with it's children.
2. The parent wait until some or all of it's children have terminated.

Process creation using fork ( ) system call, shown below.

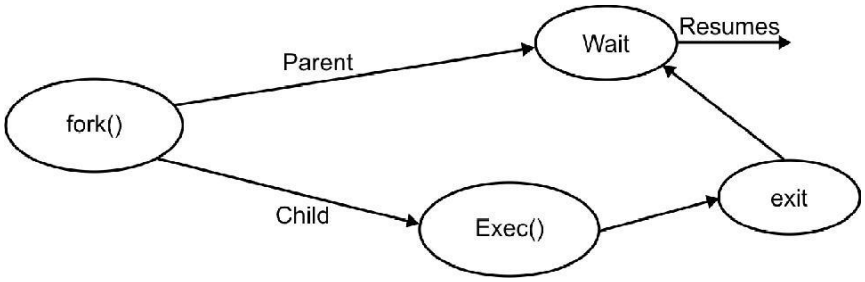


Fig. Process creation using fork ( ) system call

2. Process Termination:

– A process terminates when it finishes executing it's final statement and asks the OS to delete it by using exit ( ) system call.

– At that point, the process may return a status value to it's parent process via wait system call.

– All the resources of process including physical and virtual memory, open files and I/O buffers are deallocated by OS.

– Termination occurs in other situation also.

– A process can cause the termination of another process via an appropriate system call; terminate process ( ) .

– Such a system call can be invoked only by parent of process that is to be terminated.

– User could arbitrarily kill each other's Job.

– A parent may terminate execution of one of it's children for variety of reasons as.

1. Child has exceeded it's usage of some of the resources that it has been allocated.
2. The task assigned to the child is no longer required.
3. Parent is exiting and OS does not allow a child to continue if its parent terminates.

3.5.1 Process Vs program

Q. Differentiate between program and process.

Program	Process
1. It's a set of instructions which is written in specific language.	1. It's a program under execution.
2. It's a passive entity.	2. It's a active entity.
3. Program does not have program counter and I/O status.	3. Process has program counter, and I/O status.
4. Program doesn't have any state.	4. Process has different states as new, running etc.
5. Program doesn't have any section.	5. Process has sections as Text section, stack section, data section and Heap section.
6. Program does not require these resources.	6. Process requires resources such as CPU time, memory etc.

3.6 Interprocess communication (IPC)

Q. What is Interprocess communication.

– Interprocess communication is a mechanism which allow one process to communicate with another process.

– Two processes which want to communicate with each other may resides on same computer or on different computer connected through network.

– It enables one application to control another application and for several application to share same data without interfering with one another.

– IPC is required in all multiprocessing system.

– IPC is not generally supported by single process operating system such as DOS.

– Generally processes executing concurrently in OS may be either independent or co-operating.

– A process said to be independent if it cannot affect or be affected by other process executing in system.

– Independent process never share common data.

– A process said to be cooperating if it can affect or be affected by other process executing in the system.

– Cooperating process share data with other processes.

– There are few reasons for process cooperation as follow.

1. Information sharing:

   In number of users want to access same instance as shared life then we must provide an environment which are concurrent access.

2. Computation speed up:

   If we want a specific task to run faster, we must use divide and conquer policy that is breaking one task into number of subtask and executing it in parallel and then again combine it can speed up the computation.

3. Modularity:

   If we want to construct system in modular fashion then we must divide system functions into separate processes or threads.

4. Convenience:

   Even an individual user may work on many task at same time for eg. user may be editing, printing and compiling in parallel.

– Cooperating processes requires an Interprocess communication which allow them to exchange data and information.

– Interprocess communication can be provided with two models as.

1. Shared memory.
2. Message passing

6.1 Shared Memory system

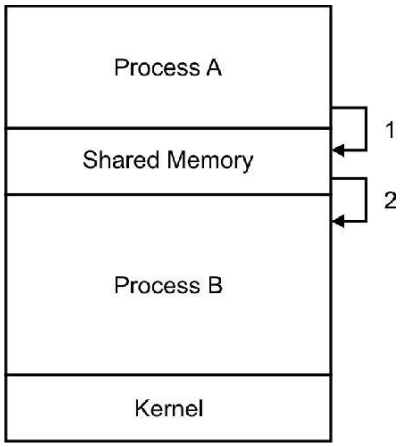


Fig.

– Interprocess communication can be provided with the help communication model called shared memory as shown in fig. above.

– This type of communication model requires that the communicating process needs to establish region of shared memory.

– This shared memory region resides in the address space of the process creating shared memory segment.

– Other processes that wish to communicate using this shared memory segment must attach it to their address space.

– Generally OS tries to prevent one process from accessing another process memory.

– But shared memory model requires that two or more processes agree to remove this restriction.

– Two processes can exchange data by reading and writing it to shared areas, here process A and B can communicate with each other through shared memory.

– But at the same time 2 process must ensure that they are not writing to same location simultaneously.

– This type of model is beneficial for tightly coupled system like parallel processing, where main memory is shared by more than one processor.

3.6.2. Message passing system

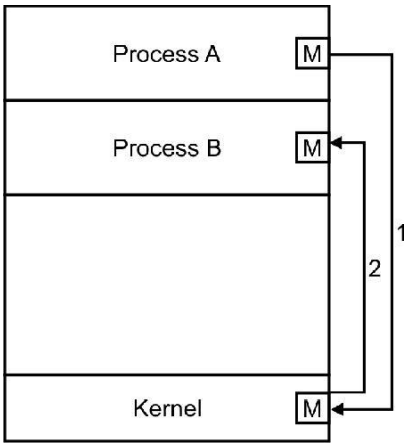


Fig.

– The cooperating process can communicate with each other by means of another way than shared memory is message passing system.

– A message passing mechanism can be provided with help of two operations as send (message) and receiver (message).

– Here process A and B want to communicate with each other so they send message to and receive messages from each other and also they should establish communication link between them.

– The link can be implemented in a variety of ways.

– A message passing provides a mechanism to allow processes to communicate and to synchronize their action without sharing the same address space and is particularly useful in a distributed environment.

– Here communicating process may reside on different computers connected by network.

– Some of the issues in it as follows:

1. Naming

– In order to communicate with two processes, they can use direct or indirect communication.

– In direct communication we must need to provide explicitly name of both sender and receiver.

3.7 Thread

Q. What is thread? Explain various benefits of using it.

– Thread is a light weight process.

– It is consider as basic unit of CPU utilization.

– Thread shares common data.

– Thred consist of program counter, thread ID, register set and stack.

– One thread can share data section, code section and other resources as open files with another thread of same process.

- Threads does not require IPC.
- Thread is a smallest unit of scheduling that can be scheduled by an OS.
- A traditional or heavy weight process has single thread of controls.
- It process has multiple threads of control, it can perform more than one task at a time.
- Below fig shows difference between single threaded and multithreaded process.

Properties of a Thread

- Only one system call can create more than one thread (Lightweight process).
- Threads share data and information.
- Threads shares instruction, global and heap regions but has its own individual stack and registers.
- Thread management consumes no or fewer system calls as the communication between threads can be achieved using shared memory.
- The isolation property of the process increases its overhead in terms of resource consumption.

Similarities:-

1. Like processes threads share CPU and only one thread active (running) at a time.
2. Like processes, threads within processes, threads within a processes execute sequentially.
3. Like processes, thread can create children.
4. And like process, if one thread is blocked, another thread can run

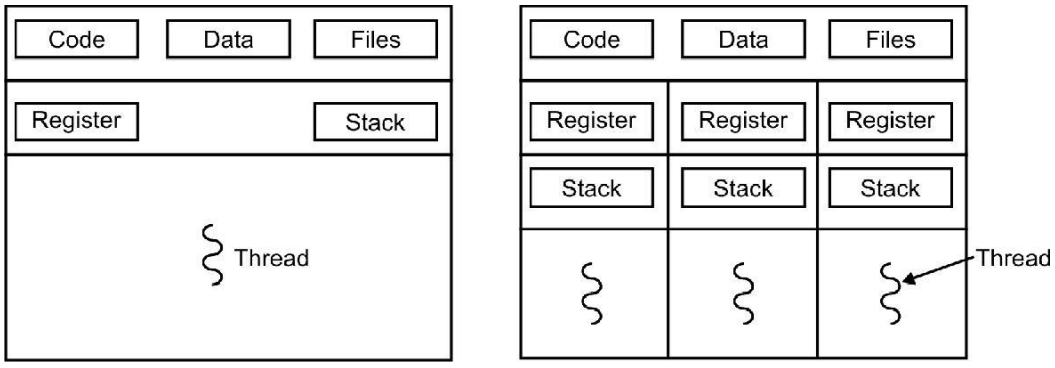


Fig.

For example in word processor, a background thread may check spelling and

grammar while foreground thread processes user input.

2. Synchronization:

   Message passing may be either blocking or non-block also known as Synchronous and Asynchronous.

– Blocking send:

  The sending process is blocked until message is received by receiving process.

– Non-blocking send

  The sending process sends the message and resume operation.

– Blocking Receiver

  The receiver blocks until a message is available.

– Non-blocking Receive:

  The receiver retrieves either a valid message.

3. Buffering:

   Basically for communication of 2 processes the messages are kept in some queue as

– Zero capacity:

  The queue has maximum length of zero, so no message is kept in it.

– Bounded capacity

  In this case queue has finite length of n, so at most n messages can reside in it.

– Unbounded capacity

  In this case queue has infinite length so any number of messages can reside in it.

Fig. 1 shows that one process have only single thread which share code, data and file of parent process.

Fig. 2 shows that each thread have their register, stack but it share code, data and file of parent process.

3.7.1. Benefits of Thread:

The benefits of multithreaded programming can be divided into four categories.

– Responsiveness

  One thread may provide rapid response while other threads are blocked or slow down intensive calculations.

– Resource sharing

  By default threads share common code, data and other resources, which allows multiple task to be performed simultaneously in a single address space.

– Economy

  Creating and managing threads along with context switch between them is much faster than performing the same task for processes.

– Scalability

The benefits of multithreading can be greatly increase in multiprocessor architecture, where threads may be running in parallel on different processors.

3.7.2 Thread Vs Process

Q. Differentiate between process and thread.

Process	Thread
1. Process is a smallest scheduling entity in multiprocessing.	1. Thread is smallest entity in multithreading.
2. Process is a program in execution.	2. Thread is a path of execution within process.
3. It is use to execute large heavy weight jobs such as running different applications.	3. It is used to carry out much smaller or light weight job as auto saving document in program.
4. Each process runs in a separate address space in CPU.	4. Thread may share address space with other threads within same process.
5. Switching between processes is some how difficult compare to thread.	5. Switching between threads is simpler and faster than process.
6. Process only have control over child processes.	6. Threads have great degree of control over other threads of same process.
7. Changes made to parent processes do not affect the child process.	7. Any change made to main thread may affect behavior of other threads within same process.
8. Cooperating process require IPC.	8. Thread does not require IPC.

BASIS FOR COMPARISON	PROCESS	THREAD
Basic	Program in execution.	Lightweight process or part of it.
Memory sharing	Completely isolated and do not share memory.	Shares memory with each other.
Resource consumption	More	Less
Efficiency	Less efficient as compared to the process in the context of communication.	Enhances efficiency in the context of communication.
Time required for creation	More	Less
Context switching time	Takes more time.	Consumes less time.
Uncertain termination	Results in loss of process.	A thread can be reclaimed.
Time required for termination	More	Less

3.7.3 User Level thread VS Kernel level thread

Q. Differentiate between user level thread and kernel level thread.

User Level Thread	Kernel Level Thread
1. User level threads are faster to create and manage.	1. Kernel level threads are slower to create and manage.
2. User level thread can run on any OS.	2. Kernel level threads are specific to O.S.
3. Scheduling is fair because of total user control.	3. Scheduling is unfair because OS decide about scheduling.
4. User threads need to be associated with process.	4. Kernel thread need not be associated with processes.
5. It is efficient.	5. It is inefficient.
6. It does not support multithreading and does not get large time as per requirement.	6. It support multithreading and it gets lot of time as per requirement.
7. All activity is carried out at user level only.	7. All activity is carried out at kernel side.
8. User level threads are created by user.	8. Kernel level threads are supported directly by OS.
9. eg. Posix, pthread, mach, c-thread.	9. e.g. windows NT, solaris, UNIX.

3.8 Multithreading Models

Q. Explain the various multithreading model for thread.

Q. Define different model of multithreaded system.

Some operating system supports user level while some supports kernel level threads but few OS supports both.

– User threads are supported above kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by O.S.

– A relationship must exist between user threads and kernel threads which can be shown with help of following models.

1. Many to one model
2. One to one model.
3. Many to many model.

1. Many to one model

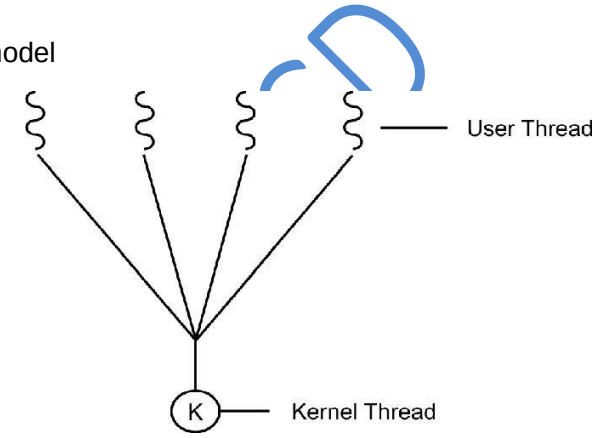


Fig. Many to one model

– In this type of model user level threads are mapped to one single kernel level thread.

– Thread management is done by thread library in user space.

– This model is more efficient.

– Entire process will block if a thread makes a blocking system call.

– Also because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.

– Only one kernel thread will manage all user level thread one to one model

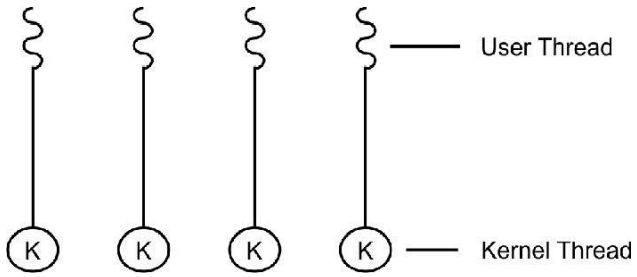


Fig. One to one model

2. One to one model

– In this type of model one user level thread map to only one thread of kernel level.

– It provides more concurrency than many to one model by allowing another thread to run when thread makes blocking system call.

– It also allows multiple threads to run in parallel on multiprocessor.

– The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.

– The overhead of creating kernel threads can burden the performance of an application.

– Most implementation of this model restrict the number of threads supported by system.

3. Many to many model

– The many to many model multiplexes many user level threads to a smaller or equal number of kernel threads.

– The number of kernel threads may be specific to either a particular application or a particular machine.

– This model also support to create many user threads as per their requirement.

– Here concurrency is not gained because kernel can schedule only one thread at a time.

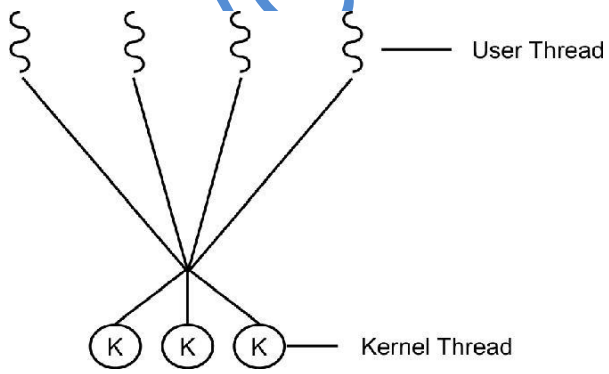


Fig. many to many model

3.9 CPU scheduling or process scheduling

Q. What is process scheduling.

– CPU scheduling is the basis of multiprogrammed OS, by switching CPU among process, OS can make computer more productive.

– Most of time CPU remains idle because process is not allotted to it. So effective CPU scheduling can overcome this problem.

– A scheduling mechanism allows one process to use the CPU while another is waiting for I/O.

3.9.1 Preemptive and Non preemptive scheduling.

Q. What is preemptive and Non Preemptive scheduling.

1. Preemptive scheduling:

– It allows a pending high priority job to preempt a running Job of lower priority.

– Lower priority Job is suspended and is resumed as soon as possible.

– Use preemptive scheduling if you have long running, low-priority jobs causing high priority Jobs to wait for long time.

– Preemptive scheduling takes effect when two jobs compete for same job slots.

– Preempted Job is resumed as soon as more job slots become available.

In this case new process selected to run when

– An interrupt occurs

– When new processes become ready.

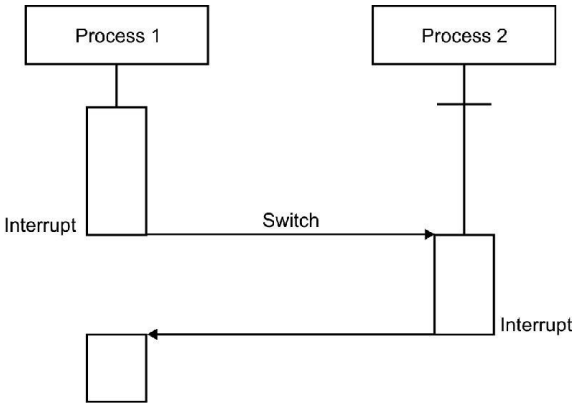


Fig. Preemptive scheduling

2. Non preemptive scheduling

– Once a process enters a running state, it is not removed from the processor unit it has completed it's service time.

– A new process is selected to run either

  – when a process terminates

  – when an explicit system request causes a wait state.

– Non preemptive scheduling algorithm is simple to implement and cost is less.

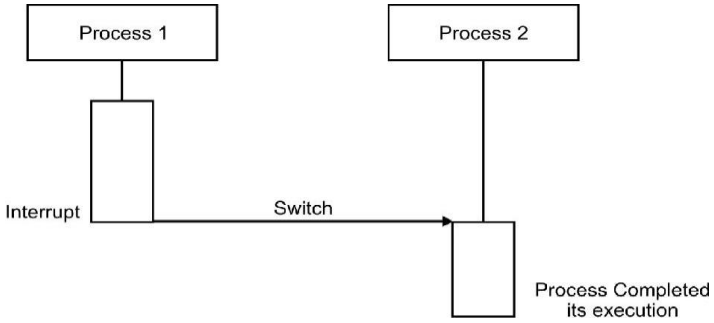


Fig. Non preemptive scheduling

– Above fig shows non preemptive scheduling in which CPU switches.

From process 1 when it completes it's execution to processes 2.

– In this case even if higher priority process enters the system then running process cannot be forced to give up the control.

CPU scheduling decision takes place under one of four conditions.

1. When a process switches from running state to waiting state such as for an I/O request.
2. When a process switches from running state to ready state such as for an interrupt occurs.
3. When a process switches from waiting state to ready state after completion of I/O.
4. When a process terminates.

Scheduling take place under condition (1) & (4) said to be non preemptive.

3.9.2 Dispatcher

Q. Explain the concept of dispatcher.

– Dispatcher is one of the component involved in the CPU scheduling.

– Dispatcher is the module that gives control of CPU to process selected by the scheduler.

– The dispatcher needs to be as fast as possible, as it runs on every context switch.

– The time consumed by dispatcher is known as dispatch latency.

– Functions of dispatcher as follows:

  – switching context

  – switching to user mode

  – Jumping to proper location in the newly loaded program.

– Dispatcher is a piece of code remain in kernel.

– Dispatcher helps to save current process state and reload, so that next time it loads exactly at same location.

– Dispatcher should be as fast as possible as switching is must.

3.9.3 CPU scheduling criteria

Q. State and Explain in brief the various scheduling criteria.

– Different scheduling algorithms have different features which allows to select specific process, in order to submit process to CPU.

– There are different criteria to consider when trying to select the best scheduling

algorithm for a particular situation and environment including.

1. CPU utilization:

– We want to keep CPU as busy as possible. Generally CPU utilization can range from 0 to 100%.

– In real system it should range from 40% to 90%

2. Through put

– Number of processes that are completed per unit time called throughput.

– For long processes, this rate may be one process per hour.

– four short transaction, it may be ten processes per second.

3. Turnaround time:

– It's time required for a particular process to complete from submission time to completion.

– Turnaround time is the sum of period spent while waiting to get into memory waiting in the ready queue and executing in CPU or I/O devices.

4. Waiting time:

– It's a amount of time spent in waiting at ready queue by process. Before submitting it to CPU.

– CPU-scheduling algorithm does not affect the amount of time during which process executes.

– It affects only amount of time that process spends while waiting in the ready queue.

5. Response time

– It's a time from the submission of request until the first response is produced.

– Actually it's a time it takes to start responding not the time it takes to output the response.

– Generally out of above five mention scheduling Criteria, the CPU utilization and throughput should be maximize and turnaround time, waiting time, response time should be minimized.

3.10 Process scheduling Algorithms

Q. Explain different CPU scheduling algorithm.

– The main problem is to decide which of the process is to be allocated to the CPU.

– So in order to overcome the above problem, various scheduling algorithm have been implemented which are as follow:

1. FCFS
2. SJF
3. Priority scheduling.
4. Round–Robin scheduling
5. Multilevel Queue scheduling

FCFS (First come first serve scheduling)

- FCFS is very simple algorithm
- In this scheme, the process that request the CPU first is allocated the CPU first.
- The implementation of FCFS policy is easily managed with FIFO queue.
- When CPU is free, it is allocated to process at the head of queue.
- The average waiting time under FCFS policy s often quite long.
- Consider following three processes

Process	Burst time
P_1	24
P_2	3
P_3	3

Gantt chart for above table is

avg waiting time for $P_1 = 0$ $P_2 = 24$ and $P_3 = 27$ so

avg waiting time for three process is

(0 + 24 + 27) / 3 = 17 MS.

- The average waiting time under an FCFS policy is not minimal and it may vary if process burst time varies.
- FCFS is a non preemptive scheduling algorithm.
- FCFS suffer from convey effect as all other process wait for one big process to get out of the CPU.
- This effect results in lower CPU and device utilization.

2. Shortest Job First scheduling.

- Another CPU scheduling algorithm than FCFS is shortest Job First.
- In this scheme process having shortest burst time gets CPU first than other process.
- If burst time of two process are same then FCFS scheduling is used to break the tie.
- More appropriate term for this scheduling method is the shortest remaining time first.
- This algorithm is available in both term as non preemptive and preemptive.
- This algorithm have advantage over FCFS, that is it minimized the average waiting time.
- It is considered as optimal with respect to average waiting time.

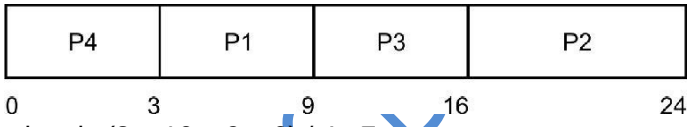
– It helps to decrease convey effect cause in FCFS, as it give chance to process having shortest burst time.

– Problem with this algorithm is that it cannot be implemented.

– It also require future knowledge to check which process have less burst time than other. Example SJF Non preemptive

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Gantt chart

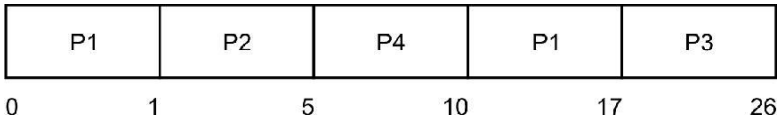


Avg waiting time is (3 + 16 + 9 + 0) / 4 =7ms

Example SJF preemptive (SRTF) shortest remaining time first.

Process	Arrival time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Gantt chart



avg waiting time =[(10 – 1) + (1 – 1) + (17 – 2) + (5 – 3)] / 14 = 6.5 ms.

Here intially P_1 gets CPU as it arrive at time 0 but P_1 gets preempted by P_2 as P_1 have 7 ms left and P_2 has 4 ms.

3. Priority scheduling:

– SJF is a special case of the general priority scheduling algorithm.

– In this scheme each process is assigned with priority and CPU is allocated to process with highest priority.

– Equal priority processed are scheduled in FCFS order.

– SJF uses the inverse of next expected burst time as it's priority, smaller the expected burst higher will be priority.

– Priorities are implemented using inyegers within fixed range.

– Priority scheduling can be either preemptive or non preemptive.

– Priorities can be assigned internally or externally.

– Internal priorities are assigned by OS using criteria as average burst time, system resource use and other factors available to kernel.

– External priorities are assigned by users, based on importance of process.

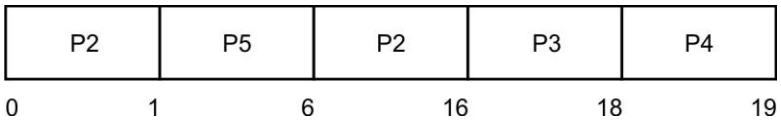
– Priority scheduling can suffer from problem of starvation, as low priority process has to wait for long time.

– Solution to above problem is aging in which priority of process is increase so that low priority process get chance to execute.

e.g.

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Gantt chart is



Avg waiting time is (6 + 0 + 16 + 18 + 1) / 5 =8.2ms

4. Round Robin scheduling (RR)

– Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called time quantum.

– This algorithm is designed specially for time sharing system.

– In this algorithm preemption is added to enable system to switch between processes.

– In this scheme every process gets equal change to run.

– To implement RR scheduling, we keep ready queue as FIFO queue of processes.

– New processes are added to tail of the ready queue.

– The CPU scheduler picks the first process from ready queue and set a timer to interrupt after 1 time quantum and dispatch the process.

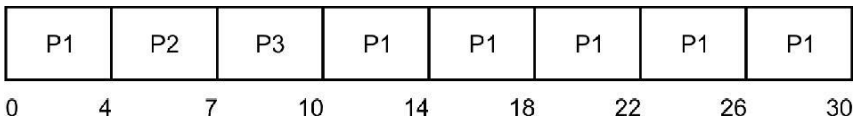
– The performance of RR is depend on time quantum selected.

– If time quantum is larger, then RR reduces to FCFS algorithm.

– If it is very small, then each process gets I/nth of processor time and share the CPU equally. Example:

Process	Burst time
P_1	24
P_2	3
P_3	3
Time quantum = 4ms	

Gantt chart



waiting time for P_1 = 6

P_2 = 4

P_3 = 7

avg waiting time is = (6 + 4 + 7) / 3 = 5.66ms

5. Multilevel queue scheduling: [S–12, 3M]

– This is another type of scheduling algorithm created for situation in which processes are easily classified into different groups.

– Two types of processes are foreground (interactive) processes and background (batch) processes.

– These two types have different response time and scheduling need.

– This algorithm position the ready queue into several separate queues, as shown in fig. below.

System processes highest priority

process Interactive processes

Interactive editing

processes Batch processes

student process Lowest priority

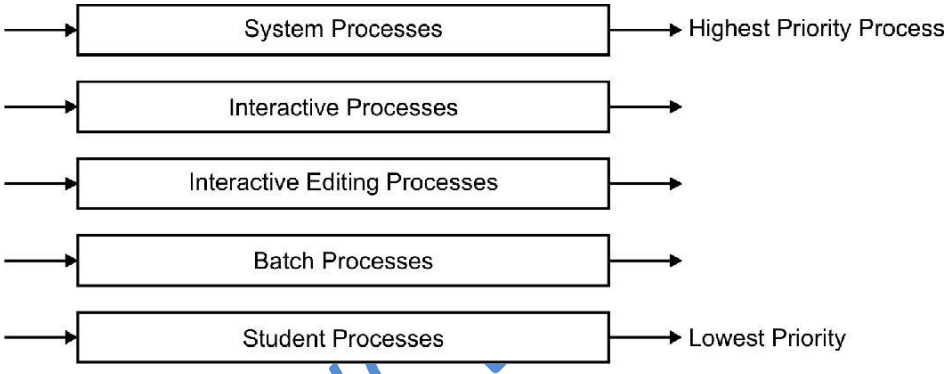


Fig. Multilevel queue scheduling

– In this case processes are permanently assigned to one queue, based on some property of process as memory, size process priority and process type.

– Here Each queue has its own scheduling algorithm.

– In this scheme scheduling must also be done between queues, that is scheduling one queue to get time relative to other queues.

– In this algorithm job can not switch from queue to queue.

– Separate queue used for foreground and background processes.

– No process in interactive queue could run until system queue gets empty.

6. Multilevel Feedback Queue scheduling. [S-09 3M]

– This algorithm is very much similar to ordinary multilevel queue scheduling, except job may be move from one queue to another for variety of reasons, as.

1. If the characteristics of job change between CPU intensive and I/O Intensive.
2. A job that has waited for a long time can get bumped into higher priority queue.

– Multilevel feedback queue scheduling is most flexible, as it can be tuned for any situation.

– It is most complex to implement because of all the adjustable parameters.

Some of the parameters which define one of these system include.

1. Number of queues.
2. Scheduling algorithm for each queue.
3. The methods used to upgrade or demote processes from one queue to another.
4. Method used to determine which queue a process enters initially.

Quantum = 8

Quantum = 16

FCFS

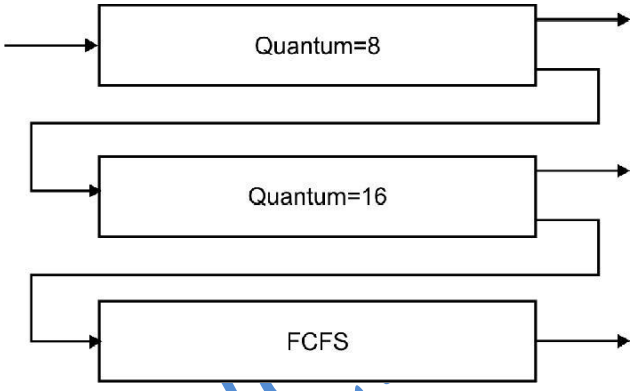


Fig. Multilevel Feedback Queues