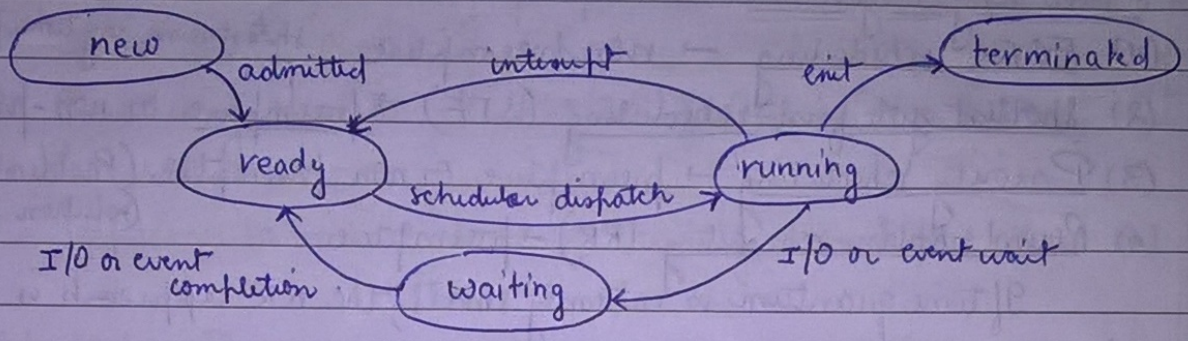
UNIT-3

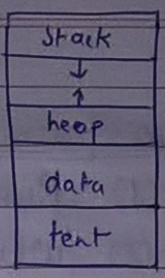
→ **Process -**

① **Process Concept -**

A process is a program in execution.

Process state :- New, Running, Waiting, Ready, Terminated





② **Process Control Block (PCB) -** Task control block, contains piece of information.

Process state	→ new, ready, waiting, running, terminated
Process number	
Program counter	→ address of the next instruction
registers	
scheduling info	→ which scheduling is using.
memory info	→ page tables or segment tables
accounting info	→ amount of CPU and real time used.,
I/O status	→ Allocated I/O devices, list of open files

PCB

③ **Scheduling Criteria -**

→ CPU utilization (range → 0 to 100%) (Should be) (MAXIMUM)

→ Throughput → No. of process completed per unit time. (MAXIMUM)

→ Turnaround Time = finish time - arrival time (MINIMUM)

→ Waiting time = Turnaround Time - Burst time (MINIMUM)

→ Response time = First response time - arrival time (MINIMUM)

④ **Preemptive Scheduling -** (1) Running state to waiting state

(2) Running state to ready state

(3) Waiting state to ready state

(4) When a process terminates.

Non-preemptive scheduling –

(1) Running state to waiting state
(2) When a process terminates.

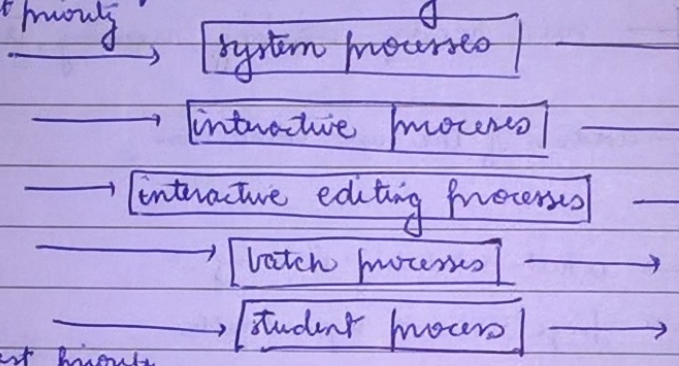
⑤ Scheduling algorithms –

(1) FCFS scheduling → non preemptive
(2) Shortest job first scheduling (SJF) → preemptive or non-preemptive (shortest remaining time first sch
(3) Priority scheduling → preemptive or non-preemptive (Problem → starva
(4) Round-Robin scheduling (RR) → preemptive (Solution → aging

If time quantum is extremely small, the RR approach is called process sharing.

(5) Multilevel queue scheduling – don't allow process to move between queues.

highest priority



lowest priority

(6) Multilevel Feedback-Queue scheduling – allow process to move between queues

It gives highest priority to any process with a CPU burst of 8 milliseconds or less. It is defined by the following parameters –

(1) The number of queues
(2) Scheduling algorithm of each queue
(3) method used to determine when to upgrade a process to a higher-priority queue.
(4) method used to determine when to demote a process to a lower-priority queue.
(5) method used to determine which queue a process will enter when that process needs service.

⑥ Algorithm Evaluation –

① → Maximizing CPU utilization under the constraint that the maximum response time is 1 second.

② → Maximizing throughput such that turnaround time is (on average) linearly proportional to total execution time.

→ Deterministic Modeling

→ Queueing Models → little's formula ($n = \lambda \times W$) (λ: average arrival rate, W: waiting time)

→ Simulations

→ Implementation → code up the algorithm & put it in OS.

⑦ Multiple Processor scheduling –

→ asymmetric multiprocessing – scheduling decisions, I/O processing, and other system activities handled by a single processor → master server. Other processors execute only user code

→ Symmetric multiprocessing – Each processor is self-scheduling.

→ Processor affinity – A process has an affinity for the processor on which it is currently running.

→ soft affinity – running on same processor, but not guaranteeing it will do

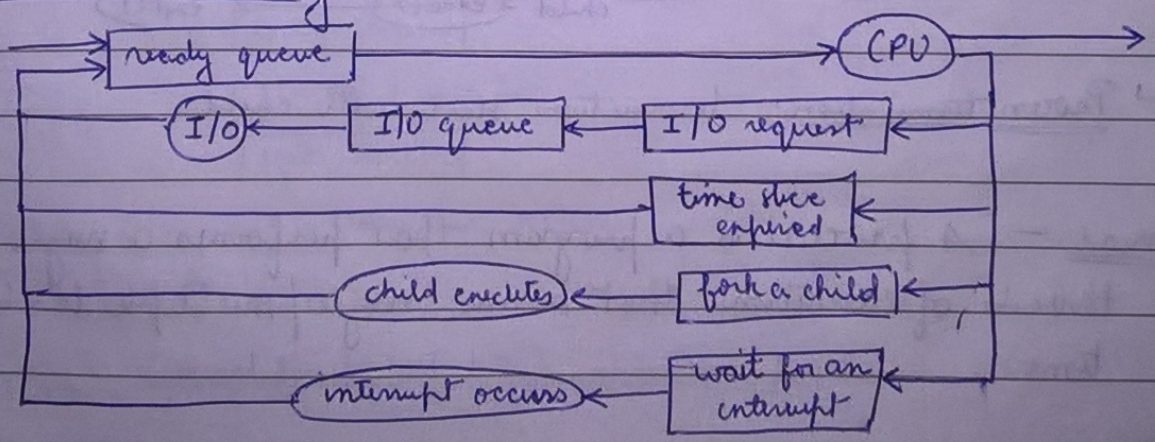
→ hard affinity – specify not to migrate to other processors

→ load balancing – Evenly distributed across all processors

→ push migration – occurs when task periodically checks the load on each processor

→ pull migration – occurs when an idle processor pulls a waiting task from a busy processor.

⑧ Process Scheduling –



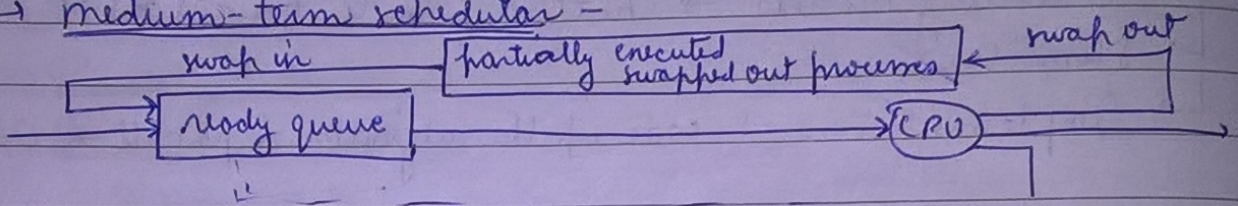
Schedulers -

→ long-term scheduler or job scheduler - select processes from the pool and loads them into memory for execution.

→ short-term scheduler or CPU scheduler - select from among the processes that are ready to execute and allocates the CPU to one of them.

Degree of multiprogramming - No. of process in memory.

→ medium-term scheduler -



Context switch - Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.

I/O bound process - Spends more of its time doing I/O

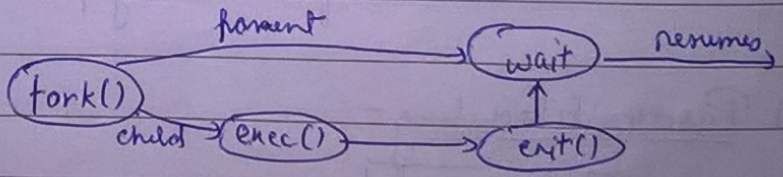
CPU bound process - Spends more of its time doing computations.

(9) Real time scheduling -

- Static priority scheduling and Dynamic priority scheduling
- scheduling decision based on fixed parameters
- assigned before their activation
- Eg - highest priority scheduling
- scheduling decision based on dynamic parameters
- Change during system evolution
- Eg - Earliest deadline scheduling

(10) Operation on processes -

→ Process Creation -



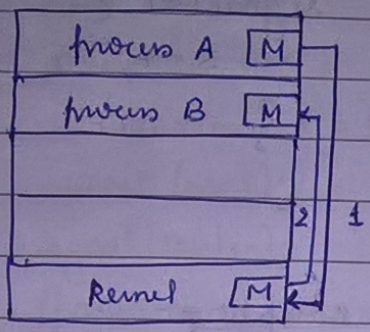
→ Process termination - termination start with child

(11) Threads - A process is a program that performs a single/multiple threads of execution. That means single/multiple tasks at a time.

(12) Inter process communication –

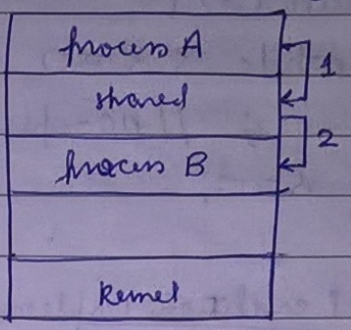
↳ reasons – Information sharing, computation speedup, modularity, convenience

Two models – Shared memory and message passing.



Message passing systems

Eg :- chat program



Shared memory systems

Eg :- producer - consumer.

(13) Precedence Graphs –

(1) Parbegin / Parend or cobegin / coend → limitation – cross dependency

(2) Fork and join construction (3) Semaphore variable

(14) Critical Section Problem –

Critical section is a segment of code in which the process may be changing common variable, updating a table, writing a file and so on

→ No two processes are executing in their critical sections at the same time.

Solution to critical section problem –

(1) Mutual Exclusion

(2) Progress

(3) Bounded Waiting

```
do {
    entry section
    critical section
    exit section
    remainder section
} while (true);
```

General structure of a typical process P_i

(15) Semaphores - (synchronization tool)

A semaphore S is an integer variable is accessed only through two standard atomic ~~functions~~ operations: wait() & signal().

```
wait(S){
    while S<=0
        ; // no-operation
    S--;
}
```

```
signal(S){
    S++;
}
```

Mutual exclusion implementation with semaphores :-

```
do {
    waiting(mutex);
        // critical section
    signal(mutex);
        // remainder section
} while (TRUE);
```

Two types of semaphores → counting semaphore & binary semaphore (also known as mutex locks)

Implementation problem → deadlock and starvation.

(16) Classical Problems of Synchronization -

(1) Producer-Consumer Problem - semaphore mutex, full, empty;

Producer Process -

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

Consumer Process -

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

(2) Readers-Writers Problem - semaphore muten, wrt; int readcount;

Reader Process -

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

Writer Process -

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

(3) Dining - Philosophers Problem -

Philosopher i - semaphore chopstick [5];

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

→ Deadlock -

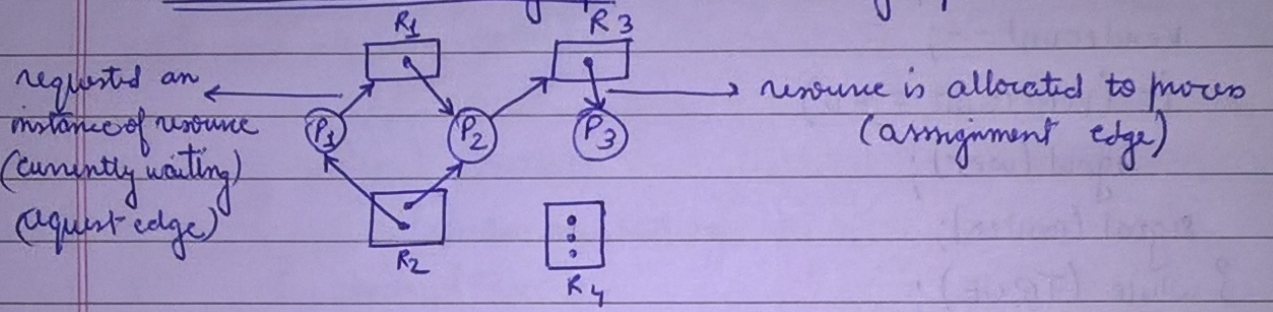
In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs for starting.

① Deadlock Characterization -

→ Necessary Conditions - four conditions hold simultaneously

(1) Mutual Exclusion
(2) Hold and Wait
(3) No preemption
(4) Circular Wait

→ Resource allocation graph - directed graph



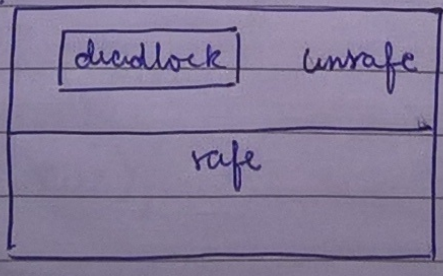
② Methods for handling deadlocks -

(1) Use a protocol to prevent or avoid deadlocks
(2) Allow the system to enter a deadlock state, detect it, and recov
(3) Ignore the problem altogether

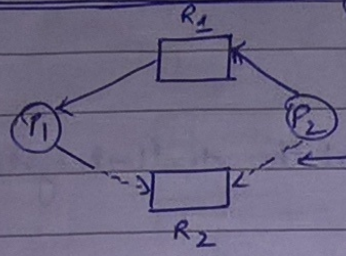
③ Deadlock prevention - By ensuring that at least one of ~~these~~ fo necessary conditions for deadlock cannot hold, we can prevent t occurence of a deadlock.

④ Deadlock Avoidance -

→ Safe State - A system is in a safe state only if there exists a sequence.



→ Resource allocation graph Algorithm –



← claim edge (it may request resource in future)

→ Banker's Algorithm –

Need[i,j] = Max[i,j] – Allocation[i,j].

Available[i] = Total instance[i] – Need[j].

→ Safety Algorithm –

(1) Work = Available and Finish[i] = false.

(2) If (Finish[i] == false AND need[i] ≤ work)

then {

{Work = Work + allocation

Finish[i] = true

} repeat step 2; }

else if no such i exists goto step 3.

(3) If finish[i] == true for all i's then system is in a safe state

→ Resource-request algorithm –

If request[i] ≤ Need[i]

then if request[i] ≤ Available

then available = available – request;

Allocation = Allocation + request

Need = Need – request

Otherwise request not granted.

(5) Deadlock detection – → n^2 operations needed to detect

→ single instance of each resource type – Make a wait for graph

if there is a cycle in wait for graph then deadlock occurs.

→ several instance of a resource type – Use bankers algorithm

$m \times n^2$ operations needed to detect

⑥ Recovery from deadlock –

→ Process termination –

- abort all deadlocked process
- abort one process at a time until the deadlock cycle is elimin

→ Resource Preemption –

issues → Selecting a victim, Rollback, Starvation