# Unit - ② Process Management :-

* **Process** :- Execution of Programs.

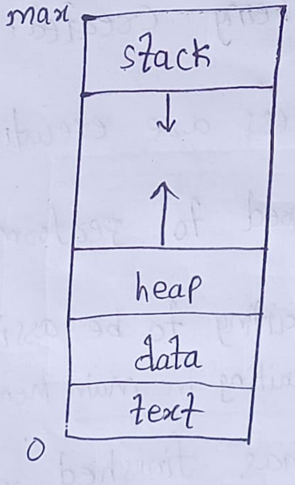
→ Process refers to the Progoam in execution. A Process ~~may~~ can be active (or) Passive i.e.

⇒ The Process in execution refers to active state that is in main memory. [This is called "active state"]

⇒ The Process are resided in secondary storage devices. This is called "Passive state".

* **Process ~~in~~ Memory** :-

→ Every Process has a structure to store in main memory.



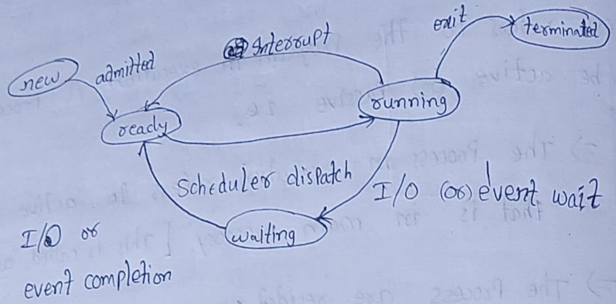
* **stack** :- It is a data structure which stores Local variables.

* **Heap** :- It is a data structure which allocates dynamically created memory

* **Data** :- It is used to store the values & Process Information

* **Text** :- It is used to store Program code.

## Diagram of Process state:-



→ During the execution of Process it undergoes different Process states which is represented as

→ As the Process executes, it changes state

(i.) new : The Process is being Created.

(ii.) running : ~~Instructions~~ Process are executing in CPU.

(iii.) waiting : Process are required to perform I O request.

(iv.) ready : The Process is waiting to be assigned to a Processor. (The Process are waiting in Main Memory for allocating CPU)

(v.) terminated : The Process has finished execution.

* Process Control Block (PCB).- The os maintains the Process related information for empty Process using a structure called "PCB".

→ Information associated with each Process (also called task control block).

=> Process state - Running, waiting e.t.c. It Indicates the state of a Process in which existed during it's execution.

→ It is the unique identity of a Proces which is created using fork system call.

=> Program Counter:- It specifies the Process of next instruction to be executed

=> CPU registers:- these are used inorder to use intermediatery result during Arithmetic operation.

=> Memory Management limits:- ~~Information~~ It is the range of memory address of Process.

=> Accounting Information:- The OS records CPU utilization and no. of clock cycles wasted.

=> I/O status information:- It is used to store what type of I/O device is utilized by Proces, list of open files.

| |
|---|
| Process state |
| Process number |
| Process counter |
| registers |
| Memory ~~Management~~ ~~Information~~ limits |
| List of open files |
| ... |

* Threads:-

→ Thread is an smallest unit of task of Process.

Ex:- Inorder to open word document spell checking software, editing software, help software are automatically loaded during Opening the word document.

→ These are Individual threads.

* Process scheduling:- scheduling is required inorder to Increase CPU utilization & increase waiting time & maximizing.

→ Process scheduler which is used to select one process for execution,

→ Different scheduling queues are

(i.) Job queue
(ii.) Ready queue
(iii.) Device queue

(i.) Job queue:- No. of Processors enter into the system are maintained in this queue.

(ii.) Ready queue:- The Process are waiting in this Queue for allocating CPU.

(iii.) Device queue:- The no. of Processes is waiting in this queue for aquiring I/o Devices.

* Schedulers:- ↗ It is the time taken by the schedulers inorder to select one Process at a time from the ready queue to allocate CPU

* Short-term scheduler (or CPU scheduler):- It selects which Process should be executed next & allocates CPU.

→ sometimes the only scheduler in a system.

→ It is invoked frequently (milli seconds) ⇒ (must be fast).

* Long-term scheduler (or job scheduler):- It selects which process should be brought into the ready queue.

→ Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow).

→ It controls the degree of multi Programming.

→ Processes can be described as either:

(i.) I/o bound Process → spends more time during I/o than computations, many short CPU bursts.

(ii.) CPU-bound Process → It spends more time during computations; few very long CPU bursts.

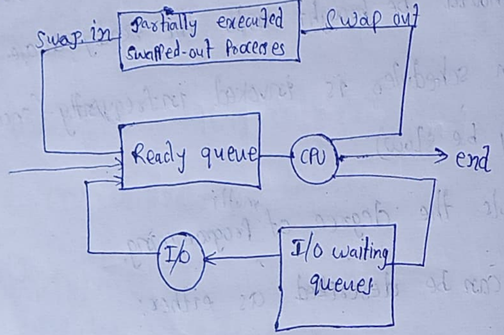
→ Long term scheduler strives for good Process mix.

→ Long term scheduler refers to the longest time taken by scheduler to select the Process from job queue into the ready queue. It decreases Multiprogramming Process.

Next Page →

*Medium - Term Scheduler:- It is the medium amount of time taken by the scheduler inorder to process the request of IO device. It requires to swap out the process inorder to serve the IO device and then swapping the process into the ready queue. It is called swapping.

~~*Conte~~



*context - switch:- To support multiprogramming the OS share the CPU to multiprocesses and switching the CPU in b/w them.

→ During this Process the OS save the address Location of Process in the PCB & again retained the same address to continue the process execution.

Next Page

*Operations on Processes:-

① Process creation

② Process termination

① Process Creation:- A Process is created using a fork ~~statement~~ System call then Process Identification is generated for every proces.

→ If PID < 0 → It is an error

→ If PID == 0 → It is an child Process

→ If PID > 0 → It is an Parent Process.

→ Both Parent & child process share same resources but having seperate memory location.

→ The Parent Process is terminated when the child process is terminated by Passing a wait of a system call to various Processes.

② Process Termination:- Parent Process terminate child Process using about System called in different situations.

(i) The child Process exceeded the usage of allocated resources.

(ii.) The child Process is no longer required.

(iii.) Parent Process is terminating such that the OS does not allow the child Process to continue.

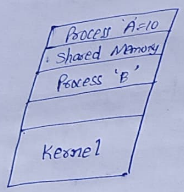
* cascading termination:- All the child processes are terminated at a time.

* Zombie Process:- The Parent Process is terminating without the child Process termination.

*Orphan Process:- Parent Process does not have child Process such that it won't wait for wait of the System call.

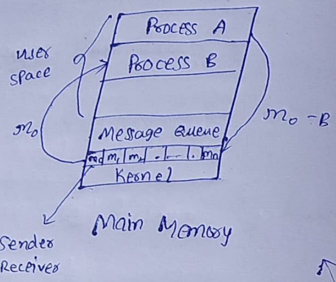
→ ~~[illegible]~~ In this single block the Processors can share the data by performing different operations.

Shared Memory



Main Memory

Message Passing



Main Memory

*Message Passing:- When more than one Proces required to send messages, it uses the ~~[illegible]~~ indirect way i.e; the messages are send to the Kernel Memory which are Placed in a Queue. Then, Kernel interpret the messages inorder to understand what is the receiver Proces in which the message is sent to the Process i.e., through the Kernel the Processors can exchange messages.

After this => Diagram

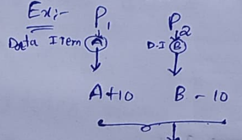
→ There are 2 types of Processors.

(i.) Independent Process

(ii.) Cooperating Process

(i.) Independent Process:- The Processors performing operations on different type of data (or) files such that the results of one process will not affect on other Processes;

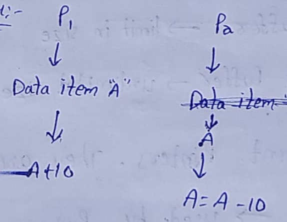
Ex:-



Both are different & one will not affect other. i.e., Result will not effect.

(ii.) Cooperating Process:- More than one Process can Perform ~~same~~ operations on same data item "A" then, results are affected on another process. For this type of Processors requiree an communication in order to share the results after applying operations after it.

Ex:-



Next Page →

## *Producer - Consumer Problem:-

The Producer Process is responsible for uploading (or) Inserting the data & the Consumer Process is responsible for downloading (or) retrieving the data.

~~Solution:- Which ty~~

→ which type of IP is used

Solution:- The shared memory technique is ~~Performed~~ applicable for Performing the ~~data~~ communication b/w the Process.

## * Implementation of this Problem using shared Memory:-

Consider buffer is a memory block which is shared b/w the Processors.

→ The Buffer can be of 2 types.

(i.) Bounded Buffer → limit in size

(ii.) Unbounded Buffer → unlimited size

→ It uses two different Pointers. They are:-

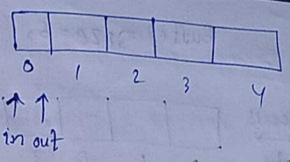
(i.) in pointer → used by Producer Process

(ii.) out pointer → used by Consumer Process.

→ Initially in=0; out=0;

```
if(in == out){
    "Buffer is Empty"
}
```

→ Buffer size=5



→ in pointer always points to the next empty (or) free location.

→ out pointer always points to the first full Position in the buffer.

* **Bounded Buffer:-** The Producer Process has to wait if the Buffer is completely filled up & the consumer process will not consume the items [Producer process has to wait for empty location]

→ Consumer Process has to wait until the Producer Process Produces the item.

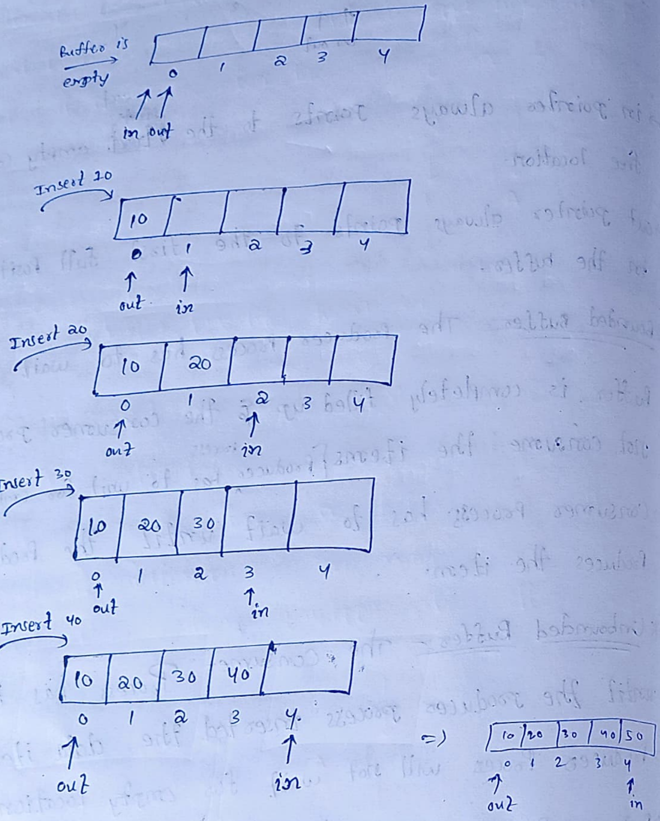
* **Unbounded Buffer:-** The Consumer Process has to wait until the producer process inserted the data items.

→ Producer Process will not wait for empty location.

## * Producer Process Implementation:-

```
struct                      item nextProduced;
{                           while(true);
    int in, out;            while(((in+1) % Buffersize) == out);
} item;                     Buffer[in] = next Produced;
item buffer[size];          in = (in+1) % Buffersize;
int in=0, out=0;
```

Buffer size = 5, in = 0, out = 0.



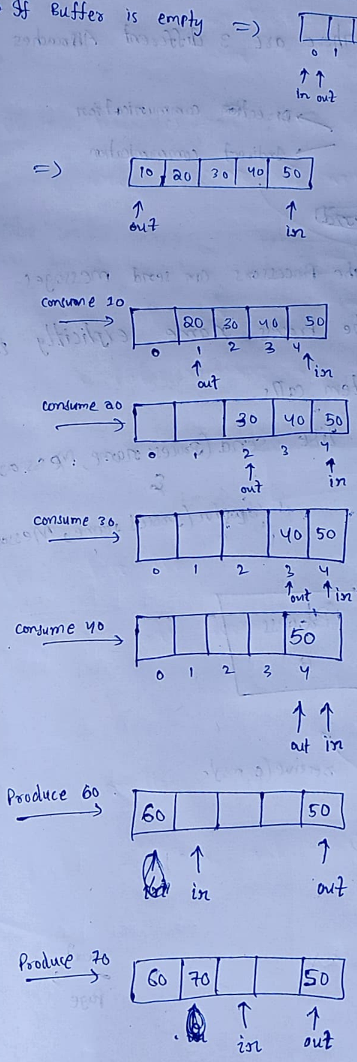
* Consumer Process implementation:-

Backside code +

```
item next consumed;
while (true) {
    while(in == out);
    next consumed = Buffer[out];
    out = (out + 1) % Buffersize;
}
```

Buffer size = 5, in = 0, out = 0

→ If Buffer is empty =>



*Message Passing:- More than one Processors Perform Communication by exchanging messages. There are 3 different Approaches for Message Passing.

(1.) Naming convention → Direction communication
→ Indirect communication

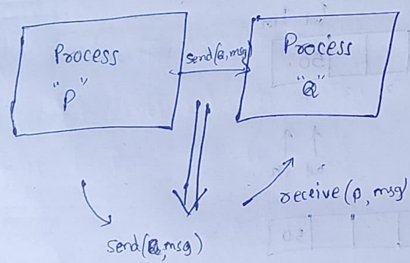
~~→ The Processors can send~~

(1.) Naming Convention:-

*Direction Communication:- The Processors can send messages directly by specifying the Process name explicitly in the send & receive system calls.

→ Two different operations like send (Process name, Message)
&
receive (Process name, Message)

Ex:-



Next Page

⇒ Here, Process "P" sends the msg to Process "Q" by executing send(Q, msg).

→ Process Q receives msg by executing receive(P, msg).

*Indirect communication:- The Processors can communicate each other by using a mediator i.e., mail box.

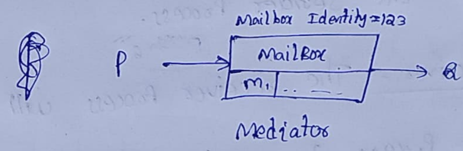
→ The sender Process "P" sends the mail m, to the receiver Process "Q", it requires to executes send () System call.

i.e., send (mailbox Identity, message)

→ The receiver Process receives the message by mailbox by executing receive () System call

i.e., receive (mailbox identity, message)

Ex:-



⇒ send(123, $m_1$)

⇒ receive(123, $m_1$)

→ Every mailbox has unique identity

② Synchronization:-
- Blocked sender
- Blocked Receiver
- UnBlocked Receiver
- UnBlocked sender

*Blocked sender:- The sender Process has to wait for the receiver Process to receive the message which is sent by the sender Process.

$\boxed{P \xrightarrow{msg} Q}$

*Blocked receiver:- The receiver Process has to wait for the sender Process to receive the message which is sent by the receiver Process.

$\boxed{P \xleftarrow{msg} Q}$

*Unblocked sender:- The sender Process will not wait for the ~~receiver~~ receiver Process to receive the message which is sent by the sender Process.

$P \xrightarrow{msg} Q$

*Unblocked receiver:- The receiver Process will not wait for the sender Process to receive the message which is sent by the receiver Process.

$P \xleftarrow{msg} Q$

Next → Page

③ Buffering:-
- Zero capacity (No Buffer)
- finite
- Infinite

* Zero capacity:- There is no buffer used b/w the Processors such that the sender process has to wait for the receiver Process to receive the message because there is no storage of messages.

* Finite capacity:- Here the buffer size is limited. The Producer Process (or) sender Process has to wait until the data item is consumed by receiver Process, In case of buffer is ~~filled~~ full.

* Infinite capacity:- Here the Buffer size is unlimited. The Producer Process (or) Sender Process will not wait for empty location because the buffer size is unlimited.

→ Consumer Process has to wait until there is a data item is produced by Producer Process (or) Sender Process