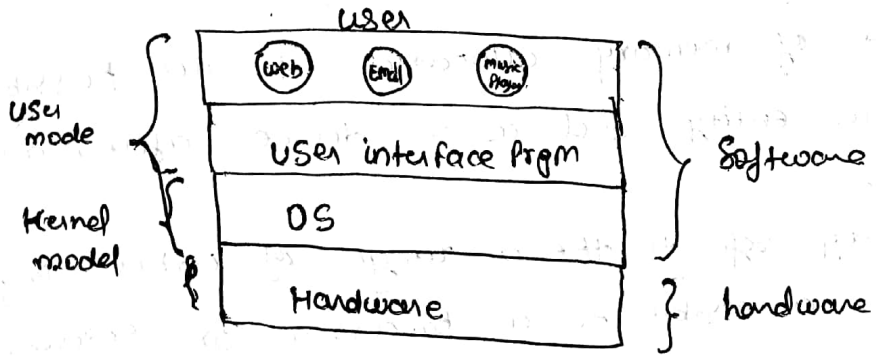
Operating System

An operating System is a program that manage the computer hardware. it also provides a basic for application progam and acts as an intermediary between a user of a computer and the computer hardware.

OS is an interface b/w user and hardware.

Hierarchy



- OS is decides the which resources should run and for how much time. that is why it is called as "resource allocator".

- OS is a Control program:

→ control execution of program to prevent errors and improper use of the Computer.

* The one program running at all times on the computer, is the "Kernel". everything else is either system program or application program.

→ The main aim of kernel is to manage the communication between the software and the hardware.

It is an Software which perform all the basic task like Process management, file management, m/m management, I/o device management, and Control the peripheral devices.

* Operating System would take the appropriate action to remove the errors, like THE error may be in CPU & m/m hardware, network failure lack of paper in the printer etc.

* OS also use to keep track which users use how many and which kind of Computer resources.

* Operating System Concept:

Process: logical unit of program which is currently in execution is called "process". Every process reside in it's corresponding address space.

- All the information about each process other than the contents of its own address space, is stored in an operating system table called "process table".

Address Space:

- it is the amount of memory allocation for all possible addresses for a computational entity, such as a device, a file, a server, or a networked computer.

Address space may refer to a range of either physical or virtual addresses accessible to a processor or reserved for a processor.

Shell:

A shell is a user interface for access to an operating system services. in general OS shells use either a command-line interface or graphical user interface (GUI) depending on a computer's role and particular operations.

with the exception of GUI also.

* System Call:

- it provide the interface between a process and the operating system.
- it is a request made the user program to the OS in order to get any kind of service.
- mostly accessed by programs via a high level application program interface (API) rather than direct system call use.
- Three most common API's are Win32 API for windows, POSIX API for POSIX-based system and Java API for the Java virtual machine (JVM)

eg
```
main()
{ printf(" ");
  scanf(" "):
}
```
internally call write() system call in order to communicate with the monitor.

can pass parameter to the system call (read & blc).
The System call interface invokes system call in OS kernel and returns status of the system call and any return values.
System call can be grouped roughly into five major categories:

1) process control: A running program needs to be able to halt its execution either normally or abnormally.
→ load and execute the program.
→ get process attribute and set process attributes.
→ wait for time.

2) File management: System call can
• create and delete file. with the name of the file and perhaps some of the file attributes.
• open and close

Device management.
info. maintance.
Communication. (read from blc.)

Tyes of OS:

Batch OS. (use very long back. Job which are similar type are treated as batch. and that will process one by one. Process has to wait until all the process ahead it will get finish.) So if there is a i/o job for any Processor that will take lot of time & other process has to wait due to that.) disadv: lack of interaction. debug diffi.

multitasking (extention of multiprogramming) (multiple job can be executed on the system at the same time by sharing the CPU time among them.)
disadv: Synchronization.

multiprogramming: keep the CPU always busy. in Batch I/O required more time and until that CPU were waiting. but free during that CPU will do other job.

multiprocessor: there are lot of CPU in same computer. & that can perform more than one task at time. i.e parallelism is included. throughput Yes.
complete the tasks within given time span.

Process Management :

- A process is just an instance of ~~sequential processes, or just~~ an executing program, including the current values of the program counter, registers, and variables.

→ An Operating System executes programs :

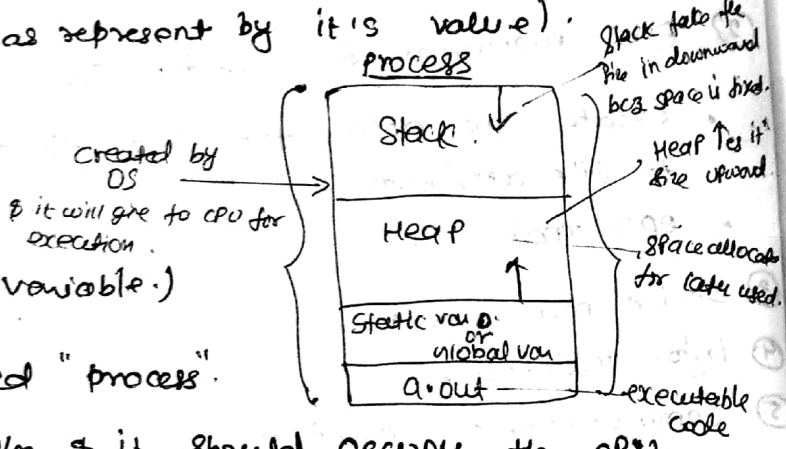
- A batch system executes jobs where as time shared systems has user program or task.

* A process includes :

→ program code (called the text)
→ program counter (current activity, as represent by it's value).
→ runtime stack
→ data section
→ process state
→ Heap (for dynamically allocated variable.)

or

Program under execution is called "process".



→ It has to reside in the main m/m & it should occupy the CPU.

Attribute of process :

Process id : it is unique identification no. which is assigned by the OS at the time of process creation.

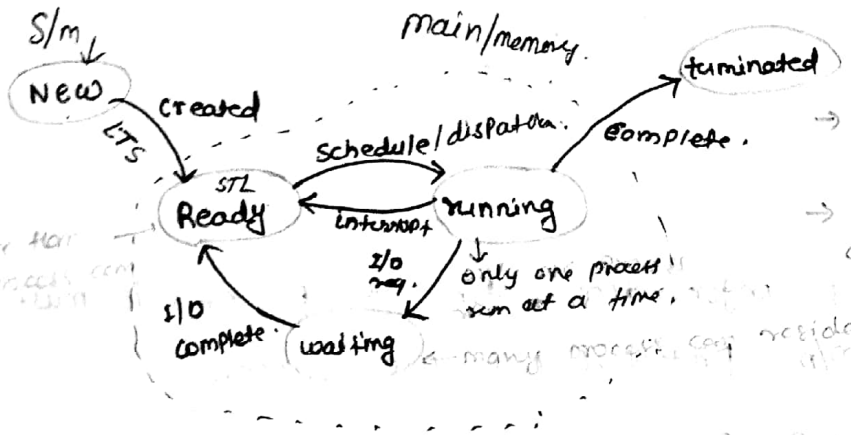
- **Process counter** : Contains the address of the next instruction to be executed.
- **Priority** : it is a parameter which is assigned by the OS at the time of process creation. higher priority will execute first - eg OS process having higher priority than user priority.
- **Cereral purpose reg** : it contains reg. info used by the process in order to execute the instruction.
- **List of open file** : it contains the info about the file which is currently in working.

List of open devices : it contains the details about the devices which is currently on execution.

- **Protection info** : it contains all the info. about the Security.

Process State:

It contains the current state information of the process where it is residing.



New: The process is under creation or process is being created.

Ready: There is no. of process waiting to be assigned to a processor.

→ one process is selected from ready state and it will be dispatched from ready state to running state. ready state having multiple processes.

Running: Instructions are being executed. Here only one process at any point of time.

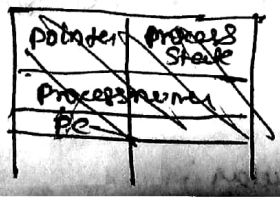
Waiting: When running process requires I/O operations, it will be moved to wait state / block state it can have multiple processes.

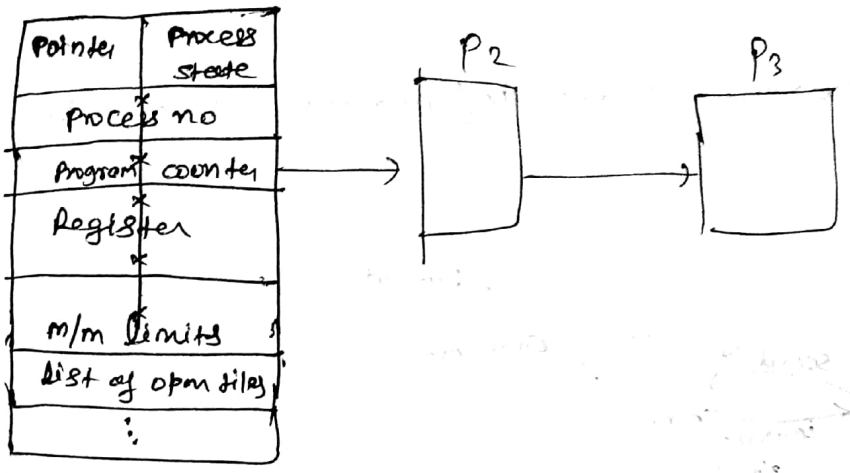
* the CPU time of the process will be spent in running state and I/O time in waiting state.
* when the process is in ready, running & waiting state, it is residing in the main memory

Terminated: The process has finished execution.

* Process control block (PCB):

- PCB is the data structure containing information associated with each process, needed by OS. and all the process is connected by linked list.
- it contains all the attributes of the process mentioned above





- PCB contains the critical information for the process, it must be kept in an area of m/m protected from user access.
- PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Process Scheduling:

- it is act of determing which process in ready state should be moved to the running state is known as process Scheduling.
- The main aim of the process Scheduling System is to keep ~~track~~ the CPU busy all time and to deliver minimum time for all program.

* Sheduling queues: it maintain the information of all ready process for CPU.

→ Job queue: Set of all processes in the System.

→ Ready queue: Set of all processes residing in main memory, ready and waiting to execute.

→ Device queue: Set of all process waiting for an I/O device.

* Process migrate among the various queues.

* Schedulers:

- it is a System Software which handle 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.

Type of Scheduler.

① Long term Scheduler: or Job Scheduler:

LTS is responsible for creating new process and bringing them into the system (ready state). it control multi-programming it selects the processes from this pool and load them into m/m for execution.
it runs less frequently.

Short-term schedular: STL → dispatcher.

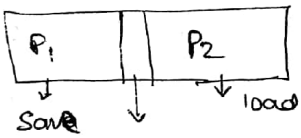
it is responsible for selecting one process from ready state for scheduling it on the running state. it only selects the process to schedule it doesn't load the process on running.
This is also known as CPU schedular and run very frequently.
The primary aim of this schedular is to enhance CPU performance and increase process execution rate.
Dispatcher is responsible for loading the selected process by STS on the CPU.

Medium term schedular:

it is responsible for suspending and resuming the process. it is also called as swapper.
It is also an intermediate level schedular used in time sharing os.

Context switching:

Saving the context of 1 process and loading the context of another process is called CONTEXT SWITCHING.



Here we context ← attribute of process. the P_1 to P_1.

context switching time { whenever the context switch occurs, the kernel saves the context of the old process in it's PCB and loads the save context of the new process schedule to run.}

Each and every time when process is moving from one state to another the context of the process will change.

minm process req. for context switching is 2 (except in RR scheduling algo).

The context switching will also take some time, so if context of the process ↑ time ↑es. which is undesirable.
context switching time is pure over_head. and

Process

Intra process (with single process communication) — How to communicate a single process.

Inter process.

* **Inter Process Communication (IPC)**: O/P of one process transfer to other inorder to speed up the execution.

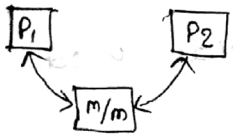
- it is a mechanism by which one process can communicate or exchange data with another process.
- Processes can communicate with each other using these two ways.

1) Shared memory
2) message passing.

Purpose of IPC:

- Data transfer or information sharing.
- Computation speed up.
- Event notification & resource sharing.
- Process control
- Synchronization

① **Using Shared memory**: it is a simple on fastest form of IPC.



- Here all the process shares same peice of memory. So Synchronization is needed. (only one process should be allowed to read/write at at time)
- **Terms:**

Critical Section:

* it is the part of a program where shared resources are accessed by processes. eg Buffer & count are the shared resource in Producer Consumer.
- A group of cooperating processes (dependent processes), at a given point of time, only one process must be executing it's critical section. if other process want to execute it's critical section, it must wait until the first one finishes.

Race condition

- it occurs when several processes access and manipulate shared data concurrently.
- The order of the execution of instructions influences the result produced.

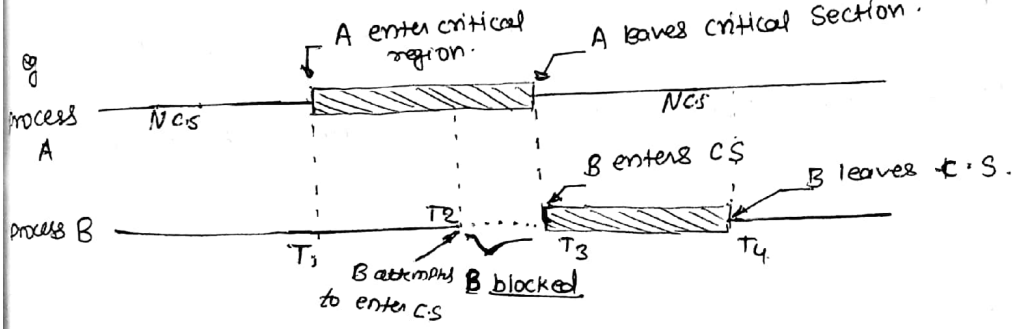
To avoid the race ... mutual exclusion

mutual exclusion: it ensure that there is no prblm of race cond^n among processes.

it is a mechanism by which in the multiprogramming environment when two processes share access the shared resource simultaneously then it prevents the simultaneous access to a shared resource.

Assumption: (to ensure the Synchronization)

1. NO two processes may are permited to enter the critical section parallely.
2. A process in it's non critical section should not prevent another process from entering it's critical section.
3. NO assumption is taken regarding the speed of the process or no. of processes.
4. NO process should have to wait indefinetly for getting a it's turn for execution. (Starvation)

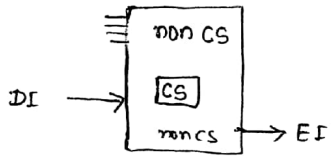


So from this example we shoco that A enters it's CS at time T_1, at time T_2 process B attempts to enter it's critical region but fail. bez another process is already in it's critical region.

How to implement mutual exclusion:

Busy waiting mechanism: (means continuously knocking the door. it may lead to Starvation

1. **Using enabling & Disabling interrupts**: (Done in Kernel mode. it means OS will do the part of Job.



→ on a single-processor system, when processes enters it's C.S then disable all interrupts and enable interrupts when it leaves critical region.

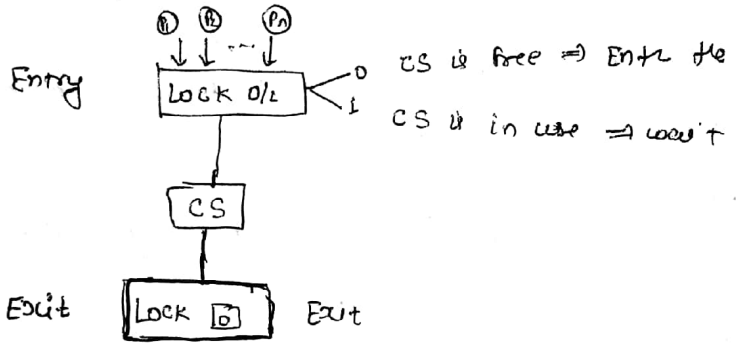
Problems: (we can not give interrupt control to user so it is rarely used.)

- Since other process will continue running and can access the shared m/m. so that race conditions may occur.

② Lock variables:

- Software mechanism implemented in user mode.
- it arises in the case of multiprocesses.
- Lock is a shared binary variable either 0 or 1.

Lock = 0 (critical section is free)
Lock = 1 (critical section is in use).



* Processor has to check the Lock value before entering into CS.
* IF lock value = 1 ⇒ the process just waits until it becomes 0.
* IF lock value = 0 ⇒ process can enter into CS and make Lock = 1.

Implementation:

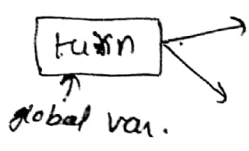
```
int     Lock = 0;

void  CS (int process)
{
    while( Lock != 0);
    {
        Lock = 1

        CS
    }
    void  exitSection(int process)
    {
        Lock = 0
    }
```

Problem: Race condition may occur.

- Suppose that one process read the Lock value as 0 and it enters into CS but before it can set the lock to 1, another process is scheduled, runs and set the lock to 1. when first process runs again, it will also set the lock to 1 and two process will be in their CS at same time.

② Strict alternation Software mech. used in user mode.

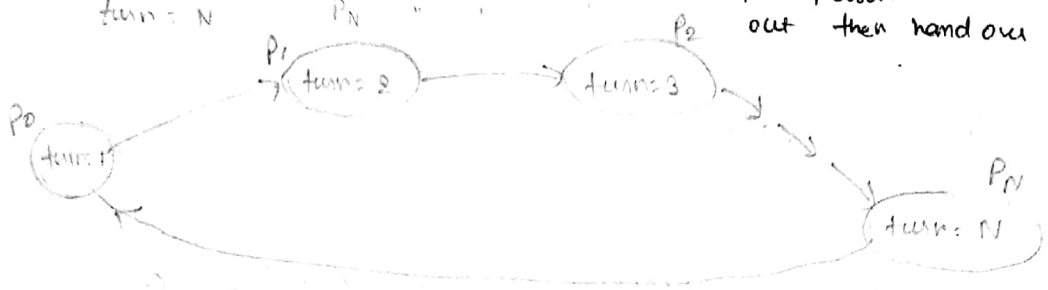
- it is a two process Solution. (P_i, P_j)
- Here we use turn variable like Lock variable in previous one.



0 Process P_0 turn to enter CS. (P_1 is Busy waiting) while leaving the CS, P_0 set turn to 1

1 Process P_1 turn to enter CS, (P_0 is Busy waiting) while leaving the CS, P_1 set turn to 0.

eg it like real Lock eg when the Person enter to CS it takes the key with him so that another person will no get enter. when he come out then hand over key to other process.



- Here every Process will must be enter in its CS one by one

P_0

```
algo   while (true) {
Entry → while (turn != 0) ; (He has to wait)
        enter-CS();
EXIT  → turn = 1;
        }
```

P_1

```
       while (true) {
       {
Entry → while (turn != 1) ; (He has to wait)
        enter-CS();
EXIT  → turn = 0;
        }
```

- Here turn value will not immediately update bcz it updated on EXIT Sec.
- it gurantee the mutual exclusion bt not progress.

Problem:

it is avoiding the condition 2. Race condition occur.

Suppose initially turn = 0, P_0 is not interested to enter into CS, P_0 is running in non CS but P_1 is intrested to into into CS bcz turn = 0.

Suppose P_0 enters into the CS and while exiting turn is change to 1, P_1 needs to enters, it again P_0 want to enter CS & P_1 is not intrested. P_0 can not enter into CS bcz turn = 1.

④ Peterson's Solution : (it will satisfy all the conditions.)

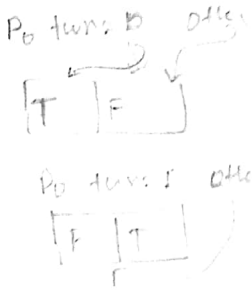
- It is Combination of LOCK variable + STRICT alternation.
- It is also implemented in user mode.
- Here we use two variable one is the turn and other is array
- 2 Process solution.

```
# define Falso 0
# define TRUE 1
# define N 2 |no. of process.

int turn;
int interested[N];

void enter_region(int Process);
{
    int other;
    turn = Process
    interested[Process] = TRue ;
    Other = 1- Process
}
while( turn == Process $$ interested[other] == TRue ); CS

void exit-cs(int Process)
{
    interested[Process] = False ;
}
```

enter-CS() wait loop

CS

exit-CS()



* Here each Process before entering critical region. calls enter-region(int Proces
after leaving the C.S call exit-cs(int Process)

- Here mutual exclusion and Progress both are guarantee.

Dis adv:

→ ~~mutual exclusion~~ is Soln available only for 2 Processes.

→ Busy waiting : wastage of CPU cycle.

⑤ Test and SET Lock (TSL) :

- Software mechanism implemented at user mode.
- mutual exclusion with busy waiting.

TSL register Flag → 0 ⇒ critical section is free
→ 1 ⇒ critical section is in use.

- TSL is a privileged instruction.
- One atomic operation at a time.
- Execution will be done without Preempting.

• it is applicable for more than 2 processes.

Assembly lng.

Algo

```
enter_CS()
{TSL register, Flag    // copy flag to reg. and set flag to 1.
 CMP register, #0      // was Flg zero?
 JNE enter_CS          // if it was nonzero, lock was set, so loop.
 return                // return to caller, critical region entered.
}

exit_CS()
{ move Flag, #0        // stor a 0 in flag
  RET                  // return to caller.
}
```

it will wake for the flag value as 1. then atomic operation.

dis adv.

* Busy waiting ⇒ Priority inversion problem. It means when High priority process comes & LP is in the C.S then

* Using waiting process suspension:

► Sleep and wakeup:

(waiting process will sleep instead of knocking door in busy waiting. once process complete that will wake up the sleeping process.)

• Here we create two system calls sleep() and wakeup()

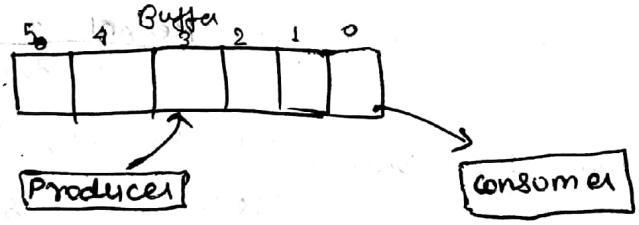
• Sleep is a system call that causes the caller to block, that is, be suspended until another process wakes it up.

• wakeup calls has one parameter, the process to be awakened.

* Producer/consumer problem: (bounded-buffer problem)

• Two processes one is producer and other is consumer shares a common fixed-size buffer.

→ Producer will put pieces of information in the buffer. while consumer will take pieces of information out of the buffer.



Problem arises when buffer is full or empty. is it is full then, or is it is empty.

• Producer sleeps if no space in buffer. Consumer wakes it up when there is space.

• consumer sleeps if no data in buffer. producer wakes it up when

Algo

```
# define N 100   // no. of slot in buffer
int count = 0

Void producer()
{
 while(true)
  {
 int item = Produce_item();
   IF(count == N)
    sleep();
   else
    insert_item(item)
     count++
   IF(count == 1)
     wakeUP(consumer);
  }
}
```

```
void consumer()
 {
  while(true)
    {
    IF(count == 0)
     sleep();
    else
    item = remove_item()
     count--;
     if(count == N-1)
       wakeUP(producer)
    }
 }
```

Problem: Lost wake up signal leads to deadlock.

(wakeup signal will lost when let producer send the wakeup signal consumer but he did'n slelt yet. he Il just about to sleep on that case wakeup signal will lost & Both of them sleep forever.

2. Semaphores: (currently implement in every os.).

- E.w. Dijkstra proposed integer variable to count wake up signal.
- it is an integer datatype to represent the pending wake-up signals. (Befor moving to sleep state consumer has to look the value of semaphore.)
- Semaphore is a process synchronization tool.
- Semaphores can be used to implement mutual exclusion among any number of processes.
- Semaphore operations are execute in Kernel mode automatically. i.e done by O.S using system call
- Semaphore could have the value → 0 (representing no wakeups were saved)
  → >0 (represents some wakeup were pending.

- Two Operations of Semaphore
  1) down (semaphore)
  2) UP (semaphore)

two types of Semaphore.
1) Counting Sem.
2) Binary Semaphore (mutex).

case 1: IF Semaphore = 0

no pending wakeup signal so process can enter into sleep() state.

case 2: IF Semaphore != 0

one or more pending wakeup signal exist, decrement the value of Semaphore by 1, without entering the sleep() state the process should continue.

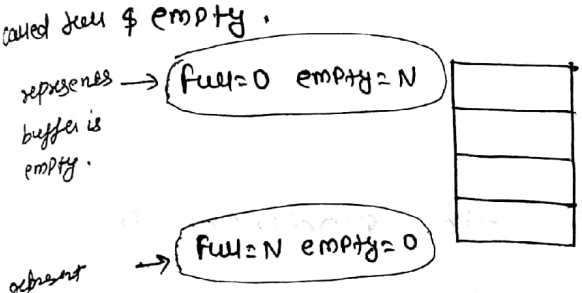
2) UP(Semaphore) → (symbol). Signal.

- increment the value of the Semaphore by 1.
- If any process is sleeping as the Semaphore value is zero a wakeup signal is sent to one of them.

Semaphore operations are done as a single atomic action.

Solving the Producer-Consumer Problem using Semaphore:

- Semaphore solve lost ~~cnet~~ wake up problem. Here we use to Semaphore called full & empty.



full → counting the no. of slot that are full.
empty → counting the no. of slot that are empty

- Initially full = 0 and empty is equals to no. of slot in the buffer.

```
Producer()
{
down(&empty)
insert_item()
UP(&full)
}
```

```
Consumer()
{
down(&full)
delete_item()
UP(&empty)
}
```

- Here the order of UP & down operation can't be reverse.

Mutexes:

- Semaphore guarantee the mutual exclusion by using third integer variable called mutex.
- A mutex is a variable that can be in one of two states unlocked or locked. i.e mutex is a binary semaphore.

if mutex → 0 (unlocked)
→ 1 (locked)

```
Producer()
{
down(empty)
down(mutex)
insert(item)
UP(mutex)
UP(full)
}
```

```
consumer()
{
down(full)
down(mutex)
delete_item()
UP(mutex)
up(empty)
}
```

Now

* Semaphore full & empty guarantees the producer will be sleeping when the buffer is full and consumer will be sleeping when the buffer will be empty.

* The third Semaphore mutex ensure synchronization. i.e one process has entered the critical section (CS) no other process is entering the critical section.

monitors:

It is a collection of procedure, variables, and conditions that are all group together in special kind of module or package. monitors have an important property that makes them useful for achieving mutual exclusion: only one process can be achieve in a monitor at any instant.

It was proposed to introduced conditional variable with two operations sample wait(var) and signal(var)

monitor example

```
integer i
condition c;
procedure producer();
  :
  sample-insert();
  :
end
procedure consumer.
  :
  sample.remove();
  :
end
```

Program: Of Producer Consumer using monitor:

```
monitor producer&consumer
  condition full, empty;
  integer count;

procedure insert(item: integer);
begin
    if(count=N)
      wait(FULL);
    else
      insert:item(item);
      count++
    if(count=1)
      signal(empty)
end
```

* Binary Semaphore : (mutex)

- it can have two values only. one represent the process/thread in the critical section & others should wait the other indicating the CS is free.

```
Struct BSemaphore
{
   enum { value (0, 1) ; //only two value 0 & 1 are allowed.
   Queue type L;
}

Down( BSemaphore S)
{
   if (Svalue == 1) // means no one has entered in the critical section.
     S.value = 0;
   else
   { Put the process
      PCB in S.L;
     sleep();
   }
}

UP(BSemaphore S)
{
   if (S.L is empty) // No process is blocked as of now
      S.value = 1
   else
   { Select a process from S.L;
     wakeUP();
   }
}
```

* Counting Semaphore :

- it takes more than two values they can have max^m value that a Counting Semaphore can take is the no. of process you want to allow into the critical section at the same time.

- Struct Semaphore
{
  int value // no. of process can enter into the critical section if value > 0 else some processes are blocked based on their value.
  Queue type L; → Contains all PCBs corresponding to processes got blocked while performing down operation unsuccessfully.
}

- Down(Semaphore S) // whenever any process want to enter to CS then apply down.
{
  S.value = S.value - 1
  if (S.value < 0) // it means already some process are waiting (blocked)
  {
    Put Process (PCB) in L;
    Sleep();
  }
  else
    return;
}

- UP(semaphore S) // whenever any process want to come out of C.S then apply UP.
{
  S.value = S.value + 1
  if (S.value ≤ 0) // it means some process are waiting.
  {
    select a process from
      L.
    wake up();
  }
}

Q) A counting Semaphore was initialize to 10. Then 6P (wait) & 4V (signal) operations were computed on this Semaphore what is the result.

$S = 10 \quad 10 - 6 = 4 \Rightarrow 4 + 4 = 8$ (6 down, 4 up)

Q) $S = 7$ then 20P, 15V $S = ?$

$S = 7 - 20 = -13 + 15 = 2$

```
Function remove : integer
begin
        If count = 0
          wait (empty);
        else
          remove = remove - item();
          Count - -
        if Count = N-1
          Signal (full)
end
Count = 0
end monitor
```

```
Procedure producer;
begin
    while(true) do
    begin
        item = produce-item;
        producerConsumer.insert (item)
    end
end
```

```
procedure Consumer
begin
        while(true) do
        begin
            item= producerConsumer.remove;
            Consume_item(item)
        end
end
```

* When monitor procedure discoves that it is not possible to Continue some operation, wait is performed on some condition variable. Process which executes procedure is suspended.

* For process to enter in the CS must received the signal( ) in order to leave of the process or some conditional var

Message Passing :

This method of IPC uses two primitives, send and receive which is like semaphore and unlike monitors are system call rather than language constructs.

→ Send (destination, &message);

→ receive (source, &message);

Different methods of message addressing : (design issue)

1. direct addressing or
   - indirect addressing
2. Synchronous or asynchronous
3. Symmetric or asymmetric ..

How the link is established
How many links are there with
it the capacity of each
what is the size of msg
is a link bidirectional or uni

Producer - consumer problem with message passing :

```
void producer (void)
{
    int item;
    message m;
    while (TRUE) {
        item = produce_item();  → Generate something to put into buffer
        receive (consumer, &m);  → wait for an empty to arrive
        build_message (&m, item);  — construct a message to send
        Send (consumer, &m)  — Send item to consumer
    }
}

void consumer()
{
    int item, i;
    message m;
    for (i=0; i<n; i++)
        send (producer, &m)
```

```
while (TRUE)
{
    receive (Producer, &m)
    item = extract_item (&m);
    Send (Producer, &m)
    consume_item (item);
}
}
```

CPU Scheduling

- We know that the obj. of multiprogramming is to have some process running at all times, in order to maximize CPU utilization.
- CPU Scheduling is the process by which we can decide the which process from ready queue should move to the running state.
- This is done by the Short term Scheduler & dispatcher.

When to Schedule :

→ If Process move from run to termination. (Then we need Scheduling for the which process should come in run state.)

→ If process move from run state to wait state. (due to some I/O oprn or Particular state. If OS is preemitive nature then it will happen. Here forcefully we send the process to read state.)

→ wait to ready state (Sometime we need to send the highest priority process.)

→ New to ready.

Goal

1. Increase the CPU utilization and throughput of the system.
2. minimum the avg. time, avg. turn around time of the process.
3. min. response time.

* **Scheduling criteria :** following many criteria have been ~~Suggested for cpu scheduling algo~~

1. **CPU Utilization :**

We want to keep the CPU busy as busy as possible. CPU usage should ~~zero~~

ranges from 40% to 90%.

Throughput:

The no. of processes completed it's task per unit time is called throughput. it may ranges from 10/s to 1/h depending on the specific process.

Turnaround time:

it is the amount of time taken to execute a particular process i.e the interval from the time of submission of a process to the time of completion is the turnaround time.

Waiting time:

it is the sum of the periods spent waiting in the ready queue

Response time:

Amount of time it takes from when a request was submitted until the first response is produced.

we want to maximize CPU utilization, throughput and to minimize turnaround time, waiting time and response time.

Some basic terms:

- **Arrival time (A.T):** The time when the process is arrived into ready state is called as arrival time of the process.
- **Burst time (B.T):** the time required by process for it's execution is called as burst time of the process.
- **Completion time (C.T):** the time when the process is completed or completes it's execution is called as completion time.

we have also

turn around time = completion time − Arrival time
waiting time = turn around time − Burst time.

response time = the time difference between the 1st response and AT.

* Scheduling algorithms :

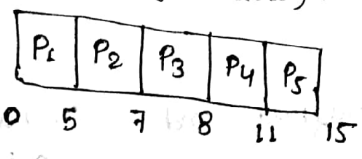
① First - come, First served algorithm :

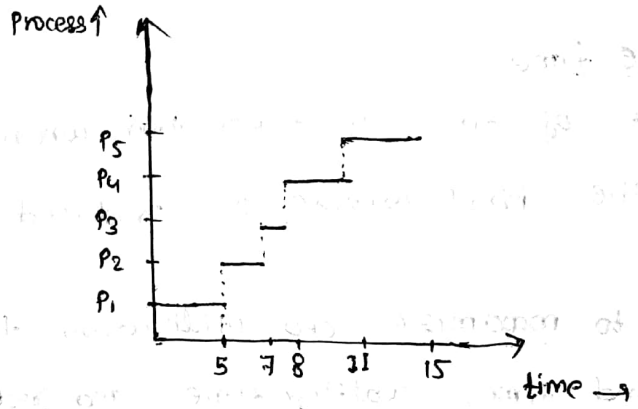
- It is a simplest scheduling algorithm that schedules a/c to arrival time of processes.
- Job are executed on the First come, First serve basis.
- it is non preemptive scheduling algo. (i.e it pick a process to run and then just lets it run until it blocks or until it voluntarily rele the CPU.).
- if two process arive at same time than pick one whose Job is small.

eg.

Process	burst time.
P_1	5ms
P_2	2ms
P_3	1ms
P_4	3ms
P_5	4ms

GANTT - Chart (always starts from zero).





T.A.T = W.T + B.T
= C.T - A.T ← not given so use

P.NO	C.T	T.A.T	W.T
1	5	5	0
2	7	7	5
3	8	8	7
4	11	11	8
5	15	15	11

Avg turn around time = $\frac{\text{total T.A.T}}{\text{no. of process}} = \frac{46}{5}$

Avg waiting time = $\frac{\text{total W.T}}{\text{no. of process}} = \frac{0+5+7+8+11}{5} = \frac{31}{5}$

Adv.

- Suitable for batch system.
- it is simple to under_stand and code.
- FIFO queue data structure can be use here.

disadv.

- Poor in Performance as average wait time is high.
- it is not suitable for time sharing system where it is imp that each user should get the CPU for an equal amount of time interval.

eg

P.NO	A.T	B.T	C.T	TAT	W.T
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

TAT = CT − AT

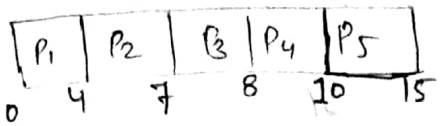
WT = TAT − BT

Avg TAT

Avg waiting time = $\frac{0+3+5+5+6}{5}$

$= \frac{19}{5}$

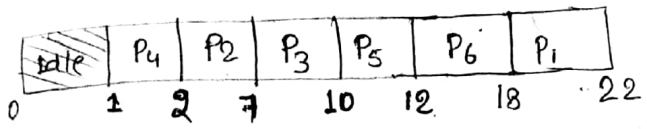
Gantt chart



eg

P.NO	A.T	B.T	C.T	TAT	W.T
1	6	4	22	16	12
2	2	5	7	5	0
3	3	3	10	7	4
4	1	1	2	1	0
5	4	2	12	8	6
6	5	6	18	13	7

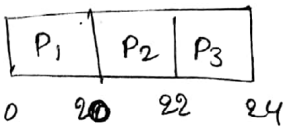
Avg waiting time = $\frac{12+0+4+6+7}{6} = \frac{29}{6}$

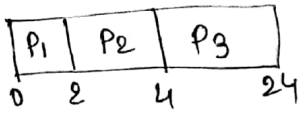


eg (a)

Process	A.T	B.T	W.T	TAT (b)
P1	0	20	0	20
P2	1	2	19	21
P3	2	2	20	22

Process	A.T	B.T	W.T
P1	0	2	0
P2	1	2	1
P3	2	20	2





Avg waiting time = $\frac{0+2+2}{3} = 1.$

Avg waiting = $\frac{0+20+22}{3} = 13$

- This is called convey effect. (If starting process have large burst time).
- In FCFS if the 1st process is having largest burst time then it will have drastic effect on average waiting time of all process. This effect is called as "convey effect". [It causes low CPU utilization]
- When all process arive at same time then:

Submission time = waiting time = completion time = turnaround time.

2. Shortest Job First (Next): Schedule a/c to Shortest Job.

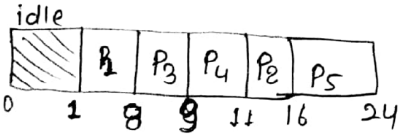
- It is a Scheduling policy that selects the waiting process with the smallest execution time to execute next. i.e it allows first to the process having shortest Job. (for execution).
- This is a non-preemptive, preemptive Scheduling algorithm.
- it is a greedy algo

***Non-Preemptive** (Runs the process till completion)

eg

	A.T	B.T	C.T	TAT	W.T
P1	1	7	8	7	0
P2	2	5	16	14	9
P3	3	1	9	6	5
P4	4	2	11	7	5
P5	5	8	24	19	11

GANTT-chart



Avg waiting time $= \frac{0+9+5+5+11}{5} = \frac{30}{5}$

Avg TAT $= \frac{7+14+6+7+19}{5} = \frac{53}{5}$

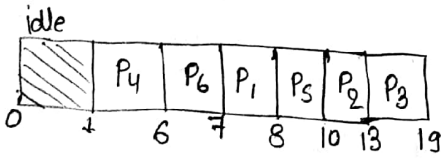
- At A.time = 1 there is no complecations about Shortest Job. bc2 on that time only one process i.e P_1 is available in the ready queue. after that P_1 takes 7 unit of B.time to complete it's execution during this execution the process arrived, then now we have schedule it a/c to Shortest Job.

* If the burst time of the processes are same then we have to go through the FCFS.

- min heap data structure might be use for implementation.

eg.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	6	1	8	2	1
2	3	3	13	10	7
3	4	6	19	15	9
4	1	5	6	5	0
5	2	2	10	8	6
6	5	1	7	2	1



Throughput $= 19\left(\frac{6}{19}\right)$ — 19: total no. of process; 6: total time req.

Average waiting time $= \frac{1+7+9+0+6+1}{6} = \frac{24}{6} = 4$

Avg T.A.T $= \frac{2+10+15+5+8+2}{6} = \frac{42}{6} = 6$

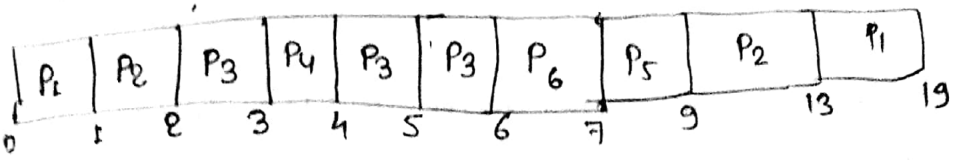
- Convoy effect might also occur here (But it is not mentioned in any book.)

Pre emptive (SRTF) Shortest remaining time first.
Strategy of allowing processes that are logically runnable to be temporarily suspended.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	0	7	19	19	12
2	1	5	13	12	7
3	2	3	6	4	1
4	3	2	4	1	0
5	4	1	9	5	3
6	5	1	7	2	1

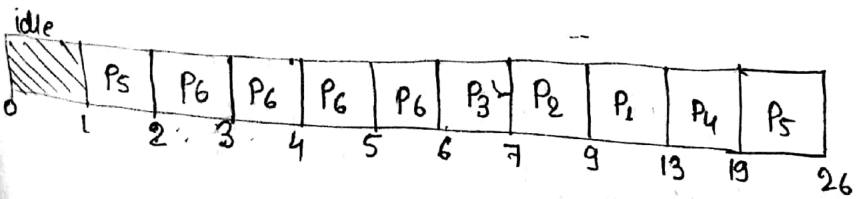
(6, 4, 2, 2) just FCFS
(6, 4, 2, 1)

Avg. W.T = ( )
Avg T.A.T = ( ).



First P_1 has arrive at A.time zero so there is no completion it will enter into execution state but after 1S process P_2 has arrive and Until this P_1 has finish 1s of execution remaining is 6s so now we will compare with P_1 remaining b.time and P_2 burst time. Hence P_2 having lowest Job that P_1 will be remove from the execution state to read queue & CPU allocated to the P_2. Same process we will do whenever any new process arrives.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	3	4	13	10	6
2	4	2	9	5	3
3	5	1	7	2	1
4	2	6	19	17	11
5	1	8	26	25	18
6	2	4	6	4	0



Average waiting time = $\frac{6+3+1+11+18+0}{6} = \frac{39}{6} = 6.3$

Average turn around time = $\frac{10+5+2+17+25+4}{6} = \frac{63}{6}$.

Advantage:

→ Best approach to minimize waiting time.

→ Easy to implement in batch system where required CPU time is known in advance.

→ The throughput is increased because more processes can be executed in less amount.

→ it is implementable with predicted/estimated burst time.

Dis-advantage:

* Impossible to implement in interactive system where required CPU time is not known.

Impossible to predict the amount of CPU time a job has left.

Longer processes will have more waiting time, eventually they will suffer starvation.

*Priority Scheduling:

- Priority Scheduling is a non-preemptive algorithm and one of the most common Scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.

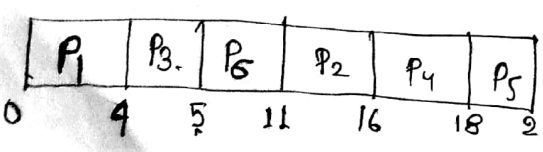
Process with same priority are executed on FCFS.

Priority can be decided based on m/m req., time req. or any other resource req.

*NON-PREEMPTIVE

P.NO	A.T	B.T	Priority		C.T	T.A.T	W.T
1	0	4	4		4	4	0
2	1	5	5		16	15	10
3	2	1	7	→ high priority	5	3	2
4	3	2	2		18	15	13
5	4	3	1	→ low priority	21	17	14
6	5	6	6		11	6	0

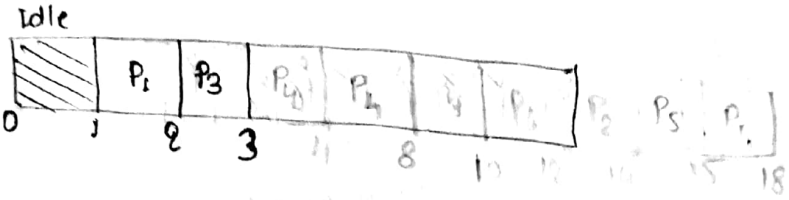
C.T - AT ↓ T.A.T - B.T ↓



Avg. WT: $\frac{0+10+2+13+14+0}{6} = \frac{39}{6} = 6.5$

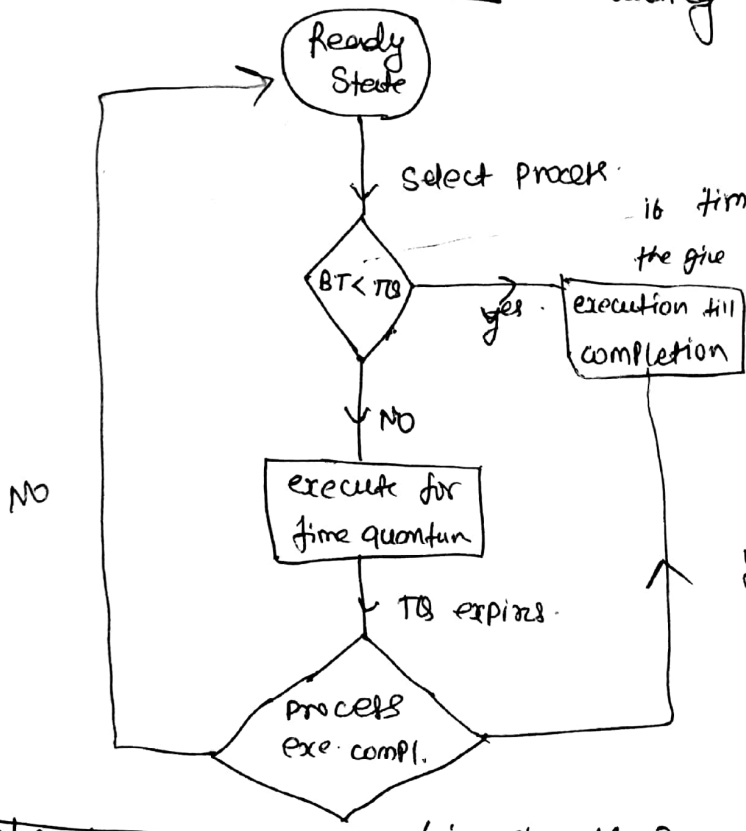
P.N	A.T	B.T	Priority	C.T	T.A.T	W.T
1	1	4	2 - low	18	17	13
2	2	2	5	14	12	10
3	2	3	7	10	8	5
4	3	5	8 - High	8	5	0
5	3	1	5	15	12	11
6	4	2	6	12	8	6

Avg. w.T : $\frac{13+10+5+11+6}{6} = \frac{45}{6}$



Round Robin: (can be implement using queue).

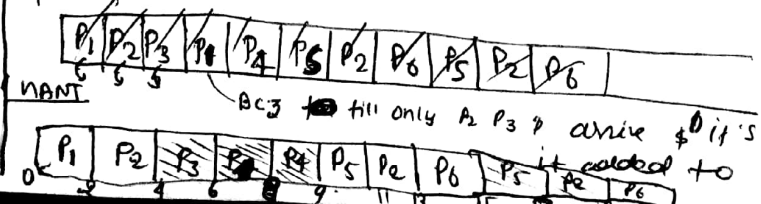
- It is most popular practically used scheduling algo. bcz it doesn't depends upon the burst time.
- Every process will get same amount of time which is called as time quantum. therefor no process wait forever.
- it is the preemptive process scheduling algo.



if time required is less than time quanta the give only required time else give the time quantum.

PNO	AT	BT
1	0	4
2	1	5
3	2	2
4	3	1
5	4	6
6	6	3

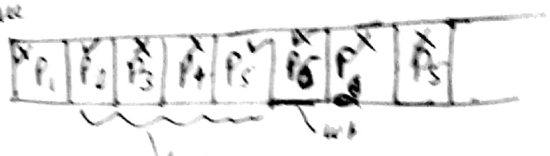
TQ = 2 (it pick the process in FCFS manner. only diff. is that it preempt the process after time quantum)



- If time quantum is less then it leads to wait for long time while less time quantum leads context switching.

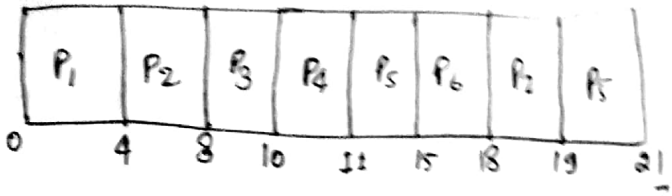
9. TQ = 4.

PNO	AT	BT	CT	TAT (CT-AT)	WT
1	0	4	4	4	0
2	1	5 → 1	19	18	13
3	2	2	10	8	6
4	3	1	11	8	7
5	4	6 → 2	21	17	11
6	6	3	18	13	10



wait while P1 completing it's work. after to queue in FCFS manner.

GANT CHART



Here Gant chart is smaller than other one so that here context switching decreases.

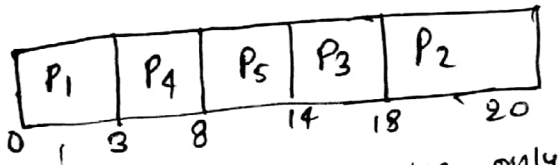
But here process are waiting for a long time.

Longest Job First Scheduling algorithm:

- Just like SJF and it is also having Preemptive & non Preemptive ~~strategy~~ strategy. But here we have to pick the process which is having longest Job.

use non Preemptive rules.

PNO	AT	BT	C.T	T.A.T	W.T
1	0	3	3	3	0
2	1	2	20	19	17
3	2	4	18	16	12
4	3	5	8	5	0
5	4	6	14	10	4



at zero we are having only one Proces

Memory Management

- m/m management is the act of allocating, removing, and protecting Computer memory for multiple processes.
- It refers to management of Primary m/m or main m/m which ~~coordinate~~ data to and from RAM and determine necessity for virtual m/m.
- The main fun of m/m management is to:

→ Allocate m/m to processes when needed. and vice versa.
→ Keep track of what m/m is used and what is free.
→ Protect once process's m/m from another.
→ Takes decision which process will get m/m and when and for how much.
→ Update the Status of m/m locn when it is allocated or leaded.

Goal of m/m management:

→ Effective utilization of memory space: Fragmentation occur leads to the m/m loss. So keeping fragmentation at each level, so there is a chance of effective Utilization.
→ Run larger program in Smaller memory. using virtual m/m.

Terminology:

Dynamic loading:

- Complete program is loaded into m/m, but Sometimes a certain Part of the Prog. is loaded only when it is called by. Prog.

i.e Some Part of the program is loaded at run time.

Dynamic linking:

- There is one program dependent on the other program then instead of loading ~~both~~ Simultaneously. CPU links the dependent program to main executing prog. when it needed.

Logical address:

- Address generate by the CPU. it is also called as virtual add.
- it is an address relative to start of a process address space.
- Set of a logical add. generated by program is called as a logical add. space.

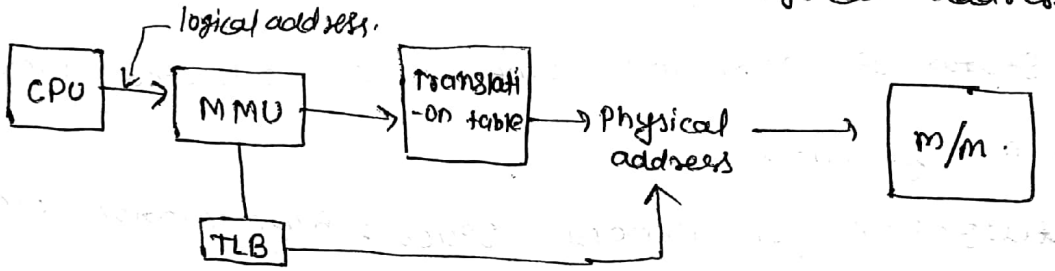
- **Physical address**: (real address / binary address)
  - Address seen by the memory unit.
  - it is the actual physical m/m address that is used to access a specific storage cell in main m/m.

* Load time and Compile time m/m address binding (LAS) and (PAS) are same. But they different in execution time binding.

***Memory management unit (MMU)**:

- Hardware device that maps virtual to physical address.
- Each reference is passed through the MMU.

Translate the virtual address to a physical address.



* **Contiguous memory allocation**: Centeralized.

- Each process allocated a single contiguous chunk of m/m. [block
- it is implemented with the help of Partitoning method.
- Access is so fast. (Just like array).
- it will always suffer from external fragmentation.

* **Non-contiguous m/m allocation**: Decentralized.

- Part of process can be allocated non contiguous block of m/m
- it is implemented with the help of paging, segmentation and virtual m/m.
- Data can't access directly. Slow access. (like linked list)
- it won't be suffer from fragmentation.

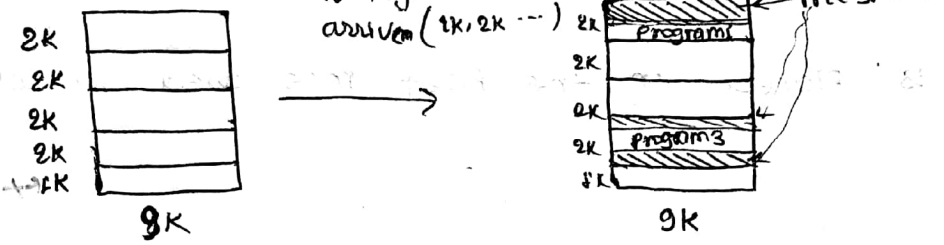
Contiguous m/m allocation
- Fixed Size Partitioning
- Variable Size Partitioning

* Fixed length Partitioning :

→ memory broken up into fixed size partitions. But the size of two partitions may be different.

• Each partition can have exactly one process.

eg



• when the process arrives, allocate it a free partition.

Adv.

Easy to implement, fast context switch.

Dis adv.

• degree of multiprocess is limited.

• Internal fragementation : m/m in a partition not use by a process is not available to other process. (amount of m/m can wasted after the process has allocated.

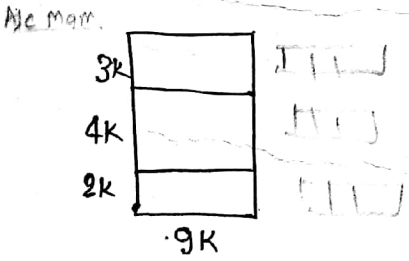
• Partition size : one size dosn't fit all (very large process).

• External fragementation also there. (by sum up all the int. fg.)

* Variable length Partitioning :

• Physical m/m broken up into variable size partitions.

• when the process is arrives, it is allocated m/m from a hole large enough to accommodate it.



: Here Partition control manager is used to create the need of in coming process, it breaks down a large block of free m/m space in order to accomodate a smaller process and also merge block the m/m released after process has terminated into large block.

Adv.

no internal fragmentation : allocate just enough for process.

Dis adv.

- External fragmentation : Job loading and unloading produces empty holes scattered throughout m/m. / unable to accomodate process of size even though we have suff. size of m/m • bcz of contiguity req.

* Partition allocation Policies / Fitting Strategies :

1) **First Fit :**
The partition is placed in the first free area in which it will fit.

2) **Best Fit :**
The smallest possible m/m partition/area in which the partition will fit. / It will perform best in the case of fixed size partition.

3) **Worst Fit :**
The largest free area/partition which is enough to allocate for program.

4) **Next Fit :**
Same as the F.F but it dosn't start from first partition. it start from the place where the last partition ended.

* (a) Var. leg. partition.

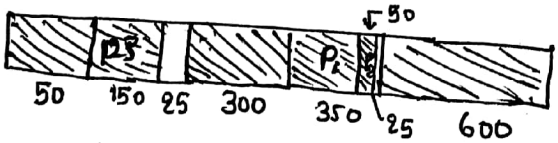
P_1	P_2	P_3	P_4
300	25	125	50

Here process is allocate in FCFS

First

Here start search from base add. and use 1st partition of the 1st block which is capable enough for the process. So here P_1=300 so from starting we got 350 as a sufficient for this. Like wise every time search from base add.

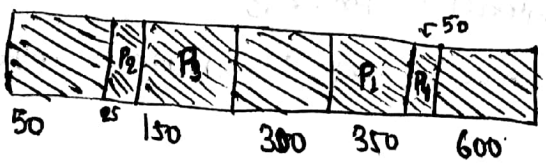
best



It will search all vacant space current present in the m/m and it will allocate smallest block which can hold the process.

So here we can't allocate the process P_4 bcz there is no any block which is capable to hold it. So this is also a eg. of external fragementation.

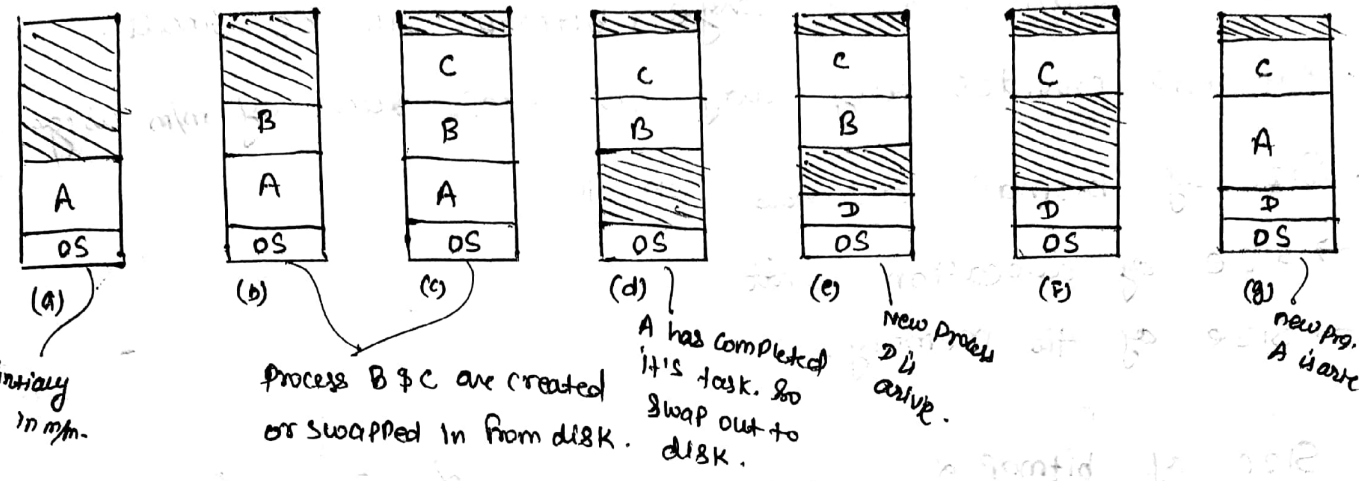
Worst



It will search the entire m/m and it will allocate the largest block first so that here the less chance of external fragmentation.

Swapping:

- it brings a process in it's entirety, running it for a while then put it back on the disk.
- it is a mechanism by which process can be swapped temporarily out of main memory to secondary memory and make that m/m available to other processes.



- When swapping creates multiple holes in m/m, it is possible to combine them all into one big one by moving all the processes downward as far as possible.
- Here mmu will manage information about which are the occupied area and which are the vacant area.

How the mmu will manage the info. about which are the occupied area and which are the vacant area.

Managing free m/m:

When m/m is assigned dynamically, the operating system must manage it.

- There are two ways to keep track of m/m usage.

1) m/m management with bit map:

- with the bit map m/m is divided into allocation units as small as a few words & as large as several kilobytes.

• Corresponding to each allocation unit, there is a bit in the bitmap.

Bit will be 0 if the unit is free (vacant)

Bit will be 1 if the unit is not free (occupied).

• The size of allocation unit is imp design issue.

• Smaller the allocation unit, larger the bitmap.

• if allocation unit is large, bitmap will be small.

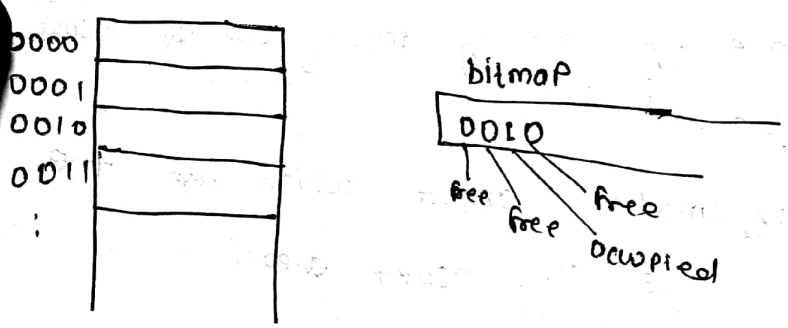
* Bit_map provides easy way to keep track of m/m usage.

Size of bitmap depends on the:

→ size of allocation unit

→ Size of the memory.

$$\text{Size of bitmap} \propto \frac{1}{\text{Size of allocation unit}} \propto \frac{1}{\text{Size of m/m}}$$



② m/m management with LinkedList:

• Another way to keep track of m/m usage.

• Here we maintain linkedlist corresponding to a process /hole P(P/H)

• Each entry in the list specifies 4 field.

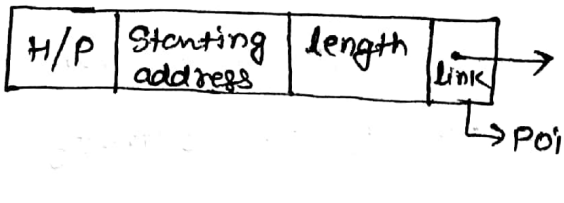
1) Hole (H) or Process (P) indicator.

2) address at which it starts.

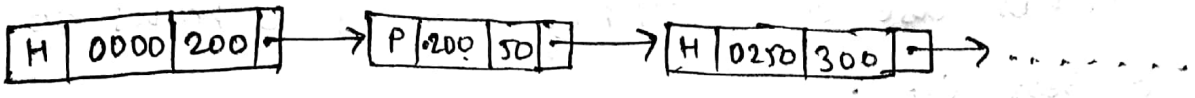
3) length of the process / Size.

4) Pointer to the next entry.

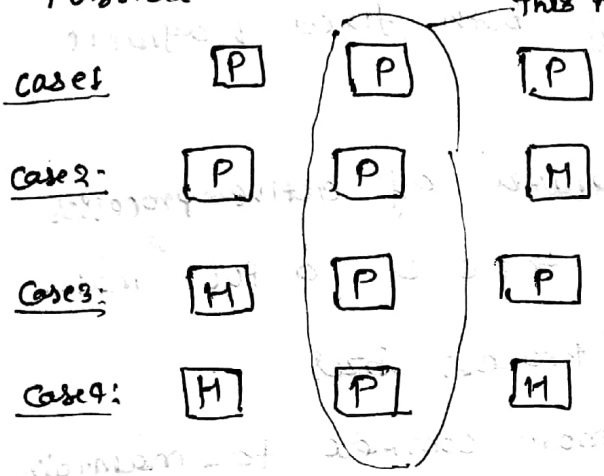
node



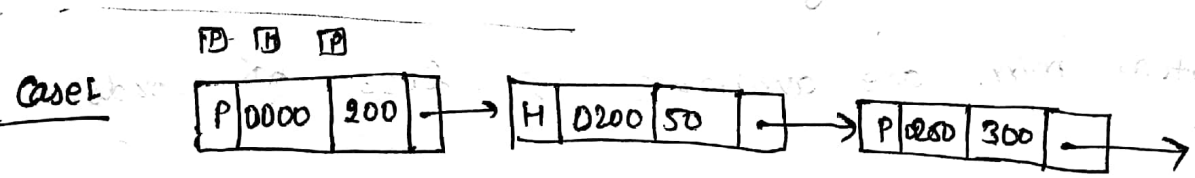
eg.

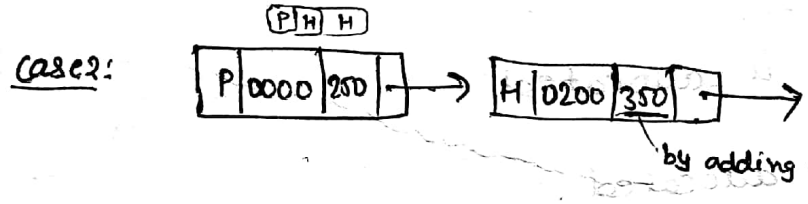


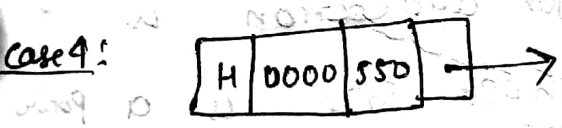
Possible.



possible node updation:







Buddy System:

- Buddy m/m allocation technique is a m/m allocation algo. that divides m/m into partitions to try to satisfy a m/m request as suitably as possible.
- This system makes use of splitting m/m into halves to try to give a best fit.
- it is used to fixed the drawback of both fixed & dynamic partitions schemes.
- fixed partitioning scheme limits the number of active processes and may use space inefficiently if there is a poor match b/w available partition sizes and process sizes.
- A dynamic partitioning scheme is more complex to maintain and include the overhead of compaction.

* In buddy system m/m are available of size 2^K words. $L \leq K \leq U$.

2^L = Smallest size block that is allocated.

2^U = large size block that is allocated.

- Generally 2U is the size of entire m/m available for allocation.

* Here the entire m/m space available for allocation is initially treated as single block, whose size is a power of 2.

- When the first request is made, if it's size is greater than half of the initial block then the entire block is allocated.
- otherwise the block is split in two equal companion buddies.

→ if the size of the req. is greater than half of one of the buddies, then allocate one to it. otherwise one of the buddies is split in half again. This method continues until smallest block greater than or equal to the size of the req. is

found and allocate it.

- When process terminates the buddy block that was allocated to it is freed.

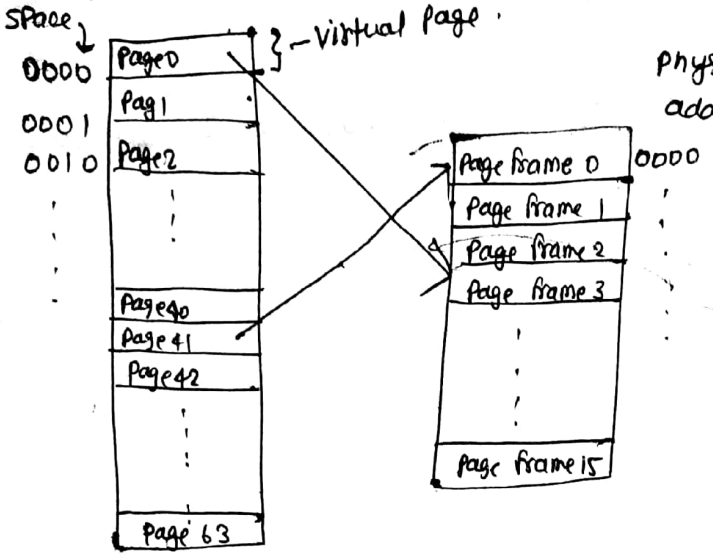
Virtual memory

(Part of sec m/m use to store the pages in such a way that no. of page = no. of frame (small portion of m/m).

- Virtual m/m is a seperation of user logical m/m from physical m/m. (main m/m.)
- Virtual m/m enables to run long program in limited space.
- Only pent of prg. needs to be in memory for execution.
- virtual m/m can also work in multiprogramming s/m.
- logical add. space > physical address space.
- The idea behind virtual m/m is that each program has its own address space, which is broken up into chunks called "Pages". (process divided into no. of page & m/m into no. of frames) & it virtual m/m can be implemented as ← Paging, Segmentation (divide in such a way that both are same (page & frame))

Paging: Non contiguous m/m allocation. in physical m/m. allocation. (paging used to avoid external frag.)

- Secondary m/m divided into numbers of "pages".
- main m/m divided into numbers of "frames"
- Address translation occur when page is taken from V.M to M.M.
- Program that are large in size is divided into no. of pages by CPU.
- CPU generates the virtual address & sent to mmu.



A/c to page table we will denote it by array.

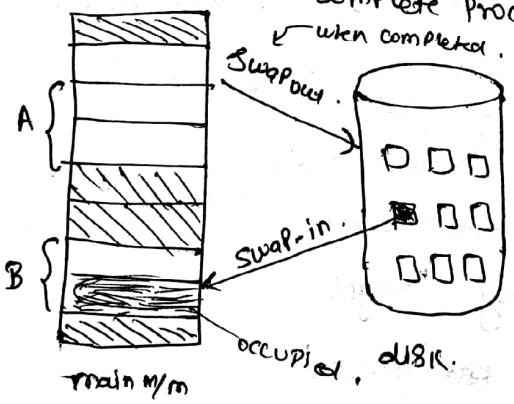
- Virtual address space is divided into fixed size unit called pages.
- Corresponding unit in physical address is called page frame.

Advantage of VM:

- only part of the prog. needs to be in m/m for execution.
- logical address space is much longer than physical add. space.
- it need to allow pages to be swapped in and out.
- less I/O required leads to faster & easy swapping of processes.
- * we need to maintain entire image of process in disk storage.

Demand Paging:

- in this technique a page is brought into the m/m for it's execution only when it demanded.
- it is a combination of paging and swapping.
- it requires that complete process should be in disk in the form of pages].



it is called as lazy swapper bcz it swaps to page only when needed.

Adv.

it reduces m/m requirement.

Swap time is also reduce (bcz only pages are swap-in or swap-out).

* it increases the degree of multiprogramming.

disadv.

. Page fault may occur. (It is used in the Performance measurement

Effective m/m access time (Ea) : [P x Page fault time + (1-P) * ma]

P → Prob. of Page fault.

(1-P) → Prob. of Page not fault

ma → general m/m access time.

Page table : (every Process has Page table.)

. Data Structure used to map Page no. into Page frame no.

is called PMT (map.) or PT.

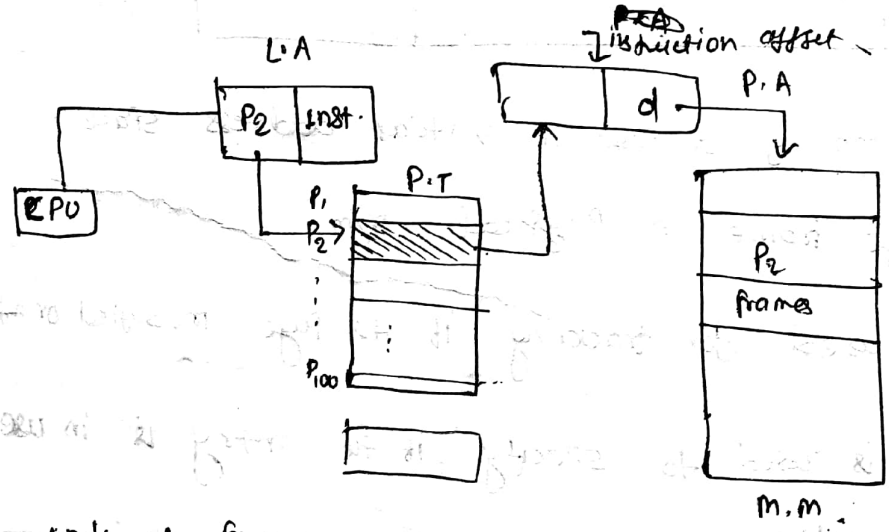
Page no. i.e V.A.

Page frame no. i.e P.A;

. The no. of slot in the P.T is equal to the no. of pages present in secondary m/m (V.m).

. it will contains the base address of the frames corresponding to each page. So here translation is not required.

. So here no. external fragementation.



. internal fragementation is there.

. Accessing PMT should be fast to increase execution sp.

. PMT should be large.

eg.

64 kb V.m

32 kb m/m

. Size of page → 4KB.

No. of page in SM = 16 Pages. & Page frame = 8 Page fram

DAT Sec in.

TLB (Translation lookaside Buffer) or associative m/m:

- It is a hardware device inside the MMU which is used for mapping virtual address space to physical m/m. it is used for increasing the speed of paging.
- it is used to the time taken to access a user m/m loc.
- TLB can't hold all the pages, but it holds only imp. pages & their frame no.
- mapping virtual address to physical address without going through PMT is done.

Page no.	Page frame no.	modified bit (Y/N)	Present/absent	Protection

- Page no: it is the page no. of a page in virtual address space.
- Frame no: it is the page frame in physical m/m.
- modified bits: it is used for tracking if the page modified or not.
- Present/absent (valid): it is used to specify if the entry is in use or not.
- Protection bits: it is used for controlling read, write & execute permissions.

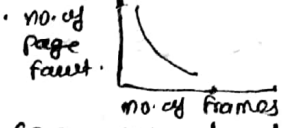
for

- When virtual address is given to MMU, the H/W first checks if that page no. is present in TLB or not by comparing to all entries.

- if valid match found, then page frame no. is taken directly from TLB without going to PMT or PT.
- if the page no. is not found in TLB mmu defec "miss" & then does page table lookup.
- Then it removes 1 entry from TLB & replace it with PMT entry just looked up.
- Suppose if page is not present in main m/m. then thi is called "hard miss". & here disk access is required.

Inverted page table:

Page replacement algorithms :



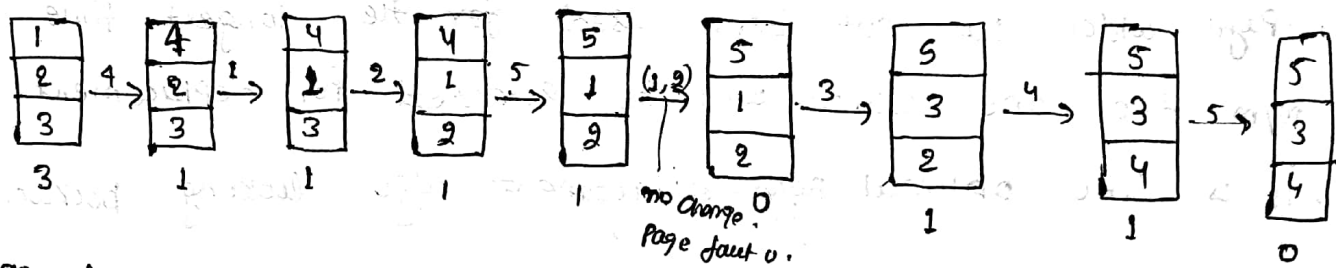
- it is technique by which we can used to decide which Page needed to be replaced when new Page comes in.
- Whenever new Page is referred and not present in m/m page fault occurs and OS replaces one of the existing pages with newly needed page.
- Different Page replacement algorithms suggest different ways to decide which Page to be replace.
- The aim of all algorithms is to reduce the no. of Page fault.

(1) FIFO (first in first out) :

- This is the simplest Page replacement algo. in this, OS keep track of all pages in the m/m in a queue. oldest Page is in the front of queue. when a page needs to be replace page in the front of queue is selected for removal.
- Performance is not always good.

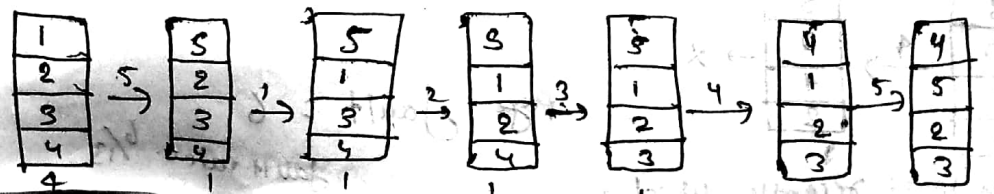
eg. Calculate the no. of page fault for following reference string
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Case1: 3 frame :



Page fault = 3+1+1+1+1+1 = 9. A/c to graph. if frame ↑ es. Page fault ↓

Case2: 4 frame.



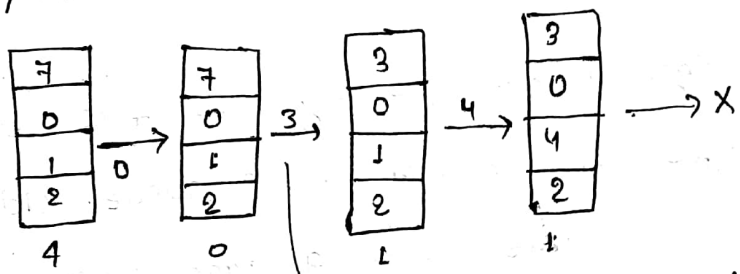
Page fault = 10
This is called Bleday is

Belady's anamoly : for some Page replacement algo, page fault may ↑es as the no. of allocated frame ↑es.

② optimal Page replacement : (look in future).

- In this algorithm, pages are replaced which are not used for the longest duration of time in the future.
- it is the one of the best Page replacement algo. bcz it cause lowest Page fault.
- it is very difficult to implement.

eg. · Reference string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, and 4 Page Slot/frames.



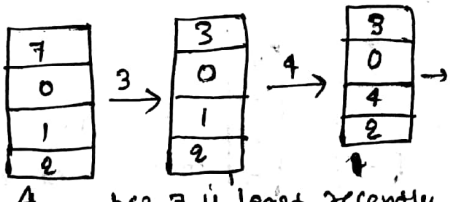
it will replace the 7 bcz it is not used for the longest duration of time in the future.

total page fault = 4 + 1 + 1 = 6.

③ LRU (least recently used) : (look in Past)

- Here we will replaced which is least recently used.
- Page which has not been used for the longest time in a m/m the one which will be selected for replacement.
- It is like optimal page-replacement algo. looking backward in th

eg. 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 frame size = 4



Page fault = 6

fault rate = 6/13

④ NRU (Not recently used).

most computes have 2 bits associated in PMT → reference bit, modifie bit.

Reference bit: set by mmu when page read/write. (R=1)

R=0 (not referenced or not in m/m)

modified bit: Set when page in m/m is monidified else 0.
or set when page is written.

- NRU use the reference bit and modified (dirty) bit.
- initially, all pages have R_bit=0 & M_bit = 0
- Periodically clear the reference bit when page fault occur.
- Page are divided into 4 classes.

	R	m	
class 0	0	0	not referenced, not modified.
Class 1	0	1	not referenced, modified
Class 2	1	0	referenced, not modified
class 3	1	1	referenced, modified.

logic

* Page that is recently used many times will be needed in future also so it will not be removed.
* Page that will be remaved will be the page that has low order of class. i.e which is not referenced till now.
- Page that has been referenced is assume that it will be need in the future also.
- Class 1 page dosn't occur mainly bc2 a page can't be modified without referencing.
- Page can be modified if it is refurenced to main m/m.

⑤ Second chance page replacement :

A simple modification to FIFO that avoid the prob. of heavily used page.

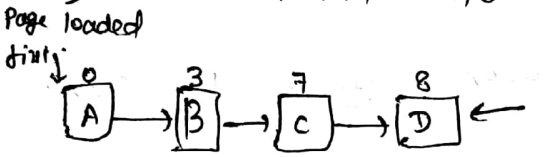
- It inspect the R bit

→ if R = 0 then page is both old & unused so replace it immediately.

→ IF R = 1 then the bit is cleared, the page put onto the end of the list of pages and it's load time is update as though it had just arrived in m/m.

i.e 1) put at the end.

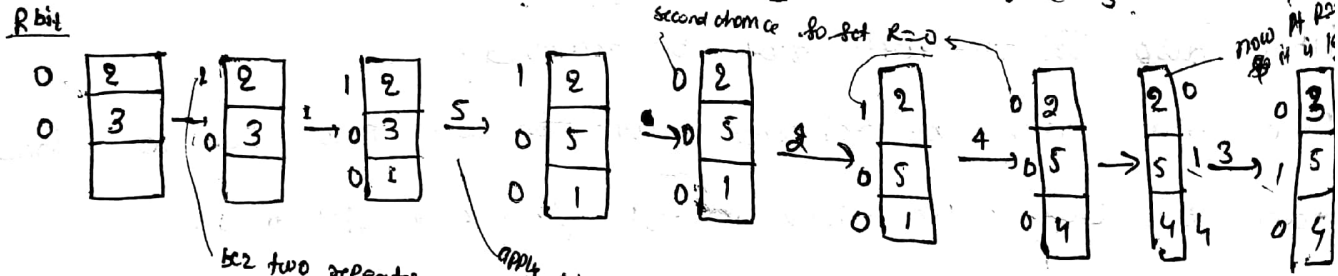
2) update pointer to next node.

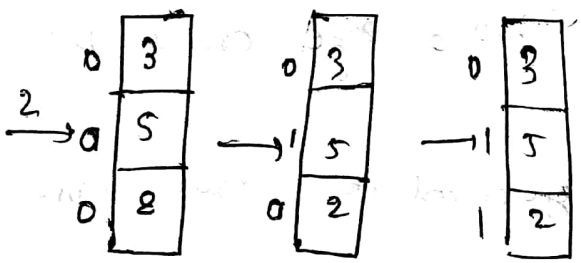


eg

⑥ Clock page replacement :

→ 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 frame size 3.

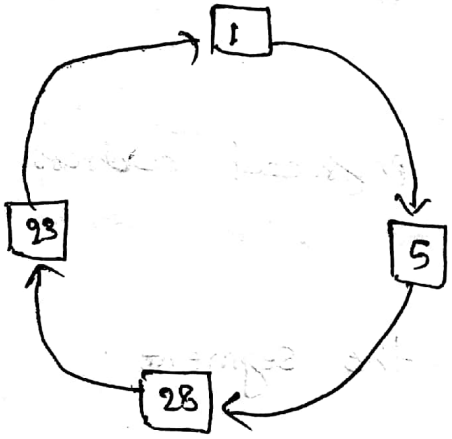




Page fault = 7.

⑥. Same as Second chance Page replacement different is here we have circular list instead of linear list.

- Second chance was inefficient as it is moving pages continously around the list.
- It keep all the page frames on a circular list in the form of a clock. A head pointer to the oldest page.



- If R=1 then just update the pointer.
- if R=0 then that page is selected for remove.

→ page evicted.
→ new page is inserted into the clock in it's place.
→ pointer update.

- Process is repeated until we found page with R=0.

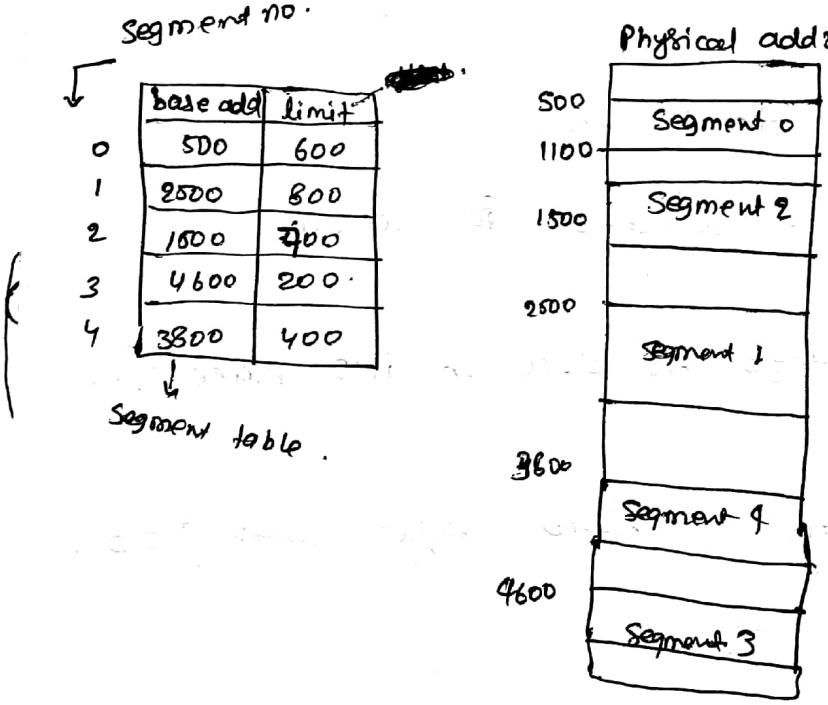
⑤ Segmentation:

A- A m/m management technique in which m/m is divided into variable sized chunk which can be allocated to process.

- Each chunk is called a segment.
- A table stores the information about all such segment and is called segment table.
- Segment table maps two dimensional logical address into one dimensional Physical address.
- Each table entry has.

Base address: it contains starting physical address where the segment reside in m/m.

limit: it specifies the length of the segment.



Adv. - no internal fragementation. (cause in paging bez size of frame was fixed

- Segment table consume less space in comparision to PT in table

disad

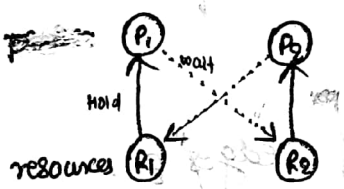
: A process are loaded & removed from the m/m the free m/m space is broken into little pieces.

- causing external fragementation

Deadlock

* A Set of processes is said to be in deadlock if each process in the set is waiting for a resource which is already allocated to another process in the same set

* Two or more processes are said to be in deadlock if they wait for the happing of an event, which would never be happen.

eg 

if P_1 requires R_1 & R_2 and also

P_2 " " "

- Here P_1 won't leave the resource R_1 until get the resource R_2 & P_2 won't leave the resource R_2 until get the resource R_1. So this situation is called deadlock.
- Every deadlock lead an starvation but vice-versa is not true. bcz
- Mandatory Conditions for existance of deadlock:

Starvation (long waiting)
deadlock (infinite ")

① Mutual exclusion:

- only one process may use a resource at time. if another process requests that resource, the requesting process must be wait until the resource has been ~~delay~~ released.

② Hold and wait: Every hold & wait should n't be deadlock.

- A process must be holding at least one resource and waiting to acquire additional resources that currently being held by other process.

③ No preemption: → System should be non preemptive.

A resource can be released only voluntairily by the process holding it. (resource can't taken away by force).

④ Circular wait: A set of processes are waiting for each other in circular form.

eg. If set of process is $\{P_0, P_1, P_2 \ldots P_n\}$

and if P_0 is waiting for the resource that is allocated to P_1

P_1 " " " " " " held to P_2

P_{n-1} is waiting for the resource that is held to P_n.

Resource allocation graph:

- it contains set of vertices V and set of edges E.

$G = (V, E)$

- assign edge
- Request req.
- Process vertice
- Resource
  - multiple instance
  - one instance

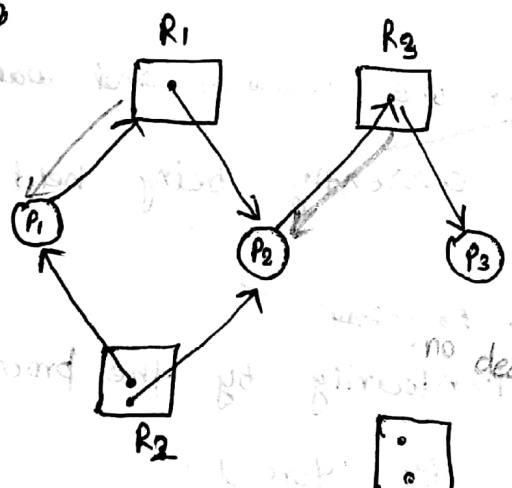
process. resource

one instance resource. Four instance resource.

process req. for resource.

Resource is allocated to process.

eg

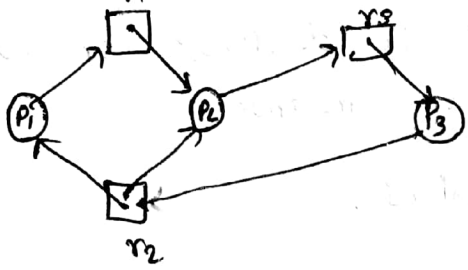


- After completion of P_3 R_3 resource will free.
- After completion of P_2 resource R_1, R_2, R_3 will be free

no deadlock.

one instance

(d)



So from cycle in the resource allocation graph we can't decide the deadlock occur

. Deadlock Detection:

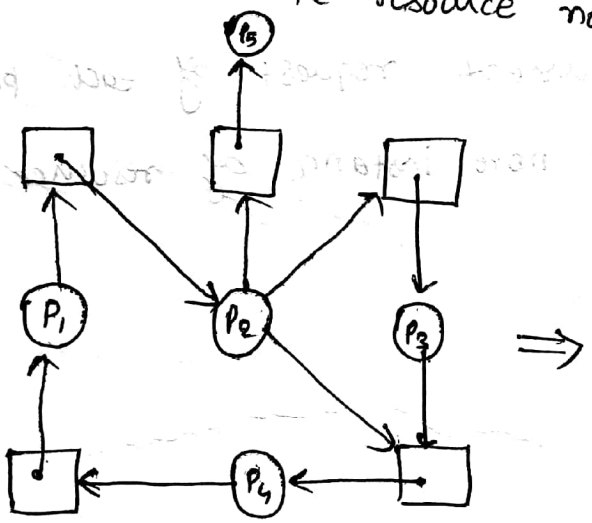
- A algorithm that examines ~~[illegible]~~ the state of the system to determine whether a deadlock has occured.

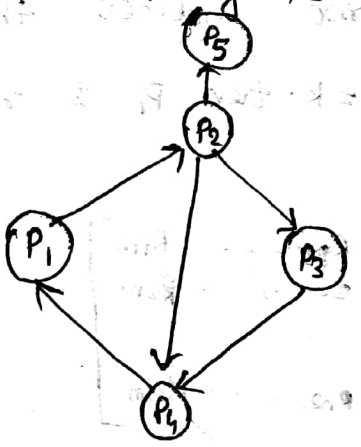
1) Single resource category:

- IF all the resources have only a single instance, then deadlock detection algorithm uses a variant of the resource allocation graph called as " wait-for-graph".
- ~~wen~~ :- write the graph traversal.

RAG → WFG : Remove the resource nodes & collapsing the appropriate edge.

eg





RAG.

⇒ P1 P2 P3 P4 P1

⇓

cycle so deadlock.

2) Multiple category:

- As we see previous in the RAG if the resource is multiple instance and there is cycle in the RAG. then also the system wasn't in the deadlock situation. So from cycle in the

we can't decide the system is in the deadlock or not.

· Matrix solution: Here we use three matrix.

1) Available / Existing resource matrix: (E)

it gives total no. of instance of each resource in existance

i.e $[E_1, E_2 \cdots E_m]$ for m- different resources

2) Current allocation matrix: (C)

A nxm matrix defines the no. of resources of each type currently allocated to each process.

$$C_{ij} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & & & \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{bmatrix}$$

Request matrix: (R)

A nxm matrix indices the current request of each process. If $R[ij] = k$ then P_i is req. k more instance of resource type R_j.

$$R_{ij} \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ R_{12} & 22 & \cdots & R_{2m} \\ \vdots & & & \vdots \\ R_{n1} & R_{n2} & \cdots & R_{nm} \end{bmatrix}$$

* We need one more matrix that is available matrix A which keep track of the no. of resources assigned to the processes. a[i] will gives no. of instance of resource i that are currently available.

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

it we add up no. of currently allocated resources + no. of available resource that is equal to no. existing

- if there is no process for which the corresponding row of request matrix R is larger or equal to availability matrix or there is no unmarked process. then we can say that system is not in the deadlock situation.

eg $E = [4\ 2\ 3\ 1]$ $C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$ $R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$

find wether the given system is deadlock or not.

With the given E & C Availability matrix $A = [2\ 1\ 0\ 0]$

Step1: check each row of req. matrix R. Here row 3 is less than equal to A. So update $A = [2\ 2\ 2\ 0]$ by adding the corresponding row of C. P_3 is marked for execution.

Step2: Checking each row of R with availability matrix A $R_2 \leq A$ so update the A by adding corresponding row of C.

$A = [4, 2, 2, 1]$ Now. P_2 is marked for execution.

Step3: $R_1 \leq A$ so update the A by $A_i + C_{ij}$ so A becomes.

$A = [4, 2, 3, 1]$

Here no. unmarked process is remaining so there is no deadlock.

* Algorithm terminates as all the process are marked for execution or.

Algo

1. look for an unmarked process P_i, for which the i^{th} row

Of R is less than or equal to A.

2. If such a process is found, add the ith row of C to A mark the process and go to step1.

3. If no such process exists, the algorithm terminates.

Deadlock Recovery:

- Once deadlock has been detected, some strategy is need for recovery.

① Deadlock through preemption:

- the ability to take resource away from a process, have another process use it, and given it back without the pro noticing.
- We successively preempt some resources from process and g these resources to other processes until the deadlock cycle u broken.
- it is highly depends on the nature of the resource.
- it is too deficult or sometime impossible to recovers

Deadlock recovery through RollBack:

whenever deadlock is detected, it is easy to see which resources are needed.

- To do the recovery of deadlock a process that owns a need resource is rolled back to a point in time before it aquire some other resource just by starting one of it's earlier checkpoints.

③ Recovery through killing processes: → Kill all (It will break the dl cy but great expense)

→ Kill one at time.

apply deadlock detection and kill the process one by one until there is no any cycle exist.

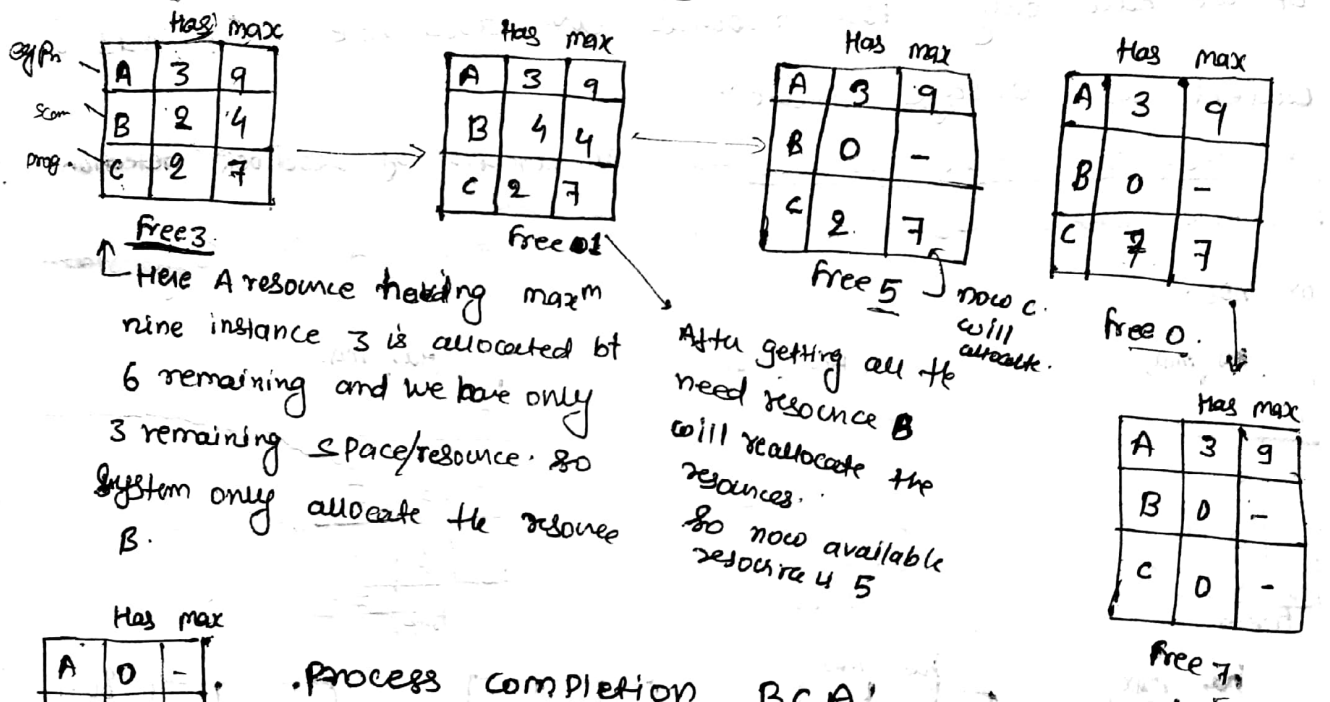
Deadlock avoidence : it analyzes the allocation state to determine whether granting a req. (resource) might lead to deadlock later. [resource allocation, deadBanker's algo.] : it guarantee that allocating resource will not cause any deadlock.

Safe and unsafe State :

- A state is said to be safe if there is some scheduling order in which every process can run to completion even if all of them, suddenly req their maxm no. of resources immediately.

or

- if need of all the processes can be satisfied with the free/available resource in some order or sq. then the system is said to be safe state.



Process completion BCA. Here we find a seq. that is guaranteed to work. So no deadlock & system is in the safe state.

Unsafe State :

- If system has allocated available all resources to all the processes need, then it is in potentially unsafe.

eg. Suppose you go the the bank for transaction, and there is only Rs 200 is remaining. and you want 150 and also another person want 100 Rs. total is 250 but available balance is 200 so at this is the deadlock situation.

to another person after deposite some money by another person.

- If a system in unsafe state it may a may not be in deadlock. but vice versa is true.
- deadlock is a subset of unsafe state.

eg.

	Has	Max
A	3	9
B	2	4
C	2	7

Free 3

A	4	9
B	2	4
C	2	7

Free 2

If suppose 1 resource is allocated to A.

A	4	9
B	4	4
C	2	7

Free 0.

A	4	9
B	-	-
C	2	7

Free 4 bt is A need 5 more resource for comp. and C also need 5.

bt we have only four resources are available. So this is called as unsafe state.

* Banker's algo for Single resources :- it is an extension of deadlock detection safety algo.

eg: Check weather the given resource allocation is in safe state or not.

	Has	max
A	0	6
B	0	5
C	0	4
D	0	7

Free 0

	Has	max
A	1	6
B	1	5
C	2	4
D	4	7

Free 2

	Has	max
A	1	6
B	2	5
C	2	4
D	4	7

Free 1

eg)

	Has	max
A	6	6
B	0	5
C	0	4
D	0	7

Free 4 →

	Has	max
A	-	-
B	5	5
C	0	4
D	0	7

Free 5 →

A	-	-
B	-	-
C	4	4
D	0	7

Free 6 →

A	-	-
B	-	-
C	-	-
D	-	-

Free 3

So. System is in Safe.

<ABCD>

b) Also system is in the Safe State. Completion order <CBAD>

9 C is an unsafe state. [Also any process will Complete.]

[No deadlock.

- If the system is in the Safe State then OS can avoid the deadlock
- If System is Unsafe State then there is possibility of deadlock

- Banker's algo. ensure that System will never enter an unsafe state.
- Banker's algorithm is resource allocation and deadlock avoid -ance algo. which test all the req. made by processes for resources, it check for safe state, if after granting req. System remains in the safe state it allow the req. & if there is no safe state it don't allow the request made by the process.

- Banker's Algorithm for multiple resources:

Here we need three matrix
→ current allocation matrix.
→ request matrix ← How much resources each process could req.
→ Existing matrix (E)

Available matrix = E − P (P: Process matrix.)

eg

C:

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0
	5	3	2	2

Resource assigned

R:

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed.

E = [6 3 4 2]

P = [5 3 2 2] A = [1 0 2 0]

1) By checking the matrix A R_4 is less than A. So we make the process D for execution. and after execution D will release the holding resources. So we need to update Availability matrix.

A = [1 0 2 0] + [1 1 0 1] = [2 1 2 1] mark Process D.

2) R_5 is less than So mark the process E for execution in matrix C. and update the value of A by adding

Corresponding code in C.

A = [2 1 2 1] + [0 0 0 0] = [2 1 2 1]

3) R_1 is less than equal to A so unmark R_1 for execution and update the value of matrix A by ...

A = [2 1 2 1] + [3 0 1 1] = [5 1 3 2]

4) $R_3 \leq A$ so A = [5 1 3 2] + [1 1 1 0] = [6, 2, 4, 2] mark R_3

5) $R_2 \leq A$ so A = [6, 2, 4, 2] + [0 1 0 0] = [6 3 4 2] mark

Here There is no unmark process is remaining so System is in the Safe State. We can run in order D A B C E.

(it may be different that doesn't matter)

Deadlock prevention: → it makes deadlock impossible

- It insure that four condn for resource deadlock don't occur simultaneously and system will never enter in deadlock State.

1) Attacking the mutual exclusion: make some of the resource sharable.

- Some devices (such as printer) can be spooled. only the printer daemon uses printer resource.
- Avoid assigning a resource when it is not absolutely needed
- As few processes as possible directly claim the resource.

So it is not possible to prevent deadlock by preventing mutual exclusion as certain resources are can't be shared safely.

2) Hold and wait:

- When a process requires a resource, it doesn't hold any other resource but this may result in a starvation as all req. resources might not be available freely always.

- A resource can get all required resources before it start execution - This will avoid deadlock but will result in reduce throughput as resources are held by processes even when they are not needed. They could have been use by other processes during this time.

- the hold & wait conditions can be eliminated by forcing a processing to release all resources held by it whenever it req. a resource that is not available.

3) Attacking the no preemption condition:

- Preempt resources from process when resources required by other high priority process.
- If a process req. for a resource which is held by another waiting resource, then the resource may be preempted from the other waiting resource process.
- Sequential I/O devices can't be preempted.
- Preemption is possible for certain types of resources such as CPU & main m/m.

4) Circular wait:

- one way to prevent the circular wait condition is by linear ordering of different types of system resources.
- once a process has some resources allocated to it, it can allocate allocate a new resource only if no. of all it's allocated resources is less than no. of assigned to the req. resource

Q A System is having 3 user processes each requiring 2 unit of resource 'R'. The minimum no. of unit of 'R' such that no deadlock will occur.

a) 3 b) 5 c) 4 d) 6

let instances = 2.

	P_1	P_2	P_3	
"2"	1	1		(then deadlock may occur in worst ca
"3"	1	1	1	(if they share 1 resource simultaneous then deadlock may occur
"4"	1	1	1	(we have still one more resource which give to any one and after finish that will allocate one-one unit to ot

① two process. Here will no. deadlock occ

Hence min^m no. of resource required is 4.

- In this types of question find the max^m no. of resource req to Perform deadlock and add 1 to that it gives the answer.

Q If P_1 P_2 & P_3 required 2 3 & 4 instances respectively then find the min^m no. of resource required to avoid deadlock.

P_1	P_2	P_3
2	3	4

max^m req. to form deadlock $\longrightarrow$ 1 + 2 + 3 $\rightarrow$ (6) So 7 would be answer.

Eg P_1 : 81 P_2 : 31 P_3 : 41 min R = ?

$max\ R = 80 + 30 + 40 = 90$ (Cause deadlock)

$min^m\ R = 91$ (not cause deadlock).

P_1	P_2	P_3	- - -	P_n
x_1	x_2	x_3	- - - -	x_n
(x_1-1)	(x_2-1)	(x_3-1)		(x_n-1)

$$\left(\sum_{i=1}^{n} x_i\right) - n + 1$$

min^m req. to avoid deadlock.

max^m req. to form deadlock.

Q if we have total resource is 100 and each process required two instance of resources. So How many max^m process require so that System not leads to deadlock.

If we have 100 process than it cause deadlock bcz every process will hold one resource & waiting for one max. So anything less than 100 will no cause any deadlock.

So Answer 99.

if $R = 100$ $P_i = 3$ (each process required 3 unit).

$3 - 1 = 2$

$\frac{100}{2} = 50$ $n \times 2 = 100$

$n = 50$ (process – cause deadlock)

P_1	P_2	- - - - -	P_{50}
2	2		2

(max resource which cause deadlock).

So Answer 49

- If there is R resource and each process required n unit of instance.

than min no. process req. to avoid deadlock $= \left[\frac{R}{n-1}\right] - 1$

ceil

- if it gives any remainder than don't substract 1.

Q A computer System has 6 tape drives, which n processes competing for them each process needs 3 tape drives. the max value of n for which the System is guaranteed to be deadlock free.

a) 2 b) 3 c) 4 d) 1

$R = 6$ $P_i = 3$ $n = ?$

$n = \frac{6}{2} = 3$ (max no. of Process required to cause deadlock).

no. 2

So any Process less than 3 would be not cause deadlock.

$(3 - 1) = 2$ (max).

$n \times$ (each process req. $- 1$) = (total resource)

if $n = x$ then $(x - 1)$ max — not cause deadlock.

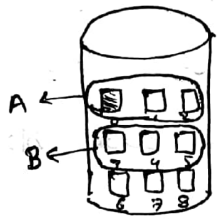
File System :- cont in bk.

Implementation of file: (file allocation): allocate space to the file so that disk space is utilized in an efficient manner.

Contiguous allocation:

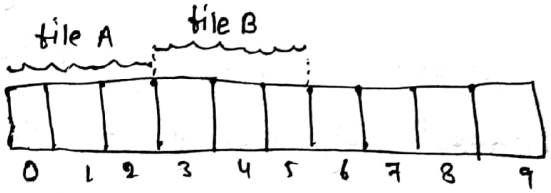
- The simplest allocation scheme is to store each file as a contiguous run of disk blocks.
- Each file occupies a set of contiguous address on disk.
- files are linear ordered.
- location of a file is defined by the disk address of the first block & it's length.
- Both sequential & direct/random access are supported.
  (with the help of index no.)

eg.



directory

file	start	len.
A	0	2
B	3	3



Advantage:

- Simple to implement.
- Read performance is excellent. (bcz entire file can read from a single opr)
- m/m compaction.

disadv.

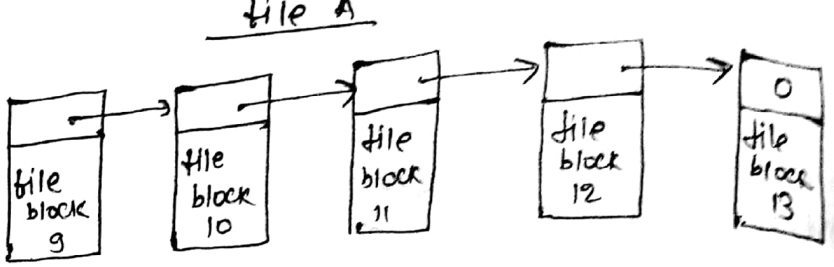
- finding space for new block.
- External fragementation.

Linked list allocation:

- it solves all the problem of contiguous m/m allocation.
- it keep each one as a linked list of disk block.
- The first word of each block is used as a pointer to the next one.

Directory

file	start	end
A	9	13



Adv.

- No external fragementation.
- if we know the 1st block address then next block can be fo
- Sequential access is fast.

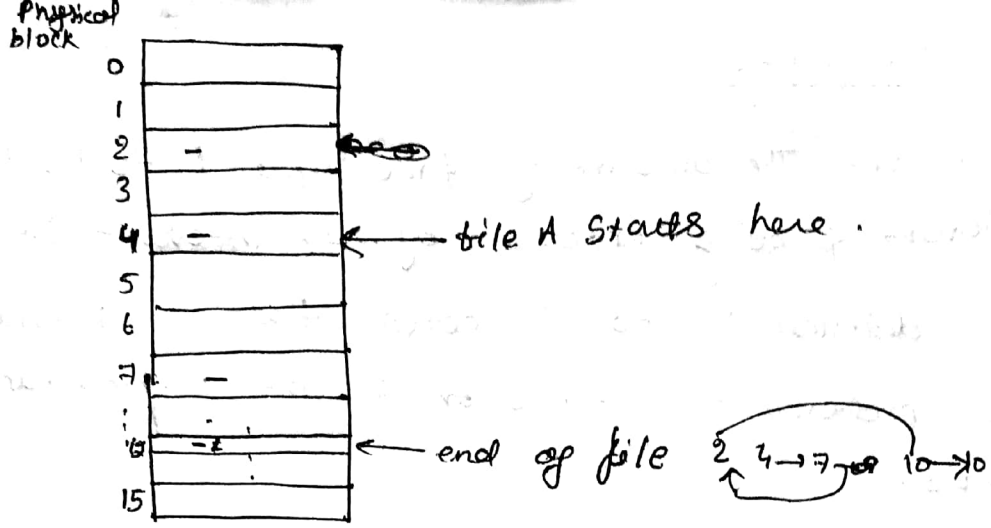
disadv.

- Random access is very slow.
- it takes more space bcz some of block space is reserve for pointer.

- LinkedList allocation using a table in m/m :
- Both disadvantage of linkedlist allocation can be eliminated by taking the pointer word from each disk block and putting it in a table in m/m.
- Here the file block will be terminated with special mark (-1) that is not a valid block no.
- FAT contains all the files which is to be used. (file allocation t
- FAT used by dos is a variation of linked allocation, where all the links are stored in a seperate table at the beginning of the disk.
- The benefit of this approach is that the FAT table can be cached in m/m, greatly improving random access speed

file A uses disk blocks 4, 7, 2, 10, 12 in this order.

- Here we can start with block and follow to chain all the way to the end.



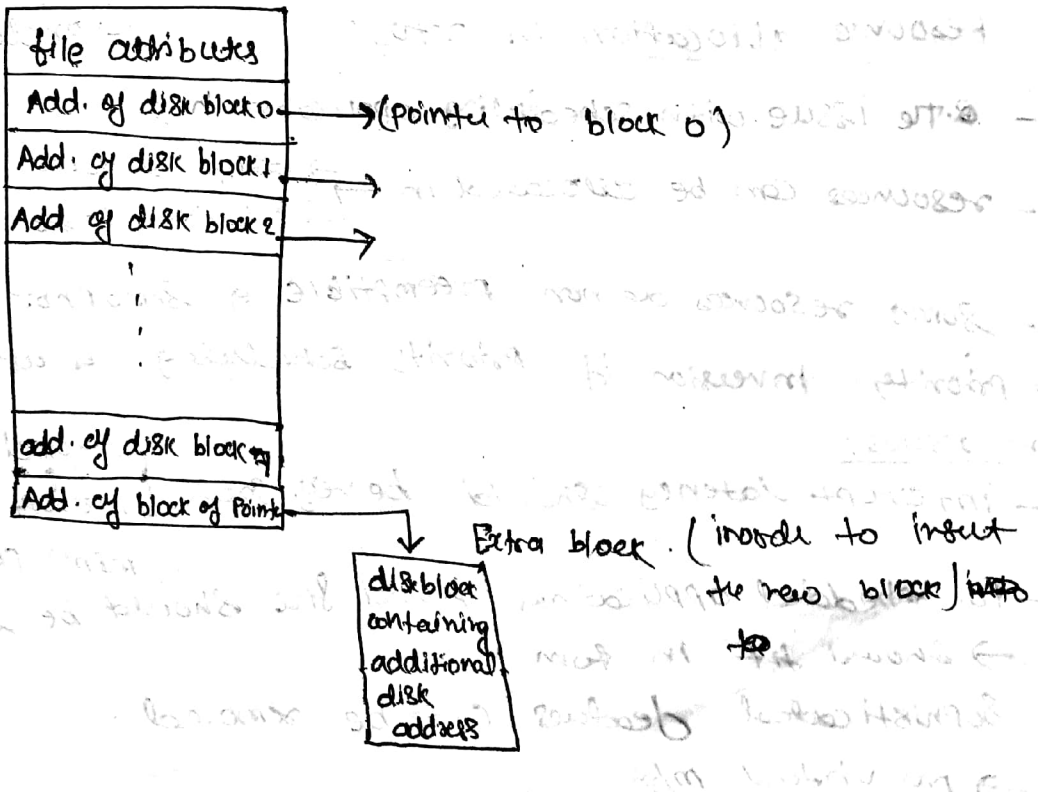
Adv. Random access is much easier.

disadv. Entire table must be in m/m all the time to make it work.

- Space loss.

I node;

- it is used to keep track of which blocks belong to which file. Each file associated with a data structure called i-node (index node)
- use to list the attribute & disk address of the files blocks.



Chara of RTOS :

- **Determinism** : The amount of time required to initially handle the interrupt & being execution of the interrupt service routine.
- OS is determinism to the extent that it performs oprn at fixed, predetermined times or within predetermined time intervals.
- **Responsiveness**: it is concerned with how long, after ack, it takes an operating system to service the interrupt.

Usercontrol : is generally much boadn in rtos.

Reliability : A transient failure in a non os may be solved by simply rebooting the system.

Resource allocation in rtos:

resource → Private
→ Shared
→ exclusive.

- The issue with scheduling applicable here
- resources can be allocated in → weighted round robin
→ Priority based.
- Some resources are non preemptible e.g. Semaphore.
- priority inversion if priority scheduling is used.

other issues:

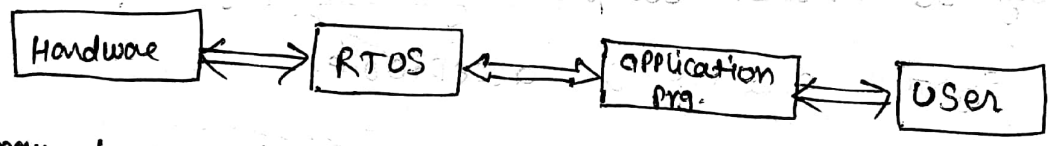
- interrupt latency should be very small → Kernel has to respond to rte
→ interrupt should be disable for minm possible time.
- For embedded applications Kernel size should be small.
→ should fit in rom.
- Sophisticated features can be removed.
→ no virtual m/m
→ no protection.

Real time OS :- fast & quick respondent system.

- The system which are strictly delay time bound are called "real time system".
- The success of a system depends not only on the logical correctness but also on the physical time at which the completion of task has happened.
- Here the time is the imp. factor. for each and every task define a fixed time constraints. processing must be done within the defined constraints.

* In RTOS there is a little swapping of programs b/w primary & secondary m/m. most of the time processes remain in primary m/m in order to provide quick response.

- Therefor in real time system m/m management is less demanding compared to other system.



- it may be use for image processing, audio clip,

eg. production control system, combustion control, control system inside a nuclear reactor, turbine control, missile control system firing of airbags, aeronautical, control system, aircraft control system etc.

functions:

- manage the processor and other system to meet requirements

- It responds to inputs immediately.
- task completed within a specified time delay.
- Synchronize with and respond to the system events.
- move data efficiently among processes and to perform coordination among these processes.

Types of RTOS:

- Hard real time system:
- No delay.
- latency is permissable for the system beyond the deadline.
- degree of tolerance for missed deadlines is negligible.
- A missed deadline can result in catastrophic failure of system
- missile system, aircraft control, firing of air bags. nuclear power plant control, air traffic control
- Soft real time system: - minor delay is acceptable.
- Deadline may be missed occasionally, but system doesn't fail & also, system quality is acceptable.
- it is also known as best effort system.
- most modern OS can serve as the base for a soft real time systems.
- eg multimedia transmission & reception, banking transaction, networking, computer games.

Firm real time

- missing a ~~delay~~ deadline might result in a unacceptable quality reduction
- But many ~~be~~ not lead to failure of the complete system

eg. In control application system, Stock market system.

Parameter for the evaluation of real time system:

- arrival time
- Start time : time at which process becomes ready.
- finishing time
- worst case execution time
- Deadline time : time at which process must finish.

Scheduling in RTOS:

- it is task to set all jobs in order to achieve their deadline.
- The scheduler is the part of the kernel responsible for deci-ding which task should be executing at any particular time
- There is three State:

1) Running (executing on CPU):

2) Ready:

3) Blocked (waiting for an event, I/O)

Flyback time: (critical response time): is time it takes to queue a new ready task & restore the state of the highest priority task to running.

*Algorithms:

- All most algo. are based on priority. i.e Highest priority process runs first.

① Clock driving:

- All Parameter about Jobs known in advance.
- Schedule can be computed offline or at some regular time instance

- min running time overhead.
- Not suitable for many applications.

2. Weighted round robin:

- Job schedule in FIFO manner.
- Time quantum given to Jobs is proportional to it's weight. eg. High speed switching network.
- Not suitable for precedence constrained Job.

* Priority Scheduling: — Static
                        — dynamic

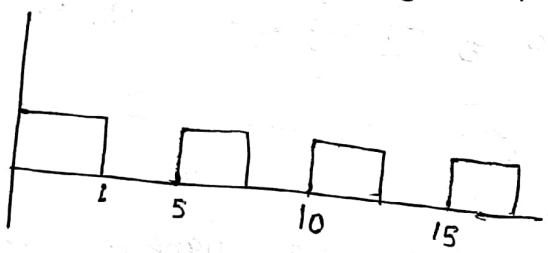
① Static priority:

① Rate monotonic scheduling algorithm:

- The static priorities are assigned a/c to the cycle duration of the Job, so a shorter cycle duration result in a higher Job priority.

These os are generally preemptive and have deterministic guarantees with regard to response time.

- for Periodic task.
- tasks priority inversely proportional to it's Period.



② Deadline-monotonic priority scheduling algo.

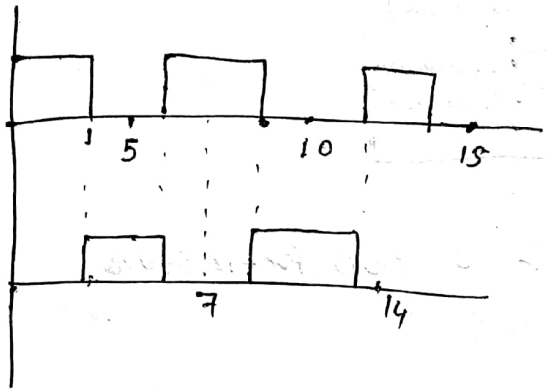
- Here priority is assign to a/c to their deadlines.
- the task with the shortest deadline being assigned the highest priority.

- All tasks have deadlines less than or equals to their min^m inter-arrival times.
- for aperiodic task also.

dynamic:

* Earliest deadline first:

- process with earliest deadline given highest priority.



UNIX as Rtos: (1970, bell lab).

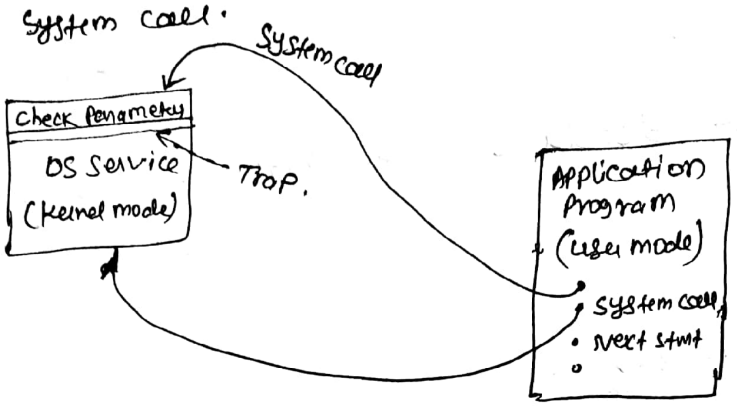
- unix os is a set of program that act as a link btw the computer and the user.
- user communicate with the kernel through a program known as shell. (it is command line interpreter) take common convert to kernel understandable lang.
- unix also have GUI similar to windows.
- it is stable, multi-user, multi-tasking systems for server, desktop & laptop.
- popular flavour of unix: Sun Solaris, GNU/LINUX, macOS X.

* Since unix and it's variants are inexpensive & widely used so after investigation on suitability of unix os for real-time application we found two major shortcomings:

① Non Preemptive Kernel:

- unix kernel can not be preempted. (user program can execute in kernel mode by using a system call.)
- That is why all interrupt are disabled when any OS routine runs. This is done to preserve the integrity of a kernel data structure.

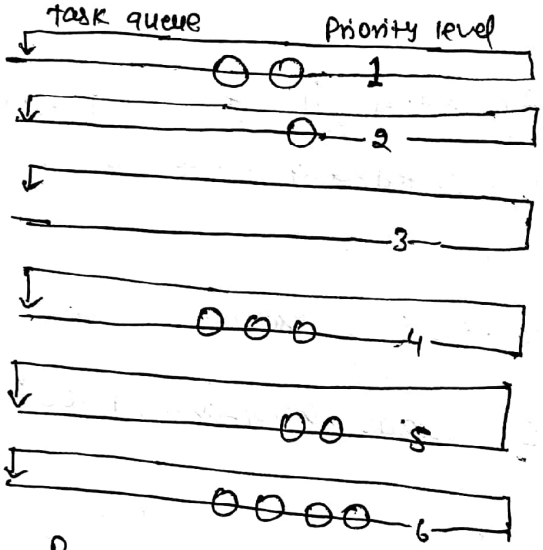
- A non preemptable kernel can cause deadline misses.
- Task Preemption time = time spent in kernel mode + context switching time
- Even user program can run in kernel mode with the help of system call.



- it causes Priority inversion since it is non preemptable.

2) Dynamic priority levels:

- Unix uses round-robin scheduling (1 sec time slice) of task with multilevel feedback queue.



- At every Preemption Point the Scheduler scan the multilevel queue from top (highest priority) and select the task at the head of the first non-empty queue.
- And each task is allowed to run for 1 slice time.
- Unix periodically computes the priority of a task based on the type of task & its execution history.

Interview questions:

- **spin lock:** it is a busy waiting state of the resource, when Process is waiting for the ~~which holds~~ and continuously checking in.
- **deadlock:** when Two resource are depend on each other and waiting for a condition to happen but it will never happen.

* The priority of task T_j is recomputed at the end of it's j^{th} time slice using follows expressions:

$$Pri(T_i, j) = Base(T_i) + CPU(T_i, j) + nice(T_i)$$

- $Pri(T_i, j)$ — Priority of task T_i at the end of it's jth time slice.
- $CPU(T_i, j)$ ÷ Weighted history of CPU utilization of task T_i at the end of it's j^{th} slice.
- $Base(T_i)$: base Priority of the task T_i
- $nice(T_i)$: The nice value associated with T_i (~~set~~ only +ve)
  ↳ set by programmer.

$$CPU(T_i, j) = \frac{U(T_i, j-1)}{2} + \frac{CPU(T_i, j-1)}{2} \quad \text{---(2)}$$

- $U(T_i, j)$: is the utilization of task T_i for it's j^{th} time slice

* eqn. 2 can recursively defined:

$$CPU(T_i, j) = \frac{U(T_i, j-1)}{2} + \frac{CPU(T_i, j-2)}{4} + \cdots\cdots\cdots$$

So

$$Pri(T_i, j) = Base(T_i) + \frac{U(T_i, j-1)}{2} + \frac{U(T_i, j-2)}{4} + \cdots + nice(T_i)$$

- once the task is assigned to a priority level, it is not possible for the task to move out from it's assigned band to other priority ~~level~~ bands due to dynamic priority recomputations.
- nice & utilization percent migrating from assigned band.

- Different base priorities segregate task into the following base band

1) Swapper
2) Block I/O : use DMA file transfer.
3) file manipulation
4) Character I/O : mouse & keyword transfer.
5) Kernel process
6) user processes

- Other deficiencies of unix:
- insufficient device driver support.
- Lack of real time file services:
- inadequate time services support.