Page No.
Date

# Unit 3: Process Management

## * Process -

- Process is defined as program under execution which competes for the CPU time & other Resources.
- A running instance of a program is called a process.
- A process is a smallest unit of work that is scheduled by operating system.
- A process needs resources such as CPU time, memory, files & input/output devices to accomplish its task.
- This resources are allocated either when the program is created or when it is executing.
- Process is also called as job, task & unit of work. the execution of process must be in sequential fashion i.e. at any time at most one instruction is executed.
- Each process has following sections
  1) Text Section - Contains the program code.
  2) Data Section - Contains global & static variable.
  3) HEAP memory allocation - it is dynamic in nature & managed via calls to new, delete, free, etc.
  4) Stack memory allocation - it is used for local variable. it contains temporary data.

Page No.
Date

5) Program counter - it contains processors registers.
- As the program executes the process changes state. the state of a process can be new, active, waiting or halted.

| Process | Program |
|---|---|
| 1) Process is a program in execution. | 1) A program is a series of instructions to perform particular task. |
| 2) Process is stored in memory. | 2) Program is stored in secondary storage. |
| 3) Process changes its states. | 3) Program occupy fixed place. |

## * Process States :-

- In a multiprogramming System, many processes are executed by the OS. But, at any instant of time only one process executes on the CPU. Other processes wait for their turn.
- The current activity of a process is known as its state. As a process executes, it changes state. The process state is an indicator of the nature of the current activity in a process.

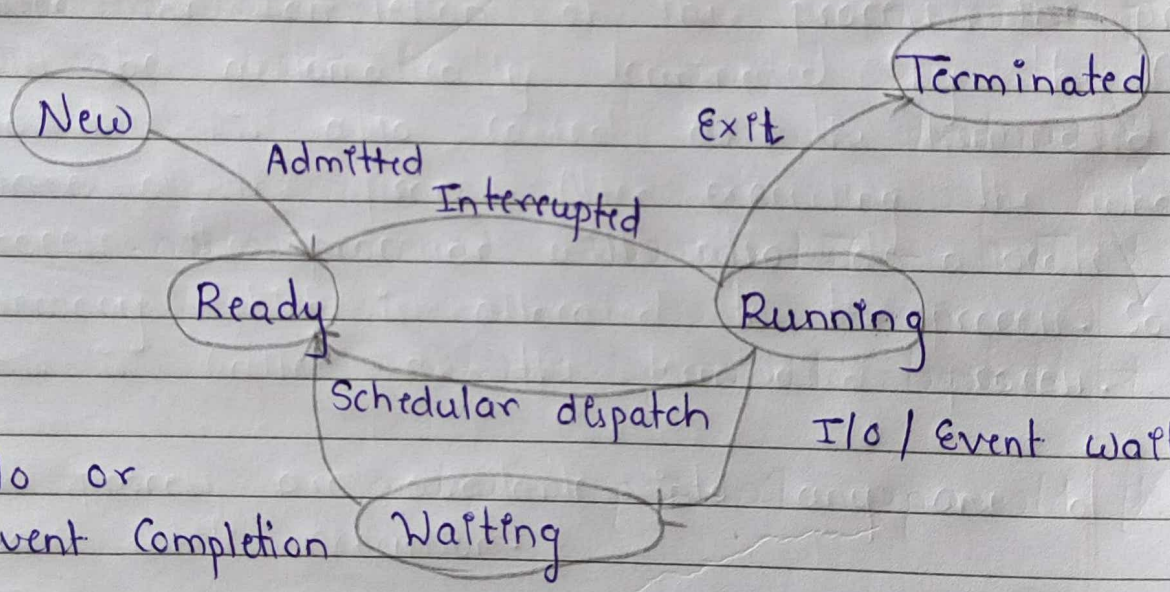



Fig. Process States & process transition.

- Above fig. shows process state diagram. A state Diagram represents the different states in which a process can be at different times, along with the transitions from one state to another that are possible in the OS.

- Each process may be in one of the following States.

1] New State :-
A process that has just been created but has not yet been admitted to the pool of execution processes by the OS. Every new operation which is requested to the system is known as the new born process.

2] Ready State :-
When the process is ready to execute but it is still waiting for the CPU to execute then this is called as the ready state. After the completion of the input and outputs the process will be on ready state means the process will wait for the processor to execute.

3] Running State :-
The process that is currently being executed. When the process is running under the CPU, or when program is executed by the CPU, then this is called as the running state process and when a process is running then this will also provide us some outputs on the screen.

4] Waiting or Blocked :-
A protes that cannot execute until some event occurs or an I/o completion. When a process is waiting for some inputs & output operations then this is called as the waiting state. And in this state process is not under the execution instead the process is stored out of memory and when the user will provide the input then this will again be on ready state.

5] Terminated State :-
After the completion of the process, the process will be automatically terminated by the CPU, So this is also called as the terminated state of the process. After executing the whole process the processor will also de-allocate the memory which is allocated to the process. So this is called as the terminated process.

- A process are available into any one of the state while execution.

| State / Activity | Description. |
|---|---|
| New | Process is being created. |
| Running | CPU is executing the process's instructions. |
| Waiting | Process is well, waiting for an event, typically I/o or signal. |
| Ready | Process is waiting for a processor |
| Terminated | Process is done running. |

Page No.
Date

## ★ Process Control Block (PCB)

1) Each process is represented in the operating system by a process control Block (PCB) also called as Task control Block (TCB).
2) The operating system groups all information on that needs about a particular process into a data structure called a PCB or process descriptor.
3) When a process is created, operating system creates a corresponding PCB & released whenever the process terminate.
4) A PCB stores descriptive information pertaining to a process, such as its state, program counter, memory management information, information about its scheduling, allocated resource accounting information, etc. that is required to control & manage a particular process.
5) Following are the sections of PCB.

| Pointer | Process State |
|---|---|
| Process Number | |
| Program Counter | |
| CPU Registers | |
| Memory Allocation | |
| Event information | |
| List of open files | |
| : | |
| . | |

fig. Process Control Block (PCB)

1. Process Number :-
Each process is identified by its process number, called Process Identification Number (PID). Every process has a unique process ID through which it is identified. The process-id is provided by OS.

2. Priority :-
Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services process priority is the preference of the one process over other process for execution. This field stores the priority of particular process.

3. Process State :-
This information is about the current state of the process. The state may be new, running, & waiting, halted and so on.

4. Program Counter :-
The counter indicates the address of the next instruction to be executed for this process.

5. CPU Registers :-
The registers vary in number & type, depending on the computer architecture. They include accumulators, index registers, stack pointers, & general-purpose registers, plus any condition-code information.

6. CPU Scheduling Information :-
This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

7. Memory Management Information :-
This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by operating system.

8. Accounting Information :-
This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, & so on.

9. I/O status Information :-
This information includes the list of I/O devices allocated to the process, a list of open files, & so on.

Page No.
Date

10. File Management :-
It includes information about all open files, access rights etc.

11. Pointer :-
Pointer points to another process control block. Pointer is used for maintaining the scheduling list.

- The PCB simply serves as the repository for any information that may vary from process to process.

## 3.1.5 Operations on Processes

(W-16, 17, 18)

- There are innumerable operations or tasks that can be performed on processes, such as creating, terminating, suspending or resuming a process etc. To successfully execute these operations/tasks, the operating system provides run-time services (or system calls) for the process management.
- In this section, we will discuss only process creation and termination operations of processes. The processes in the system can execute concurrently and they must be created and deleted dynamically.

**1. Process Creation:** (W-1

- When a new process is to be added to those currently being managed, the operating system builds the data structures that are used to manage the process and allocates address space in main memory to the process. This is the creation of a new process.
- There are four principal events that cause processes to be created:
  - (i) **System Initialization:** When an OS is booted, typically several processes are created.
  - (ii) **Execution of a Process Creation System Call by a Running Process:** Often a running process will issue system calls to create one or more new processes to help it do its job. Creating new processes is particularly useful when the work to be done can easily be formulated in terms of several related, but otherwise independent interacting processes.
  - (iii) **A User Request to Create a New Process:** In interactive systems, users can start a program by typing a command or (double) clicking an icon.
  - (iv) **Initiation of a Batch Job:** Here, users can submit batch jobs to the system (possibly remotely). When the OS decides that it has the resources to run another job, it creates a new process and runs the next job from the input queue in it.
- System call CreateProcess() in Windows and fork() in Unix which tells the operating system to create a new process.
- When the operating system creates a process at the explicit request of another process, the action is referred to as process spawning. When one process spawns another, the former is referred to as the parent process, and the spawned process is referred to as the child process (or sub-process).
- The parent may have to partition its resources among its children, or it may be able to share some resources among several of its children. A sub-process may be able to obtain its resources directly from the operating system.
- When a process creates a new process, two possibilities exist in terms of execution:
  - (i) The parent continues to execute concurrently with its children.
  - (ii) The parent waits until some or all of its children have terminated.
- When a process creates a new process, there are two possibilities in terms of address space of the new process.
  - (i) The child process has the same program and data as parent.
  - (ii) The child process has new program loaded into it.

**2. Process Termination:** (W-17)

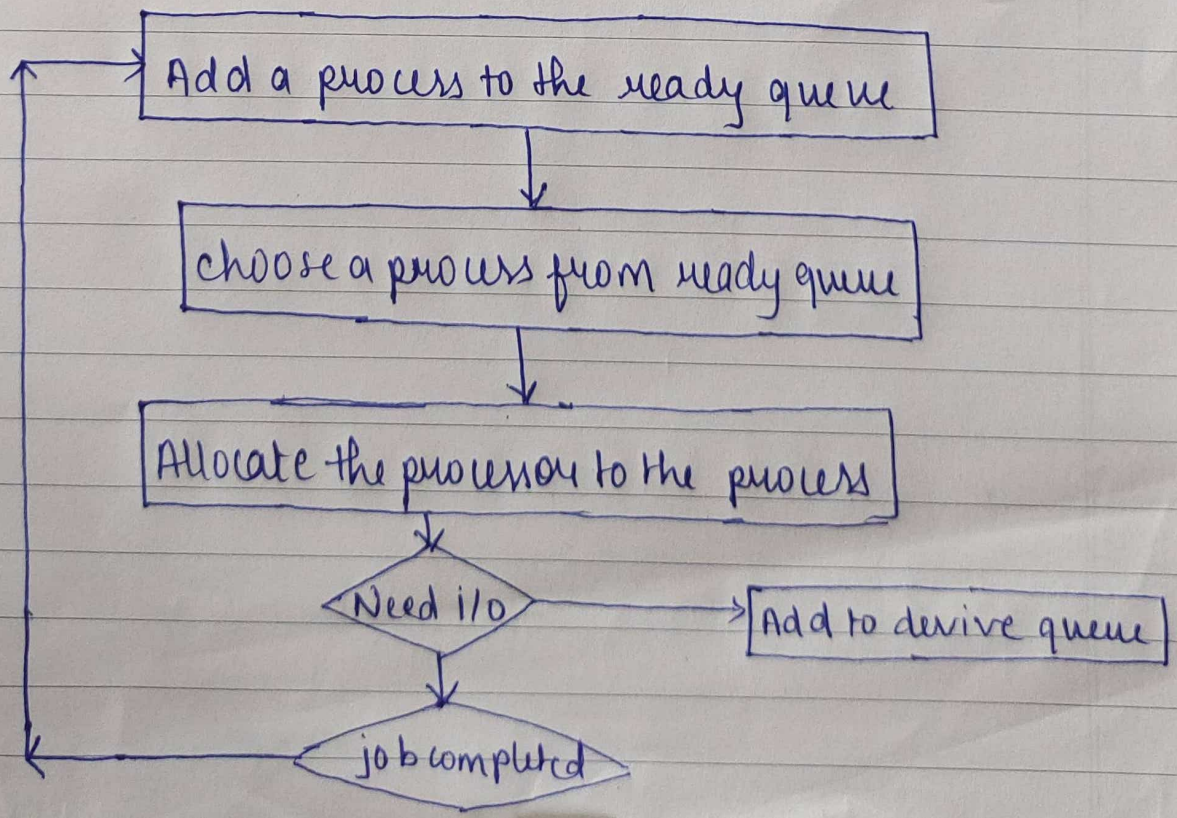
- Depending upon the condition, a process may be terminated either normally or forcibly by some another process.
- Normal termination occurs when the process completes its task (operation) and invokes an appropriate system call ExitProcess( ) in Windows and exit( ) in Unix to tell the operating system that it is finished.
- A process may cause abnormal termination of some another process. For this, the process invokes an appropriate system call TerminateProcess( ) in Windows and kill( ) in Unix that tells the operating system to kill some other process.
- Generally, the parent process can invoke such a system call to terminate its child process. This usually happen's because of the following three reasons:
  - (i) Cascading termination in which the termination (whether normal or forced) of a process causes the termination of all its children. On some operating systems, a child process is not allowed to execute when' its parent is being terminated. In such cases, the operating system initiates cascading termination.
  - (ii) The task that was being performed by the child process is not required.
  - (iii) The child process has used up the resources allocated to it more than that it was permitted.

ALDEL EDUCATION TRUST'S
ST. JOHN COLLEGE OF ENGINEERING AND MANAGEMENT

Date

## Process Scheduling :-

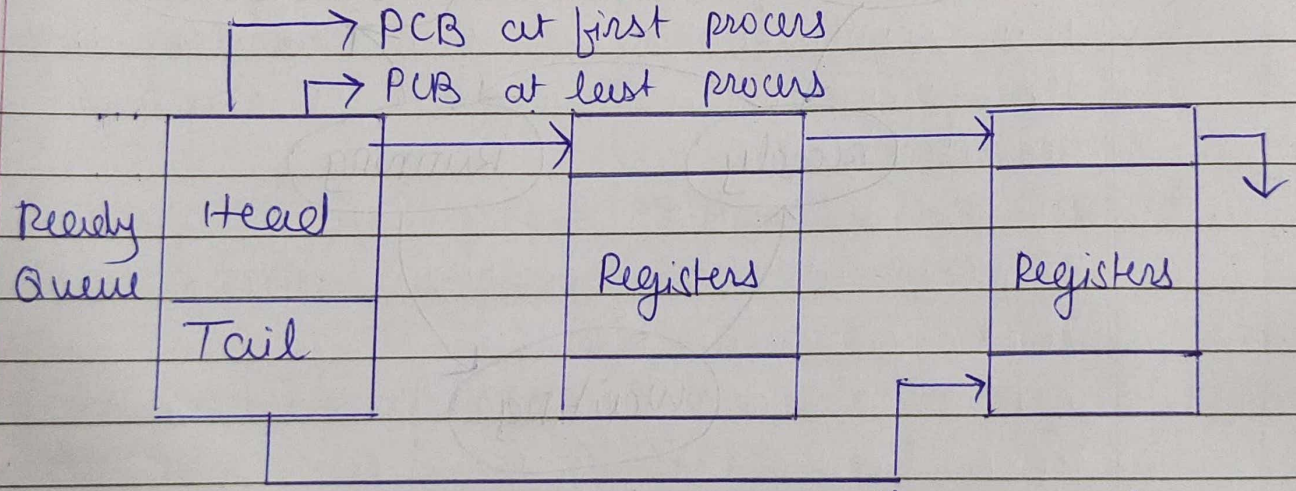
- We perform many programs at a time on the computer, but there is a single CPU. So for running all the ~~perf~~ program ~~those~~ concurrently or simultaneously, then we use the scheduling.
- Scheduling is that ~~what~~ which each process have some amount of time of CPU. Scheduling provides time of CPU to the each process.
- This procedure to determining the next process to be executed on the CPU is called process scheduling and the module of OS that process decision is called the schedular.



# * Process Scheduling

1) Scheduling is that in which each process have same amount of time of CPU. Scheduling provides time of CPU to the each process.
2) The procedure of determining the next process to be executed on the CPU is called process scheduling and the module of operating system that makes this decision is called the scheduler.

## 1] Scheduling Queues




1) For a uniprocessor system, there will be more than one running process. If there are more than one processes, the rest will have to wait until the CPU is free and can be rescheduled.
2) The processes, which are ready and waiting to execute, are kept on a list is called the ready queue.

PAGE No.
DATE / /

3) The list is generally a linked list. A ready queue header will contain pointers to the first and last PCB's in the list. Each PCB has a pointer ~~in the list — Each~~ field which points to the next process in the ready queue.

4) There are also other queues in the system. When a process is allocated to the CPU, it executes for a while and eventually quits, is interrupted or waits the occurence of a particular event such as the completion of an I/O request.

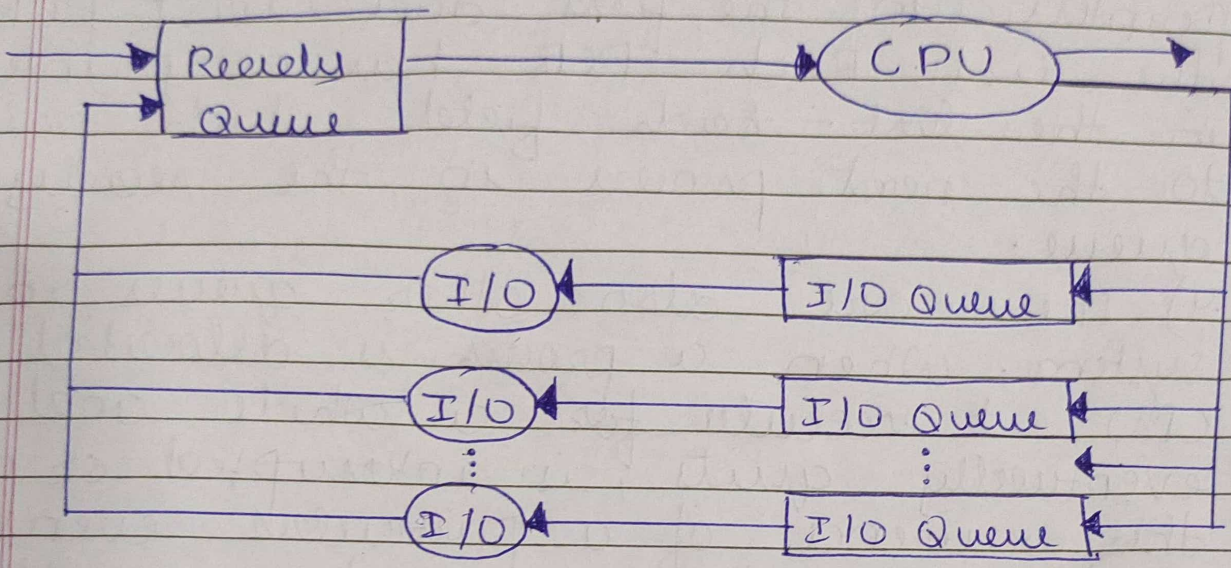
~~5) The process therefore may have to wait for the~~

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

6) A process enters the system from the outside world and is put in ready queue. It waits in the ready queue until it selected for the CPU. After running on the CPU, it waits for an I/O operations by moving to an I/O queue.

7) Eventually it is served by the I/O device and returns to the ready queue. A process continues this CPU, I/O cycle until it finishes and then

it exits from the system.




Queuing diagram representation of CPU scheduling.

8) The above figure shows the queuing diagram for representing the process scheduling. Each rectangle box represents a queue.

9) Two types of queues are present namely the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrow indicate the flow of processes in the system.

10) As new process is initially put in the ready queue. It waits there until it is selected for execution or is dispatched.

PAGE No.
DATE / /

Once the process is assigned the CPU and is execution, one of several events could occur.

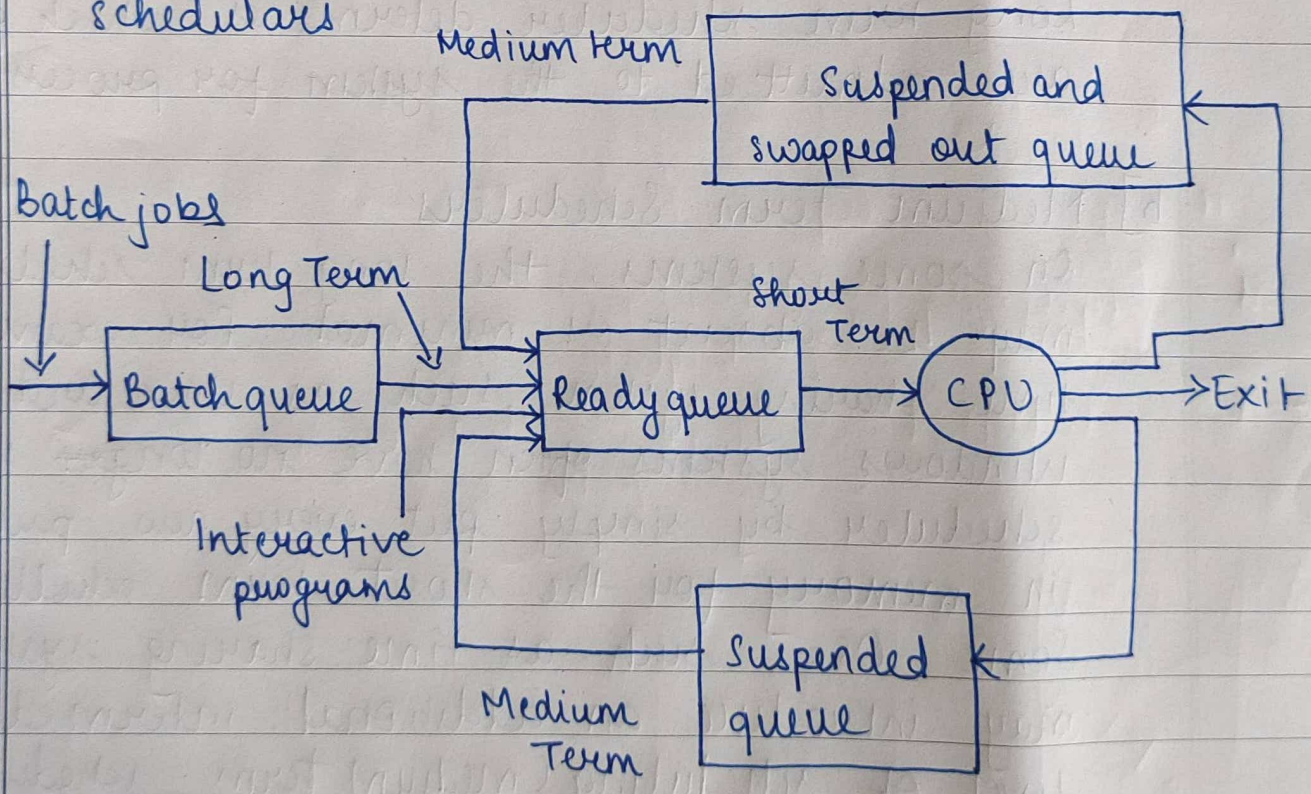
1) The process could create a new sub-process and wait for the termination of sub-process.
2) The process could issue an I/O request and then be placed in an I/O queue.
3) The process could be removed forcibly from the CPU, as an result of interrupt and again put in the ready queue.

ALDEL EDUCATION TRUST'S
ST. JOHN COLLEGE OF ENGINEERING AND MANAGEMENT

Date

# Schedulars

- Schedulars are special system softwares which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- In other words, the job of process scheduling is done by a software routine (module) called schedulars




## Working of Schedulars

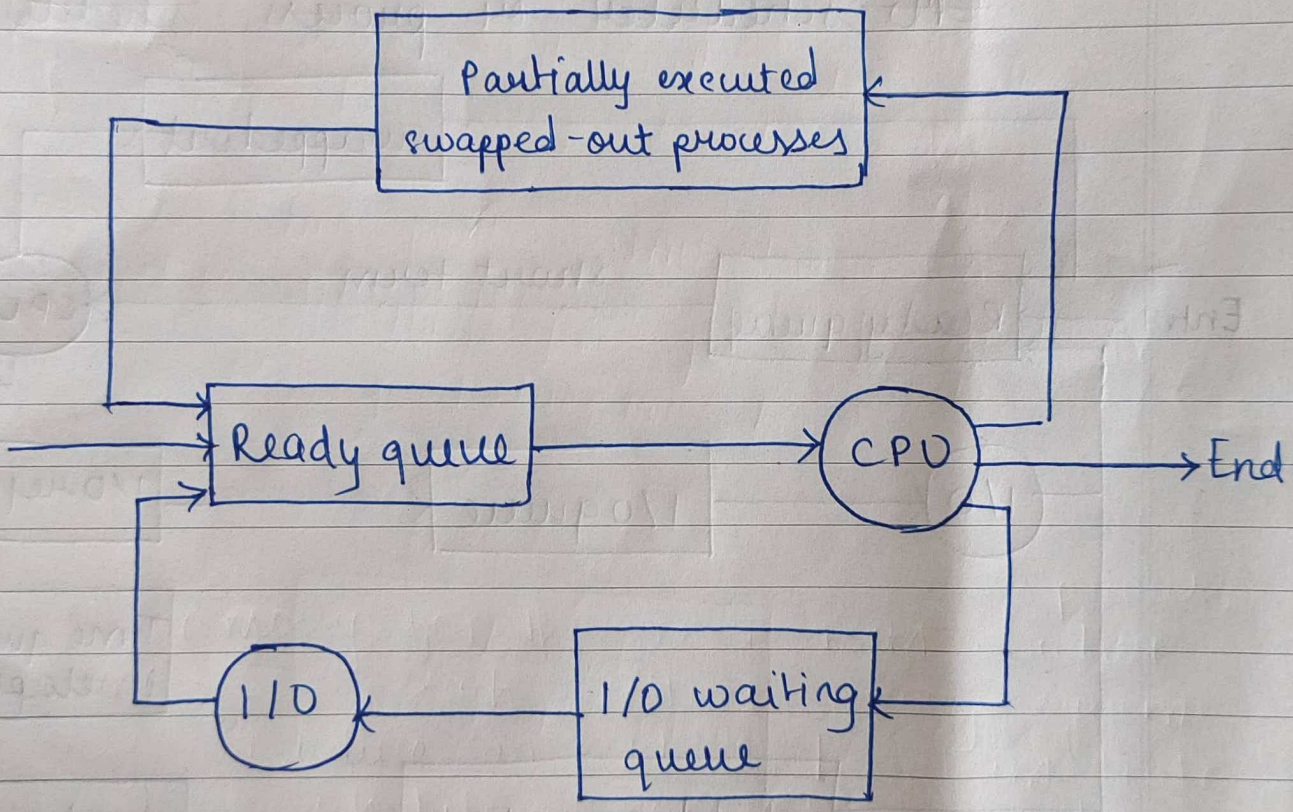
- Various types of schedulers are explained below:

a) Long Term Schedulers:
   - The long term schedulers determines which

jobs are admitted to the system for processing
- In batch system there are more jobs submitted that can be executed immediately. These jobs are spooled to mass storage device, where they are kept for later execution (ready queue).
- The long term scheduler selects jobs from this job pool and loads into memory for execution. It is also called job scheduler or admission scheduler.
- Long term scheduler determines which programs are admitted to the system for processing.

b] Medium Term Schedulers
- On some systems, the long term scheduler may be absent or minimal. For example, time-sharing system such as Unix and Microsoft Windows systems often have no long~~er~~ term scheduler by simply put every new process in memory for the short term scheduler
- Some OSs, such as time-sharing systems, may introduce an additional, intermediate level of scheduling (medium term scheduler). Medium term scheduling is a part of swapping so it is also known swapper.
- The medium term scheduler may decide to swap out a process which has not been active for some time or a process which has a low priority, or a process which

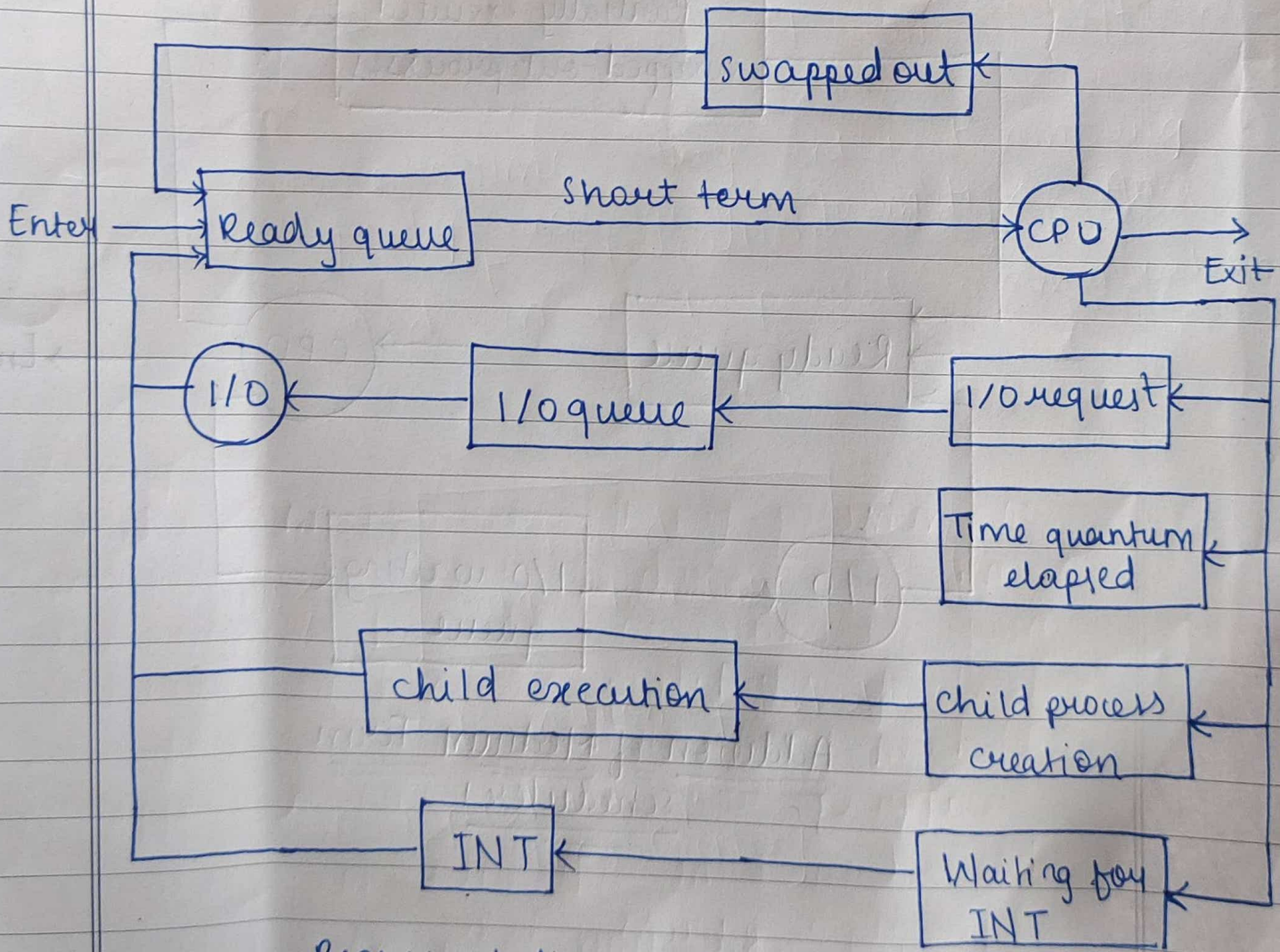
is page faulting frequently on a process which is taking up a larger amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available or when the process has been unblocked and is no longer waiting for a resource




Addition of Medium Term schedulers

## c] Short Term schedulers

- The short term schedulers selects from memory, which are ready to execute and allocate the CPU to one of them. It is also called as CPU scheduler or process scheduler.
- ~~The short term. It is also to called as CPU scheduler or process scheduler.~~




Representation of Process shedulong.

process spends more time doing computations; few very long CPU bursts.

**Difference between Short Term Scheduler, Long Term Scheduler and Medium Term Scheduler:** (S-18)

| Sr. No. | Short Term Scheduler | Long Term Scheduler | Medium Term Scheduler |
|---|---|---|---|
| 1. | Scheduler which selects the jobs or processes which are ready to execute from the ready queue and allocate the CPU to one of them is called short term scheduler. | The scheduler which picks up job from pool and loads into main memory for execution is called long term scheduler. | The medium term scheduler is that it removes the process from main memory and again reloads afterwards when required. |
| 2. | It is CPU scheduler. | It is job scheduler. | It is a process swapping scheduler. |
| 3. | Frequency of execution is high, (in milliseconds) | Frequency of execution is low, (in few minutes). | Execution frequency is medium. |
| 4. | Speed is very fast. | Speed is lesser than short term scheduler. | Medium term scheduler is called whenever required. |
| 5. | It deals with CPU. | It deals with main memory for loading process. | It deals with main memory for removing processes and reloading whenever required. |
| 6. | It provides lesser control over degree of multiprogramming. | It controls the degree of multiprogramming. | It reduces the degree of multiprogramming. |
| 7. | Minimal in time sharing system. | Absent or minimal in time sharing system. | Time sharing system use medium term scheduler. |

### 3.2.3 Context Switch

Page No.
Date

## * 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 a Context Switch.

- CPU switching from one process to another process is called a Context switch.

- A Context Switch is the computing process of storing and restoring the state (context) of a CPU so that execution can be resumed from the same point at a later time.

- This enables multiple processes to share a single CPU. The Context Switch is an essential feature of multitasking OS.

- The Context of a process is represented in the PCB of the process, it includes the value of the CPU registers the process state and memory management information.

- When a Context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. The Context switching time is an overhead time. During switching the time System does not do any useful work.

- Context Switch times are highly dependent on hardware support. It speed varies from machine to machine, depending on the memory speed, the no. of registers that must be copied and the existence of special instructions.

- Typically, the speed range from 1 to 1000 microseconds. Context Switch times are highly dependent on hardware support.

- Some hardware systems employ two or more sets of processor registers to reduce the amount of Context Switching time. When the process is switched, the following info is stored.

1] Program Counter.
2] Scheduling Information.
3] Base and limit value.
4] Currently used register
5] Changed State.
6] I/o State
7] Accounting

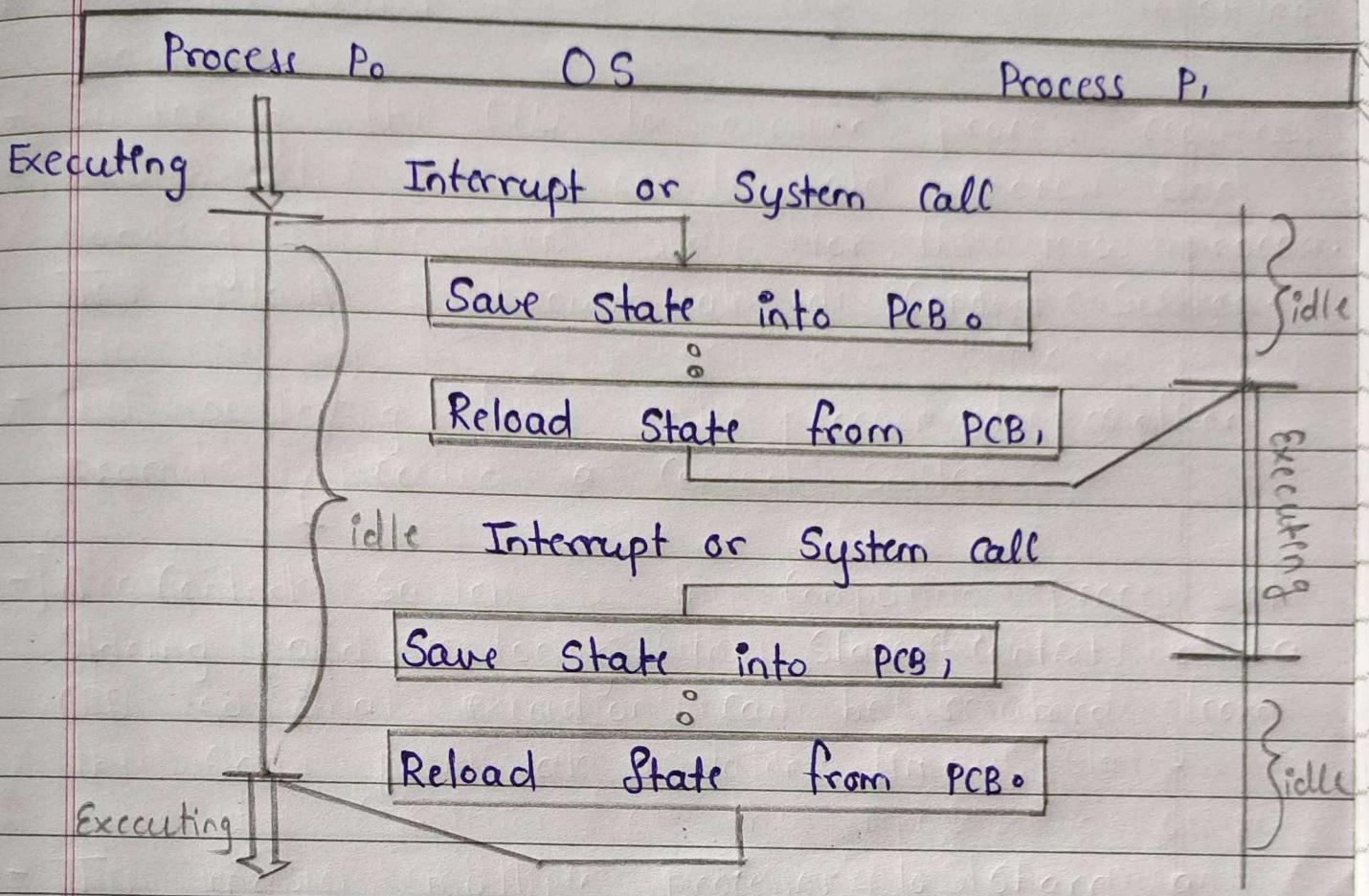



fig. CPU Switch from Process to Process

## 3.3 INTERPROCESS COMMUNICATION (IPC) (S-16, 19, W-16, 18)

- Operating system provides co-operating processes to communicate with each other via an InterProcess Communication (IPC) facility. (See Fig. 3.13).
- There are applications that involve each executing multiple processes concurrently. Processes work together to perform application-specific tasks. They are referred to as co-operating (interacting) processes.
- Co-operating processes are "loosely" connected in the sense that they have independent private address spaces and run at different speeds. The relative speeds of the processes are not normally known.
- From time to time, they interact among themselves by exchanging information. An exchange of information among processes is called an inter-process communication.
- When multiple processes run on a system concurrently and more than one process requires CPU at the same time, then it becomes essential to select any one process to which the CPU can be allocated. To serve this purpose, scheduling is required.
- Moreover, the multiple processes running on a system also need to intercommunicate in order to reciprocate some data or information. This kind of intercommunication between several processes is referred to as Inter Process Communication (IPC).

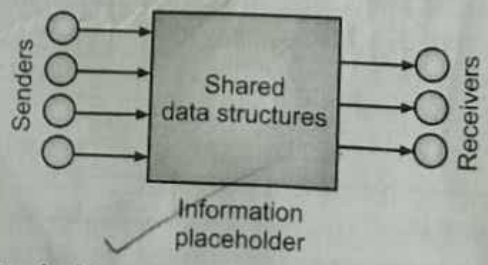
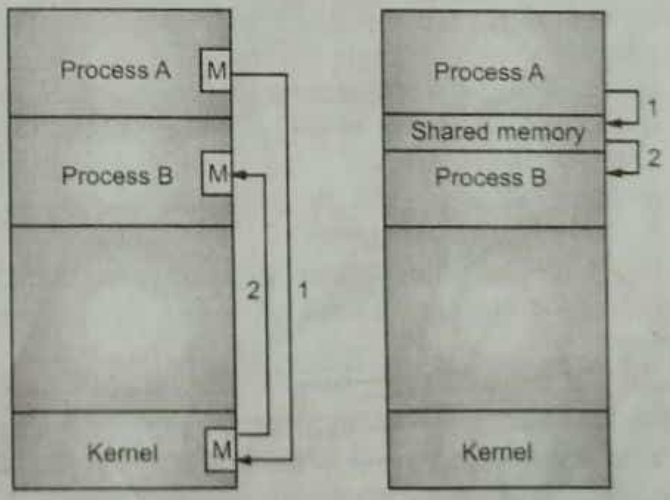



**Fig. 3.13: A Typical InterProcess Communication (IPC) Facility**

## 3.3.1 Introduction to IPC

(W-16, 18)

- Processes executing concurrently in the operating system might be either independent processes or co-operating processes. InterProcess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.
- This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The IPC interfaces make this possible.
- The Interprocess Communication (IPC) is a set of techniques for the exchange of data among multiple processes. IPC techniques divided into methods for synchronization, message passing, stored memory etc.
- A capability supported by some operating systems that allows one process to communicate with another process. The processes can be running on the same computer or on different computers connected through a network.
- IPC enables one application to control another application, and for several applications to share the same data without interfering with one another.
- IPC is required in all multiprocessing systems, but it is not generally supported by single-process operating systems such as DOS, OS/2 and MS-Windows etc.
- IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network. For example, chat program used on the World Wide Web. IPC is best provided by a message passing system.
- **Purposes of IPC:** Data transfer, sharing data, event notification, Resources sharing and synchronization, and process control.
- InterProcess Communication (IPC) is one of the most common concepts used in operating systems (OS) and distributed computing systems. It deals with how multiple process can communicate among each other.
- Two fundamental models allows inter-process communication as explained below:
  1. **Shared Memory Model:** Two processes exchange data or information through sharing region. They can read and write data from and to this region.
  2. **Message Passing Model:** In message passing model the data or information is exchanged in the form of messages.




**(a) Message Passing** **(b) Shared memory**

**Fig. 3.14: Communication Models**

## 3.3.2 Shared Memory System

(S-18)

- IPC using shared memory requires a region of shared memory among the communicating processes. Processes can then exchange information by reading and writing data to the shared region.
- Typically, 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 it to their address space.
- Normally, the operating system does not allow one process to access the memory region of another process.
- Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.

## ★ Shared Memory System :

- IPC using shared memory requires a region of shared memory among the communicating processes. Processes can then exchange information by reading and writing data to be shared region.
- Normally, the operating system does not allow one process to access the memory region of another process.
- Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.
- The form of the data and the location are determined by these processes & are not controlled by the operating system.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

- Consider the producer-consumer problem; a producer process produces information that is consumed by a consumer process.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.
- Two types of buffers can be used:
  1) Unbounded Buffer - The unbounded buffer places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

  2) Bounded Buffer - The bounded buffer has a fixed size buffer. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

- Shared memory is faster than message passing systems are typically implemented using system calls & thus require the more time-consuming task of kernel intervention.

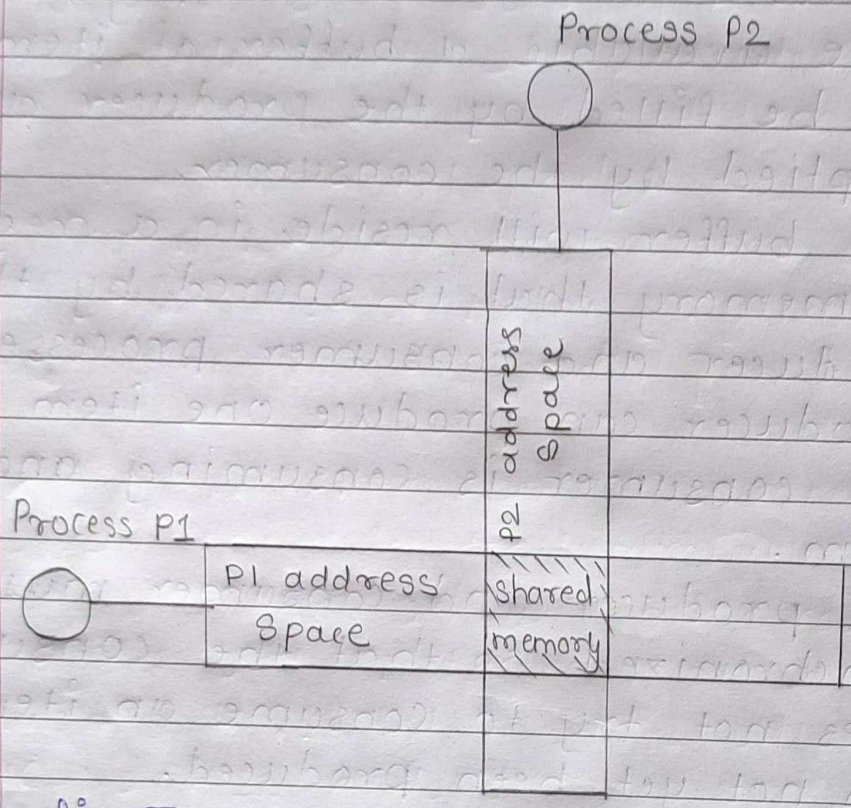
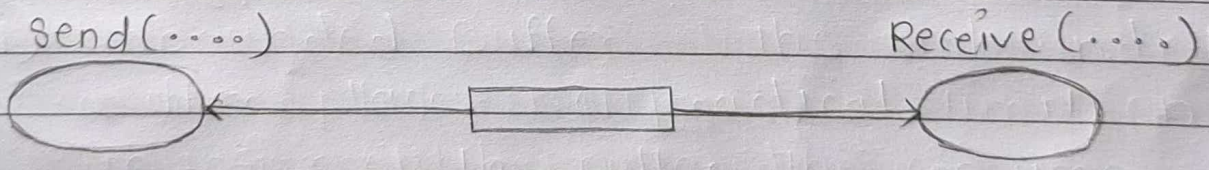



Fig. Inter-process Communication through Address Space Sharing

- In contrast, in shared memory systems, system calls are required only to establish shared-memory regions.

## ★ Message Passing Systems :

- Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.
- For example, a chat program used on the world wide web could be designed so that chat participants communicate with one another by exchanging messages.
- The function of a message system is to allow processes to communicate with one another without the need to resort to shared data.




- Fig. Process of Message Passing

- An IPC facility provides the two operations:
  1) Send (message)
  2) Receive (message)
- If process A & B want to communicate, they must send messages to and recieve messages from each other; a communication link must exist between them.

PAGE No.
DATE / /

# * Threads

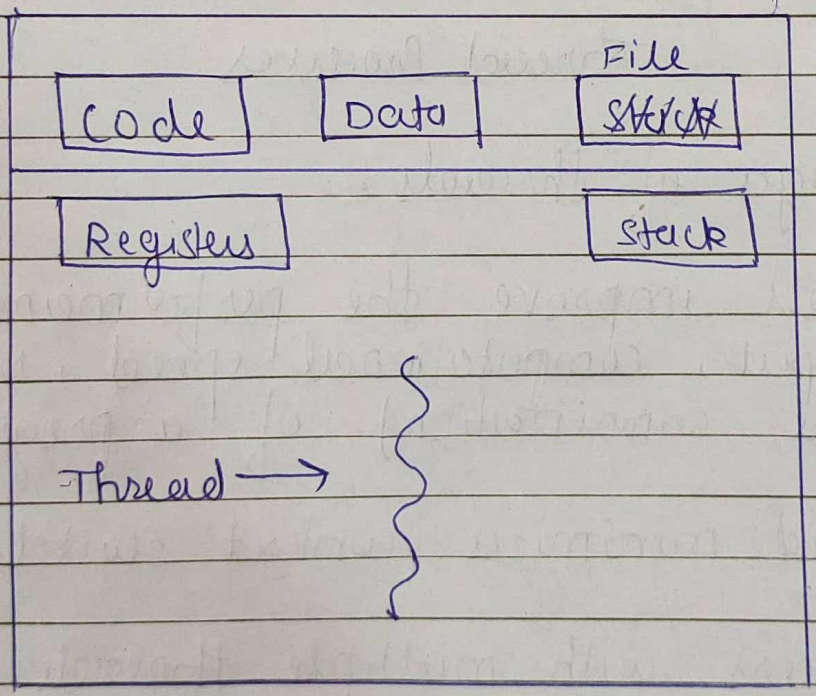
1) A thread, sometimes called a Light Weight process (LWP) is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set and a stack.
2) A thread is defined, "as a unit of concurrency within a process and had access to the entire code and data parts of the process". Thus, thread of the same process can share their code and data with one another.
3) It shares with other threads belonging to the same process its code ~~stack~~ section, data section, and other operating-system resources, such as open files and signals.
4) Traditional (or heavy weight) process has a single thread of control. If a process has multiple threads i.e it can do more than one task at a time. Many software packages that run on desktop PCs are multithreaded.

PAGE No.
DATE / /

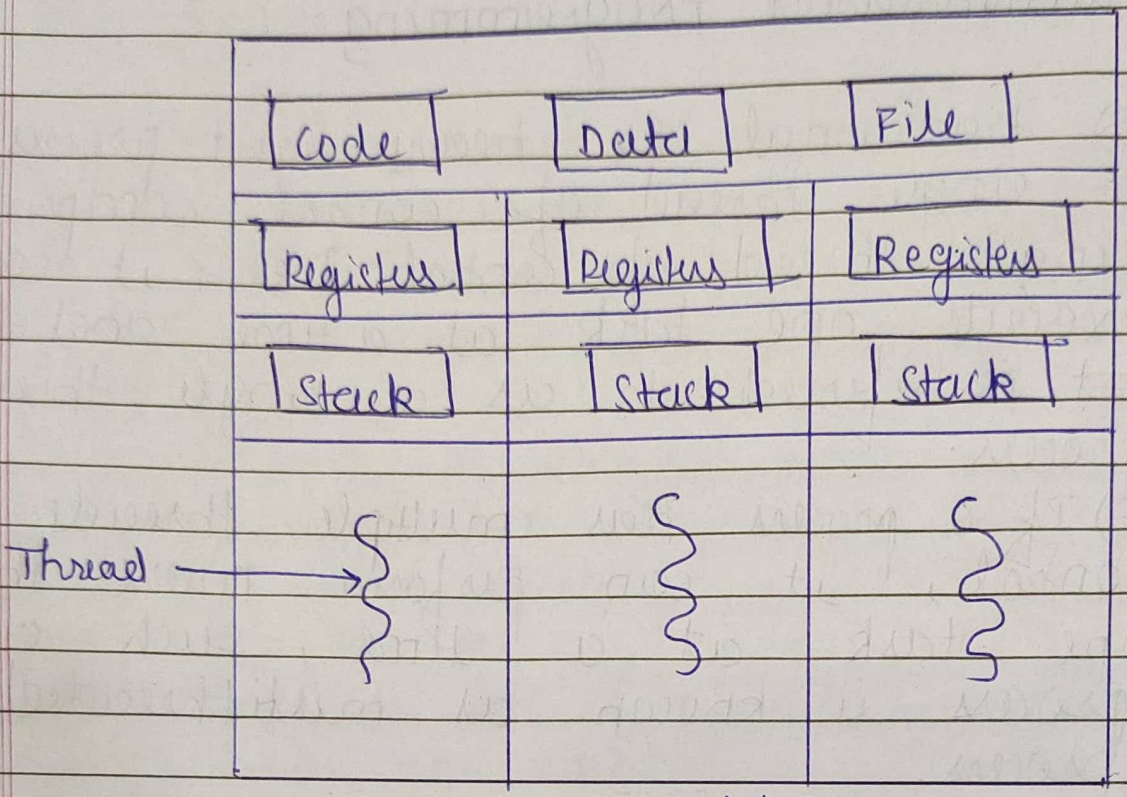
# * Multithreaded Programming

1) A traditional or heavyweight process has a single thread of control comprises a single thread of control i.e, it can execute one task at a time and thus, it is referred to as a single-threaded process.

2) If a process has multiple threads of control, it can perform more than one task at a time, such a process is known as multithreaded process.




(a) Single Threaded process




(b) Multithreaded Process

Threaded Processes

Advantages of threads:

1) Threads improve the performance (throughput, computational speed, responsiveness, or some combination) of a program.

2) Thread minimizes context switching time.

3) A process with multiple threads makes a great server for example printer server.

4) Because threads can share common data, they do not need to use their inter-process communication.

PAGE No.
DATE / /

Benefits :

1) Responsiveness : Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing lengthy operations, thereby increasing responsiveness to the user. A multithreaded web browser could still allow user interaction in one thread while an image being loaded in another thread.

2) Resource sharing : By default, threads share the memory and the resources of the process to which they belong. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.

3) Utilization of Multiprocessor Architecture : The benefits of multithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor. A single threaded process can only run on one CPU, no matter how many are available.

* User and Kernel Threads

- Practically, threads can be implemented at two different levels namely, user level (user threads) and kernel level.

- A library which provides the programmers with an Application Programming Interface (API) for thread creation and management is refered to as a thread library.

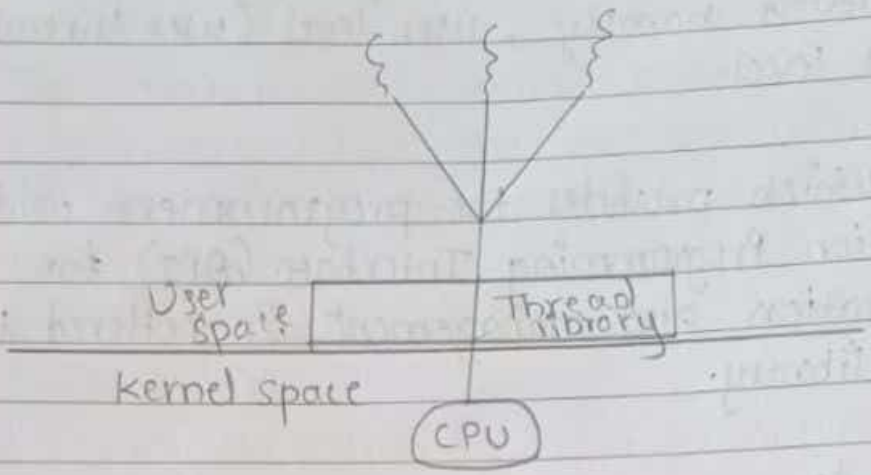
* User threads.

- The threads implemented at the user level are known as user threads. In user level thread, thread management is done by the application; the kernel is not aware of existence of threads.

- User threads are supported above the kernel and are implemented by a thread library at the user level. The library provides support for thread creation, scheduling and management with no support from the kernel.

- Because the kernel is unaware of the user level threads, all thread creation and scheduling are done in user space without the need for kernel intervention. Therefore, user level threads are generally fast to create and manage.

- User thread libraries include POSIX PThreads, Mach C-Threads and Solaris 2 UI-threads.




= Advantages of User level threads:

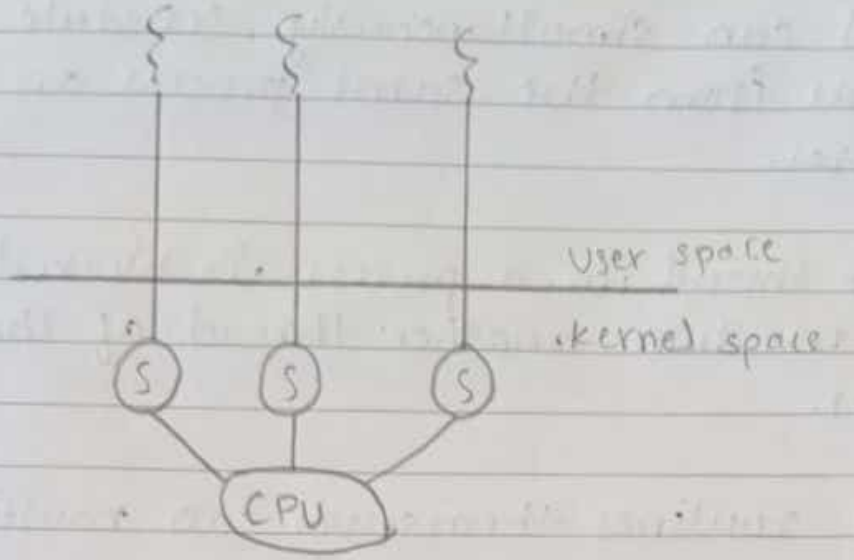
1) User level thread can run on any operating system
2) A user thread does not require modifications to operating systems.
3) User level threads are fast to create and manage.
4) User thread library is easy to ~~pa~~ portable
5) Low cost thread operations is possible.

= Disadvantages of User level thread:

1) Multithreaded applications cannot take advantage of multiprocessing.
2) At most, one user level thread can be in operation at one time, which limits the degree of parallelism
3) If a user thread is blocked in the kernel, the entire process (all threads of that process) are blocked.

PAGE NO.
DATE / /

* Kernel Threads.




- The threads implemented at kernel level are known as kernel threads. Kernel threads are supported directly by operating system.

- The kernel performs thread creation, scheduling and management in kernel space. Because thread management is done by the operating system, kernel threads are generally slower to create and manage than are user threads.

- However since the kernel is managing the threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. Also in a multiprocessor environment, the kernel can schedul threads on different processors.

- Most ~~com~~ contemporary operating system- including Windows NT, windows 2000, Solaris 2, BeOS and Tru64 UNIX - support kernel thread.

= Advantages of kernel level thread.

1) Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

2) If one thread in a process is blocked, the kernel can schedule another thread of the same process.

3) Kernel routines themselves can multithreaded.

= Disadvantages of kernel level threads:

1) The kernel level threads are slow and inefficient.

2) For instance, threads operation are hundreds of times slower than that of user level threads.

3) Transfer of control from one thread to another within same process requires a mode switch to the kernel.

PAGE NO.
DATE / /

## * Multithreading Models.

Many systems provide support for both user and kernel threads, resulting in different multithreading models. Three types of multithreading models implemented.

### 1. One-to-One Model.

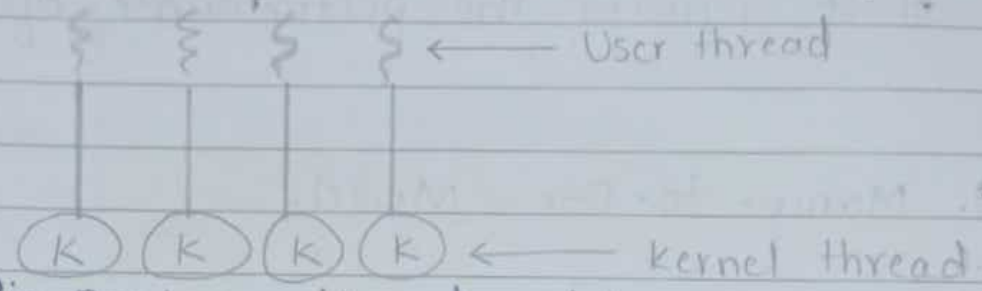



fig- One to one thread model.

- The one-to-one model maps each user thread to a kernel thread.
- It provides more concurrency than the many-to one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors.
- Windows NT, Windows 2000 and OS/2 implement the one-to-one model.

### = Advantages of One-to-One Model.

1) More concurreny because of multiple threads can run in parallel on multiple CPUs.
2) Multiple threads can run parallel.
3) Less complications in the processing.

= Disadvantages of One-to-One Model.

1) Thread creation involves Light Weight Process creation.
2) Every time with users thread, kernel thread is created.
3) Limiting the number of total threads.
4) Kernel thread is an overhead.
5) It reduces the performance of system.

2. Many-to-One Model.

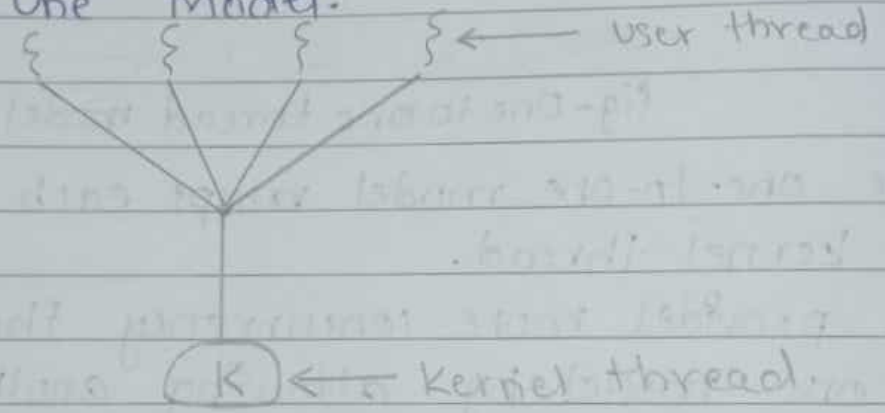



Fig- Many to One model

- The many to one model maps many user level threads to one kernel thread. Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call.
- Only one thread can access the kernel at a time.
- Multiple threads are unable to run in parallel on multiprocessors.
- Green threads a thread library available for Solaris 2-uses this model.

= Advantages of Many to One Model:

1) Totally protable.
2) Easy to do with few systems dependencies.
3) Mainly used in language systems, portable libraries
4) Efficient system in terms of performance.
5) One kernel thread controls multiple user threads.

= Disadvantages of Many to One Model:

1) Cannot take advantage of parallelism.
2) One block call blocks all user threads.

3. Many to Many Model.

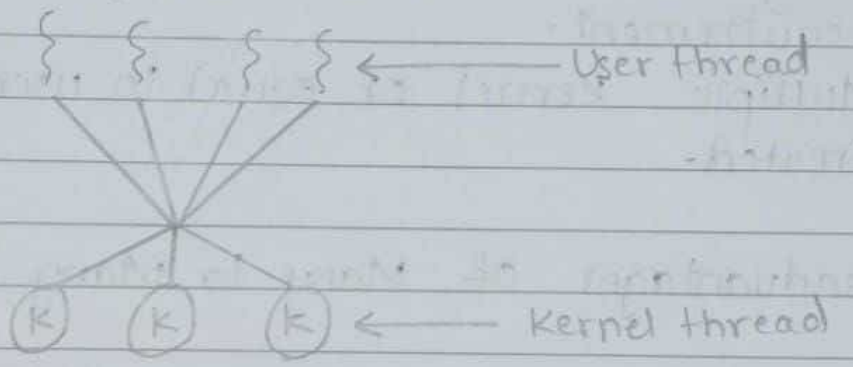



fig. Many to Many Model.

- The many to many model multiplexes many user level threads to a smaller or equal number of kernel threads.
- The number of ~~kern~~ kernel threads may be specific to either a particular application or a praticular machine

- This model allows developer to create as many threads. Concurrency is not gained because the kernel can schedule only one thread at a time.
- The one to one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application.
- Solaris 2, IRIX, HP-UX and Tru64 UNIX support this model.

= Advantages of Many to Many Model.

1) Many threads can be created as per user's requirement.
2) Multiple kernel or equal to user threads can be created.

= Disadvantages of Many to Many Model:

1) True concurrency cannot be achieved.
2) Multiple threads of kernel is an overhead for operating system.
3) Performance is less.