22-01-22

Date
Page

Operating System

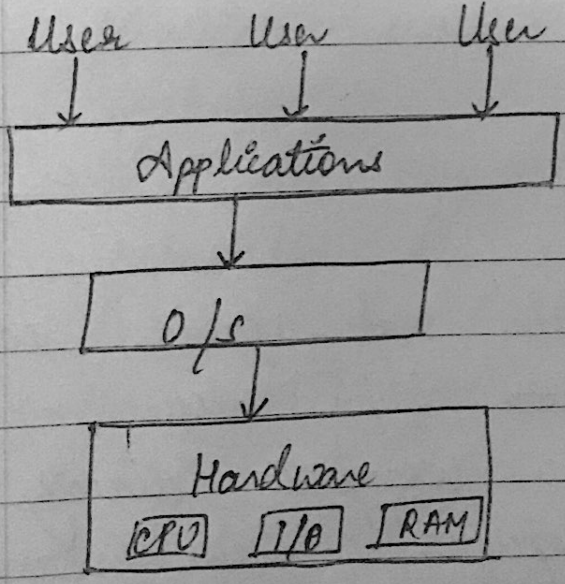
What is an operating system -

→ Operating system provides an interface so that user can interact with computer hardware. It acts as a resource manager as it manages all the resources of computer

E.g. UNIX, LINUX, MAC, MS-DOS

UNIX - Uniplexed information computer system.

LINUX - Lorable intellect not using XP



O/S is a system software which works as an interface between user and computer hardware.

As user cannot directly access with hardware.

∴ We have to write programs for accessing hardware each hardware and these programs are already written in O/S.

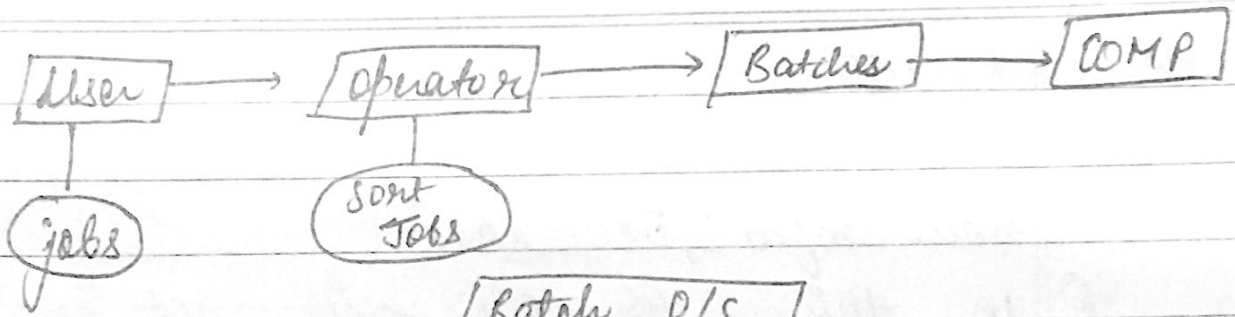
Works of O/S

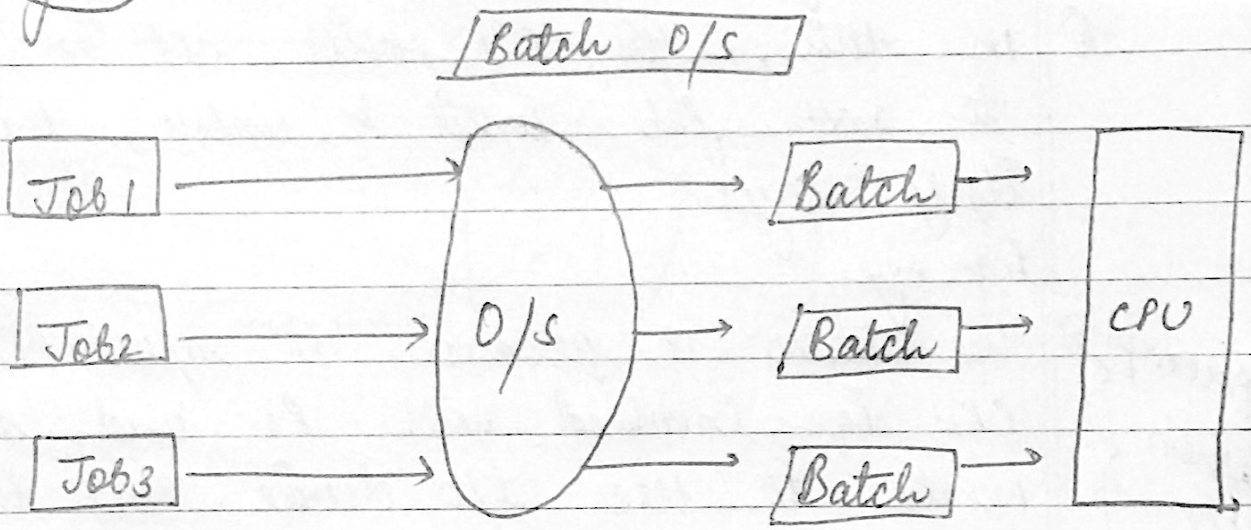
i) Resource Management
ii) User Interface
iii) Memory Management (RAM)
iv) Storage Management (Files/HD)
v) BIOS checking at the time of booting
vi) Process Management by CPU Scheduling algorithms
vii) Security
viii) Treats the problem of process interferance (when a process interferes in other process, it gets blocked)

Types of OS-

i) Batch OS-

Used at the time of punch cards
In Batch O/S, ~~user~~ O/S does not interact with the computer directly.
There is an operator which takes similar jobs having the same requirement ~~sang~~ and group them into batches.
It is the responsibility of operator to sort jobs with similar needs.





Advantages –

i) Multiple users can share the batch systems
ii) It is easy to manage large work repeatedly in batch systems.
iii) Processors of BOS would know how long the job would be when it is in queue.

Disadvantages –

i) Difficult to debug.
ii) costly, CPU will complete one job & then another.
iii) The other jobs will have to wait, if one job fails.
iv) Idle time is more

(2) Multiprogrammed OS

- Just loading more processes into the main m/m, i.e., RAM.
- In this, the CPU will not be given to next job until & unless, the job itself says.

For e.g.

We have 3 processes in m/m P_1, P_2, P_3 CPU is involved with P_1 and now P_1 wants to use I/O device. For that time, when P_1 goes for using I/O device CPU will start executing process P_2.

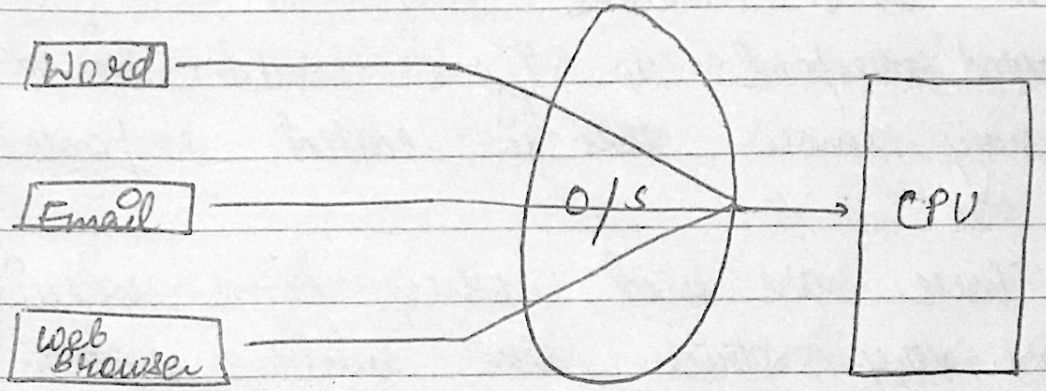
preemption when job itself says

- This reduces the idle time of the CPU

(3) Time Sharing / Multi tasking OS –

- Each task is alloted some time to execute so that all the tasks works smoothly. Each user gets time of CPU as they use a single system.
- The alloted time period is called quantum.
- The tasks can be performed from single as well as multiple users

- After this, the O/S switches to the next task.



Advantages

- Each task gets equal opportunity.
- CPU idle time can be reduced
- supports parallel programming.
- Fewer chances of duplication of s/w

Disadvantages

- Reliability problem.
- Data communication problem.
- One must have to take care of the security and integrity of user programs and data.

E.g. Multics, UNIX, etc.

(4) Real time OS- (both should be online)

- The time interval required to process and respond to I/p's should be very small. This is called response time.
- These are used when time requirements are very strict like missile system, air traffic control system, etc.

Types

Hard Real time

- Time constraints are very strict, even shortest possible delay is not acceptable
- These are build for saving life like automatic parachutes bag.
- Virtual m/m is not found in this system

soft real time

- Time constraints are less strict, even shortest possible delays are acceptable.

Advantages

- **Maximum Consuption:**
  Maximum Utilization of devices and system, thus, more o/p from all the resources.
- Error free.
- Can be embedded due to small size
- The time assigned for task shifting is very less
- Memory is best-managed.

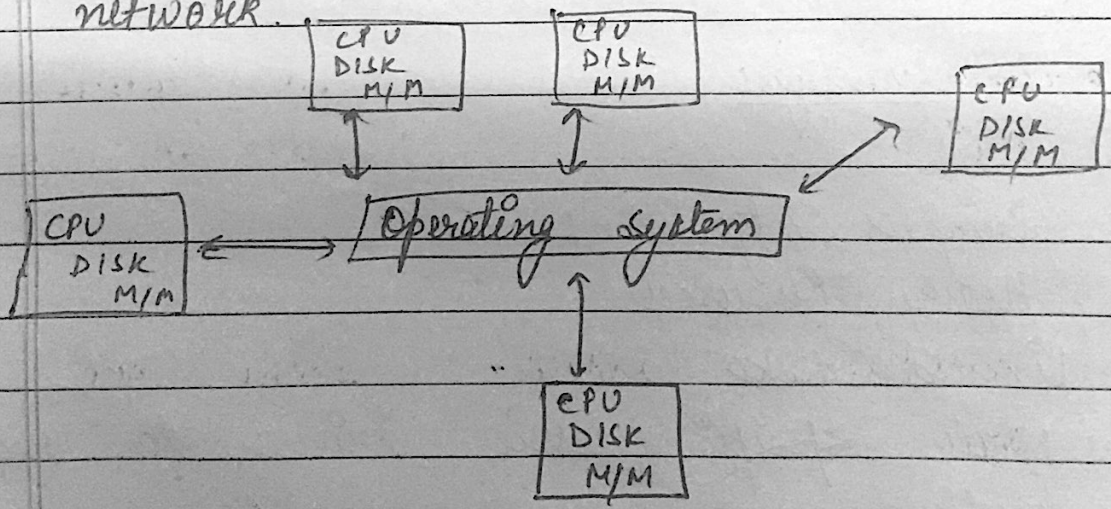
Disadvantages

- Complex Algorithms
- Heavy Resources
- Limited tasks
- Needs specific device drivers to respond earliest.
- Systems are less prones to switching tasks.

(5) **Distributed OS-**
Processing environment is distributed but connected through n/w.

The independent system posses their own memory and cpu. These are referred to as loosely coupled systems or distributed systems.

- All systems posses different size of processors and functions
- One can access the files which are not even present on their system but on some other system because of network.



Advantages -

- failure of one system will not effect other
- load on host computer reduces
- Since resources are being shared, computation is highly fast and durable
- Delay in processing reduces.

- System is scalable.

Disadvantages

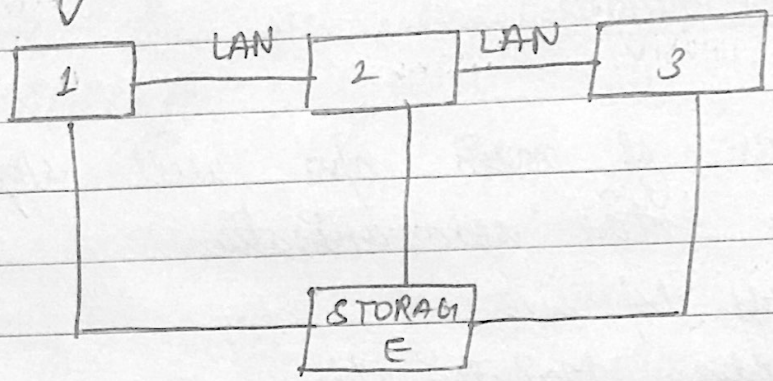
- failure of main n/w will stop the entire communication.
- Very expensive
- Highly Maintainable

6. Clustered OS -

A normal comp. cannot perform all the difficult tasks. so, for complex computation many computers are joined together using LAN or over n/w to form a powerful computer system

- Individual computer systems are joined together.
- Share common storage and closely linked through LAN
- Individual computers are called nodes.
- In most cases, they share hardware and sometimes the OS as well.

- Main goal is to achieve high availability (work will never stop even if node fails)



- Works on the concept of **parallelization**

Types

Asymmetric Clustered	Symmetric Mode
One node is at hot standby mode and works as a backup or substitute	No node is on Hot standby mode
Work is done by other computers	Each node monitors other nodes
This keeps track of other nodes	
If any node fails, then the node on hot stand by will take its place	All the computation work is utilized. Skill

Types

HARDWARE

High performance disk sharing among systems

SOFTWARE

Allows the systems to work together

Advantages

i) High Availability
ii) Cost efficiency
iii) Additional scalability
iv) Fault tolerance
v) High performance
vi) Processing speed is high

Disadvantages

i) Cost effective – cost is high
ii) Required Resources –
Clustering requires additional servers and hardware, making monitoring and maintainance difficult.

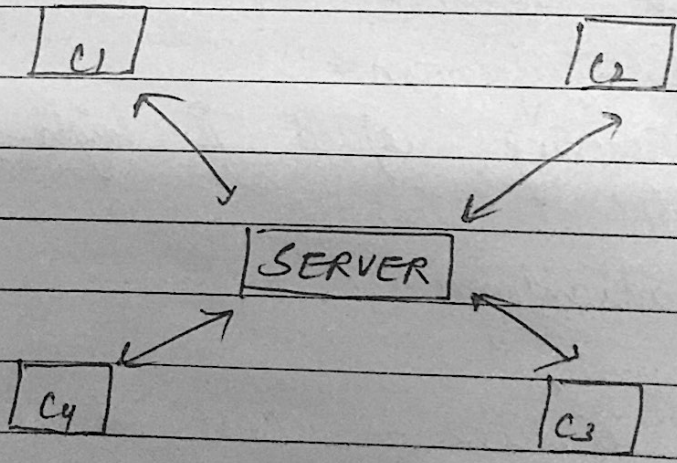
iii) Maintaince –
Isn't Easy to system establishment, monitor, and maintain

⑦ Network OS / Centralized

These systems run on servers and provide capability to manage data, users, groups, security, applications & others networking functions.

⊙ Allows shared access of file, printers, security, applications and other networking functions over a small private n/w.

⊙ tightly coupled systems



Advantages

i) Highly stable centralized systems
ii) Server access is possible remotely.
iii) Security concerns are handled through servers

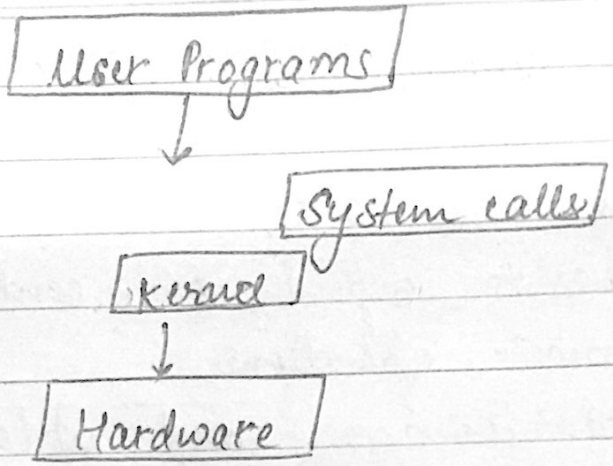
Disadvantages

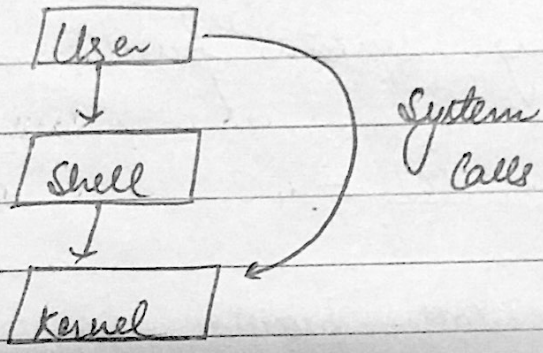
i) Servers are costly
ii) User has to depend on central location for most operations
iii) Maintainance and updates are required regularly.

Operating system functionalities -

1) System call -

System call provide an essential interface between a process and the operating system.
System call invokes kernel to peform a process
To access the functionality of OS, to switch to kernel mode from user mode, then it is called system call or is done with the help of system call
System calls varies from OS to OS.





Types of System Calls -

i) File Related -
open(), Read(), Write(), close(), create(), etc

fopen() → system call open() → Kernel open the file.

ii) Device related() -
used to access device / hardware), we have to use system call to take privilage from OS

Read(), write(), ioctl(), Reposition(), fcntl().

iii) Information related -
information regarding any device / file or ~~any~~ any attribute

getPid(), getPPid(), etc
information or meta data

E.g 

hello.txt) data about data
file name (meta data)

iv) Process control -
For loading any process into the main m/m.
load(), execute(), abort(), fork(), wait(), signal(), Allocate(), etc.

Fork() - multiprogramming environment
parent and child creation for executing multiple processes, simultaneously.

v) Communication -
Intercommunication of process.

Eg pipe(), create / delete(), connections(), shmget()

shmget() getting value of shared m/m

vi) Security related()

chmod(), umask()

fork() system call –

for creating child program.
child will be the clone of parent having ~~id~~ a unique id.
This is used for creating a multiprogramming environment.

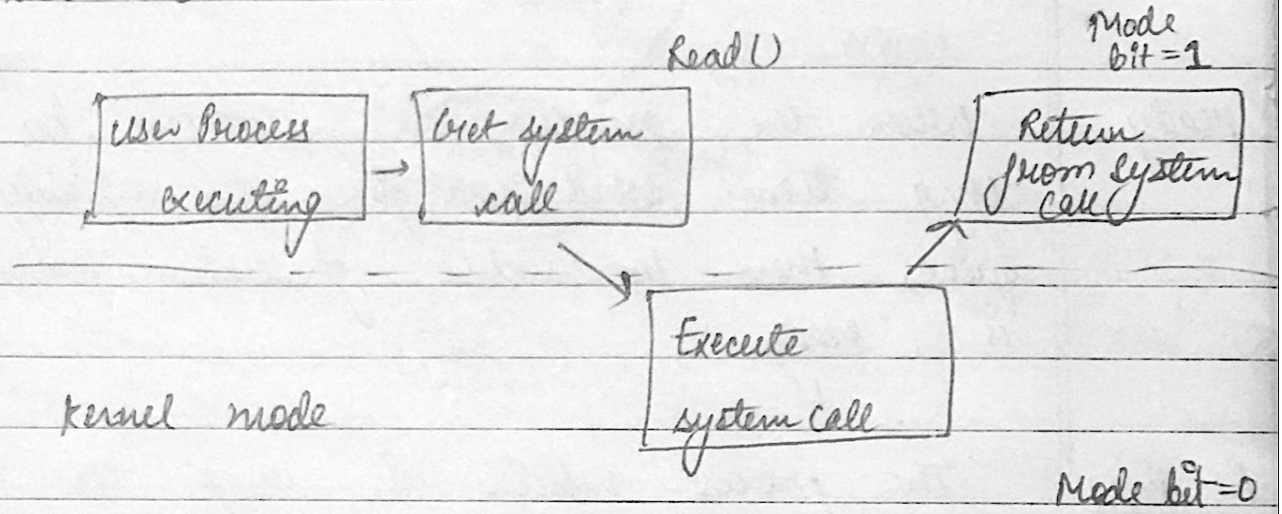
fork() → 0 (child process)
→ +1 +ve → parent
→ -1 → child not created

2^n no. of creation of program

$(2^{n}-1)$ no. of child processes.

```
main(){
fork();
printf("hello"); // will be printed 2 times, one by parent, one by child
}
```

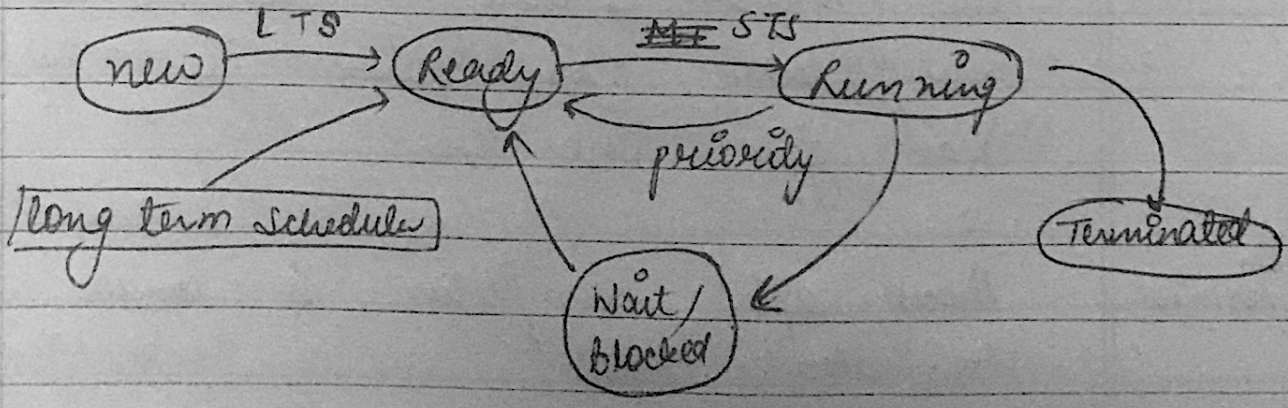
☆ User mode V/S Kernel Mode



* User can't access hardwares/ files/ devices, process, security features directly, so Processor switches b/w User mode & Kernel mode

Our processor is **dual mode**

☆ Process states



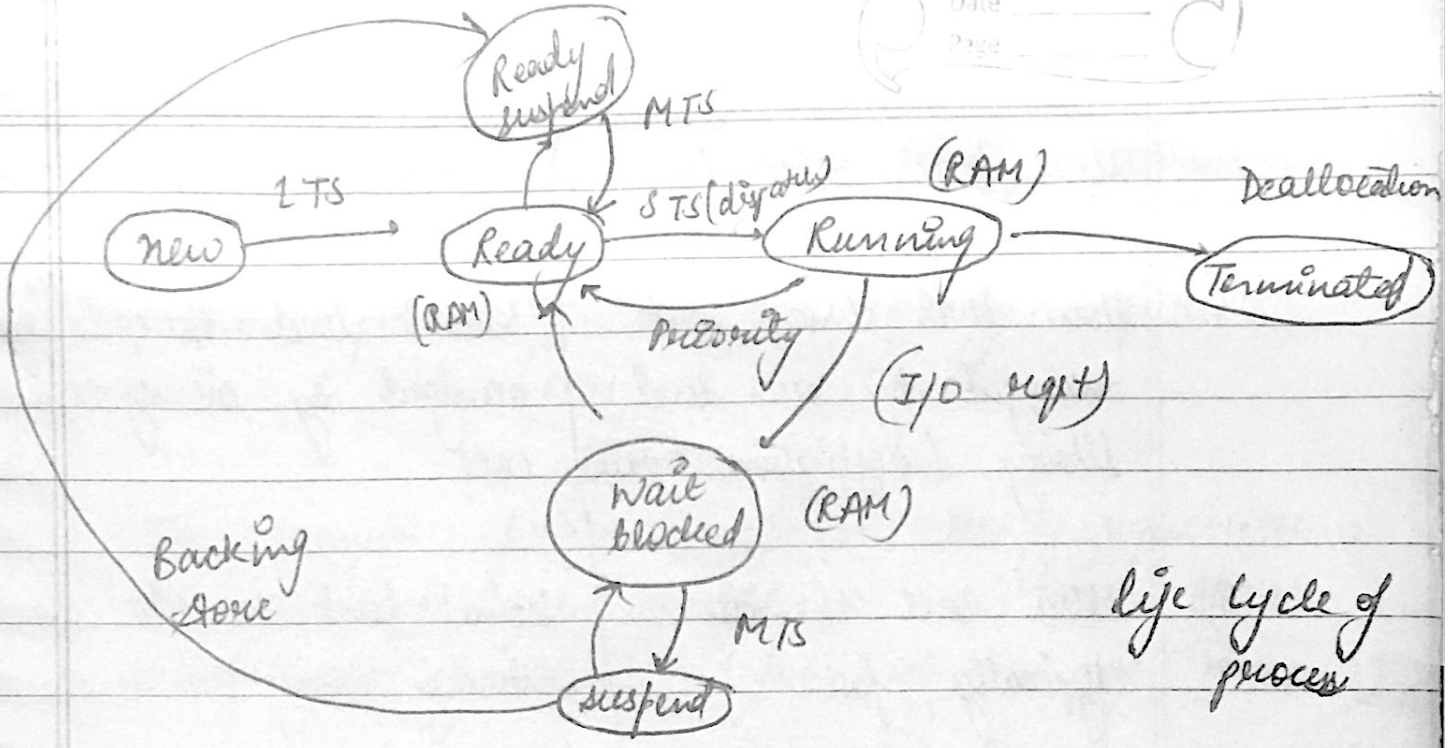
new - when the process is in the secondary storage or is newly created, then it is called new state.

ready - when the process is chosen by the long term scheduler for the ready queue, then the status of that process is ready.

Running - The process which is using the CPU currently or the process under execution is called Running process.
Short term scheduler sends a process from ready to running

Wait/blocked - When a process wants to perform any I/O operation or wants to use any kind of resource in middle of the execution, then it is send to wait/blocked state where it has to wait for some time.

Terminated - Process after completion is under terminated state.



life cycle of process

Process V/s Thread –

	Process	Thread
i)	Heavy weight task	Light weight task
ii)	System calls involved	system calls is not involved
iii)	Context Switching is slower (uses diff. stack + register)	context Switching is faster (vice versa)
iv)	Different processes have different copies of data, files and code	They share same copy of data & code
v)	Independent	Interdependent
vi)	Blocking a process will not block any thread	Blocking a thread will block entire process

	User level thread	Kernel level thread
i)	User level thread are managed by user level library (Application creates)	Kernel level threads are managed by OS system call
ii)	User level threads are typically fast	Kernel level threads are slower than user level
iii)	context switching is faster	context switching is slower
iv)	If one user level thread is blocked, then entire process is blocked	If one kernel level thread is blocked, No effect on others

Process > Kernel > User level thread
(CST)

1. SCHEDULING ALGORITHM—

Scheduling is selecting a process from ready queue and insert it into the CPU for execution

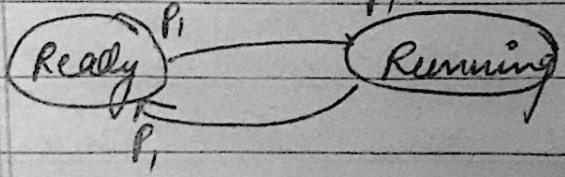
According to the degree of multiprogramming it is our tendency to put as many as possible processes in the ready queue which is in (RAM).

To transfer process from ready queue to the running queue (to CPU), we have to use scheduling algorithms. (for selection of processes)

Scheduling Algorithms

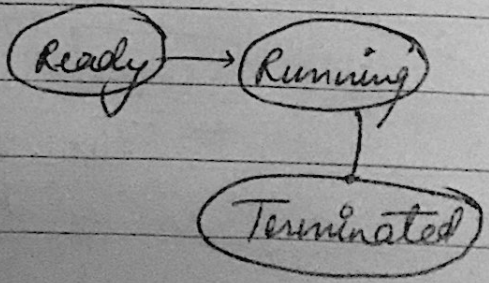
preemptive

(to take CPU from a already running process & take that process to the ready queue)



Non-preemptive

(CPU will not be snatched before the completion of any process)



CPU scheduling time

1. **Arrival time** - The time at which process enter the Ready queue or state.

2. **Burst time** - duration / time required by a process to get execute on CPU.

3. **completion time** - The time at which the process is completed.

4. **Turn Around time** - completion time - Arrival time

5. **Waiting time** - TT - Burst time

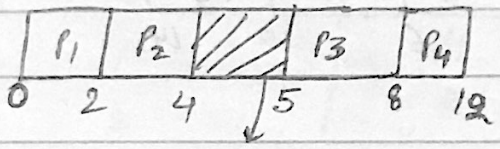
6. **Response time** - (The time at which process gets CPU first) - (Arrival time)

FCFS -

The process which comes first will execute first.
This is a type of non-preemptive scheduling.
This is based on arrival time.
CPU will complete the processing of the

process which arrived first

P	AT	BT	CT	CT-AT = TAT	TAT-BT = WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2



remains idle

* In case of non-preemptive scheduling, Response time = waiting time.

Shortest Job first (SJF)

Schedules the job according to the shortest Burst time.

The job which requires CPU for a shorter period of time will get the CPU first for its execution

It is a non-preemptive scheduling algorithm.

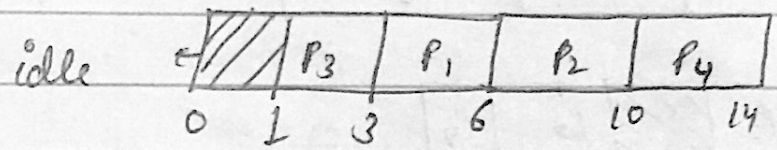
It is also known as shortest job next (SJN) or shortest process Next (SPN)

CT-AT

TAT-BT

Q =

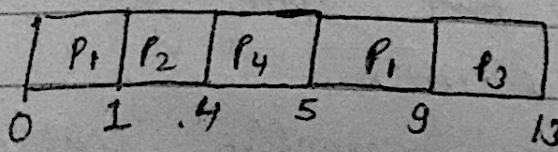
PN	AT	BT	CT	TAT	WT	RT
P_1	1	3	6	5	2	2
P_2	2	4	10	8	4	4
P_3	1	2	3	2	0	0
P_4	4	4	14	10	6	6



Shortest Remaining time first - (SRTF)

SJF with preemption is SRTF

PNO	AT	BT	CT	CT-AT TAT	TAT-BT WT	RT
P_1	0	5	9	9	4	0
P_2	1	3	4	3	0	1-1 = 0
P_3	2	4	13	11	7	9-2 = 7
P_4	4	1	5	1	0	4-4 = 0



* In preemptive RT and WT are different

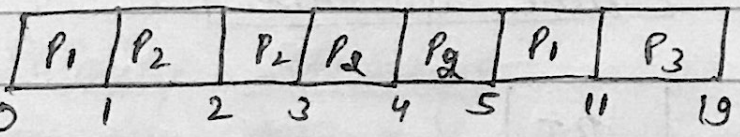
Avg TAT = $24/4 = 6$

Avg WT = $11/4 = 2.75$

Avg. RT = $7/4 = 1.75$

Q

P No	Arrival Time	Burst Time	CT	TAT
P_1	0	7	11	11
P_2	1	4	5	4
P_3	2	8	19	17



Round Robin Scheduling -

In this scheduling, each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation free as all processes get fair share of CPU.
- One of the ~~not~~ most commonly used technique in CPU scheduling at a core.

- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.
- It's throughput is low and the process will have a larger waiting time and response time.

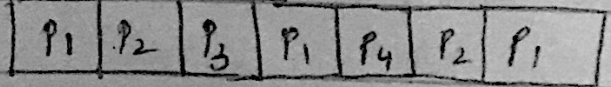
Criteria → Time Quantum

Q

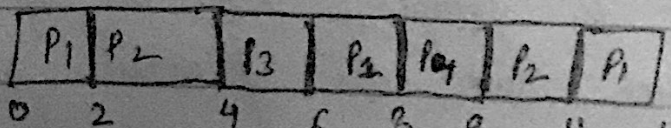
PNo	AT	BT
P_1	0	5 3 1
P_2	1	4 2
P_3	2	2
P_4	4	1

TQ = 2 sec

Ready Queue



Running time



CT	TAT	WT	RT
12	12	7	0
11	10	6	1
6	4	2	2
9	5	4	4

Context Switches = 6

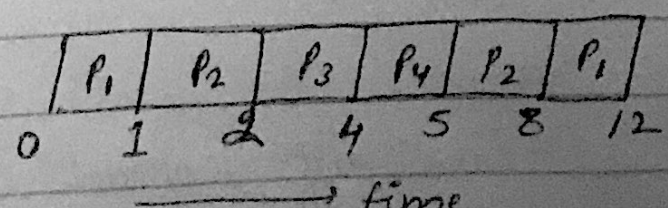
Q How many types context Switches happened?

Context Switching-
Saving the work of previous process and inserting new process in the ready queue.

Priority Scheduling-
Preemptive type of scheduling.

Q

Priority	Process NO	AT	BT	CT	TAT	WT
	P					
10	P_1	0	5	12	12	7
20	P_2	1	4	8	7	3
30	P_3	2	2	4	2	0
40	P_4	4	1	5	1	0



Avg WT = 10/4 = 2.5
Avg TAT = 22/4 = 5.5

The process having higher priority will be executed first by the CPU than the process having low priority.

Higher the number, higher ~~priority~~ will be the priority.

$P_4 > P_3 > P_2 > P_1$

* If two processes ~~have~~ have same priority and same arrival time, then they will be executed according to FCFS.

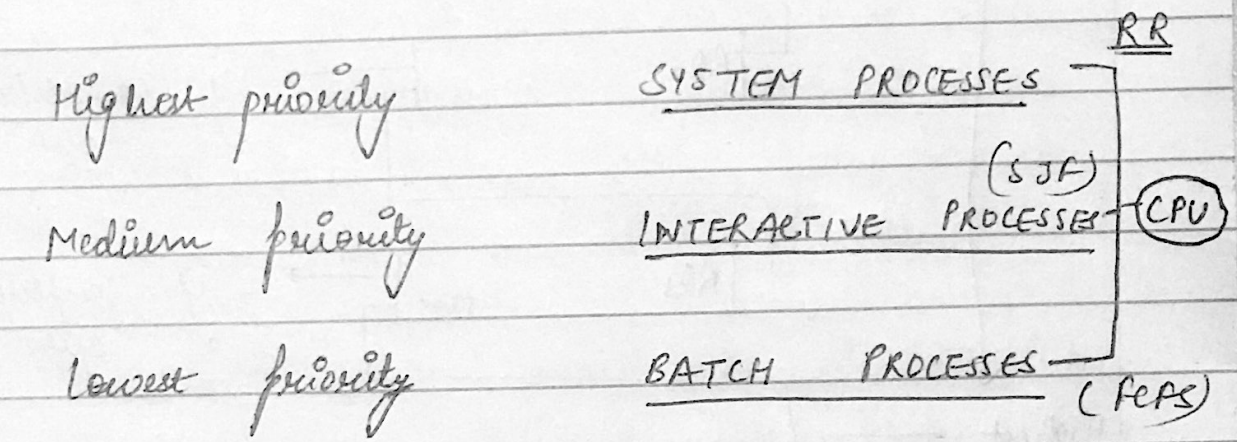
Multilevel Queue scheduling-

In the previous algorithms, we had only one queue in which process exists. The processes were transferred from READY state to RUNNING state.

But in real time environment, we have multiple types of processes like system processes, user processes having diff. priorities.

In MLQS, we have diff. queues based on

their priorities, scheduled by different scheduling algorithms.

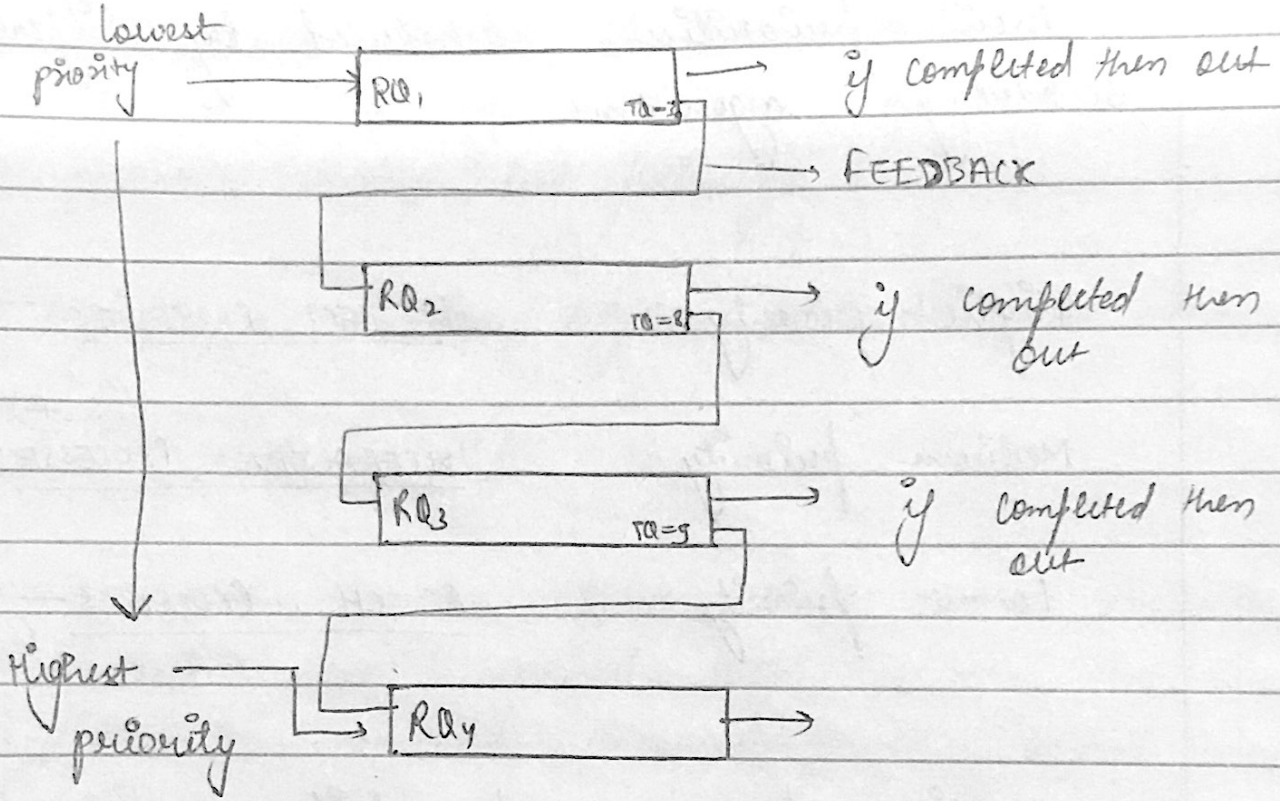


→ Interactive process - to which we interact
→ Batch process - which runs in the background

Multilevel Queue Scheduling (with feedback)

If our system contains a multiple processes in the ready queue, the process having least priority will not get CPU and have to wait for a longer period of time. This, problem is called starvation

The One of the method of resolving this problem is with the help of feedback



If the lowest priority queue process is not getting CPU for execution, it will be ~~shifted~~ from ~~higher~~ lower priority queue to ~~lower~~ higher priority queue, to avoid the situation of starvation

This shifting process is done because of the addition of feedback

☆ Process synchronization

Our system executes data in 2 forms

1. Serial
2. Parallel

1. SERIAL –

In this, one task is completed at a time and all the tasks are executed by the processor in a sequence (one after the other)

→ In serial processing execution of one process doesn't affects another process

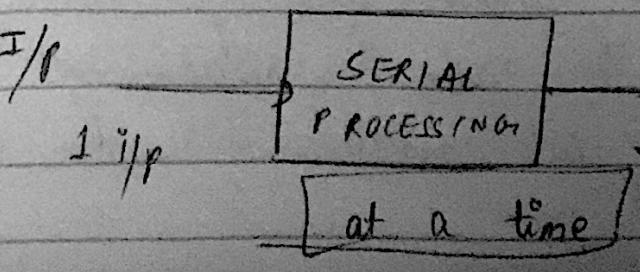
→ The performance of serial processing is higher than parallel processing.

→ Work load of processor is higher

→ Data is transfered in bit by bit format

→ Requires more time.

→ less costly than parallel processing.



Independent processes don't share resources

Nand
Date
Page

(2) PARALLEL

In this type, multiple tasks are executed by the processor simultaneously. The multiple tasks may be completed at a time by different processors

→ Multiple processors are used in parallel processing
→ Execution of one process might affect another
→ Performance is higher
→ Work load per processor is low.
→ Data transfer are in byte form (8 bit) on clock pulse
→ Requires less time
→ Costly.

Process Synchronization is a way to coordinate processes that ~~use~~ use shared data.
It occurs in O.S among cooperating processes.

Cooperating processes are processes that share resource. The execution of one cooperative process affects another

code, buffer, resources, m/m.

Nand
Date
Page

While executing many concurrent processes process synchronization helps to maintain shared data consistency and cooperating process execution.

Data inconsistency may result in race condition

A race condition occurs when two or more operations are executed at the same time, not scheduled in proper sequence, and not excited in critical section correctly.

Eg

```
int shared = 5;
$ ./a (4)
```

```
P1
int X = shared
X++;
sleep (2);
shared = x;
```

```
P2
int y = shared,
y--;
sleep (1)
shared = y;
```

sharing a common variable 'shared'

Ans should be '5' but, we are getting 4, thus, this is called race condition and for this we need process synchronization

Critical Section-

It is the part of the program where shared resources can be accessed by various cooperative processes

A critical ~~section~~ section is a segment of code that can be accessed by only one single process at a certain instance of time. This section consists of shared data resources that need to be accessed by other processes

The entry to this section is handled by down()/wait() fn, represented as P(). The exit from critical section is controlled by the signal()/up() function. , represented by V()

Only one process can be executed inside the critical section at a time. Other processes waiting to execute their critical sections have to wait until the current process finishes executing its critical section

Nand
Date
Page

ENTRY SECTION (wait())

↓

CRITICAL SECTION

↓

EXIT SECTION (signal())

For creating a solution to achieve synchronization - we must follow these 4 rules-

i) Mutual exclusion } Primary rules
ii) Progress
iii) Bounded wait } secondary rules
iv) No ~~not~~ assumption related to H/W speed.

(1) Mutual exclusion -
Another process cannot use critical section while some process is already using it

(2) Progress - let P_1 and P_2 be 2 processes
if P_1 is blocking P_2 from ~~entro~~ entering into critical section even when critical section is not in use, then we can say there is no progress.

③ Bounded Wait - If one process is using CS, then the waiting time for another process should be finite or for a fixed period of time. The code should be written in such a way that the second process should not starve.

④ No assumption related to H/w speed

The code should be universal (~~Kkkkk~~ Hardware + software independent)

Lock Variable -

→ Works at user level

→ If any process wants to use critical section, it has to acquire a lock variable, completes its tasks and then it releases the lock variable

LOCK = 0 (CS is vacant)

LOCK = 1 (CS is in use)

entry

i) while (LOCK == 1);

ii) LOCK = 1

3) Critical section

4) LOCK = 0

} No mutual exclusion guarantee

Nand
Date
Page

Semaphores

To prevent race condition.

It is an integer variable which is used in mutual exclusion manner (mutual exclusive manner) by various concurrent cooperative processes in order to achieve synchronization.

Semaphore
- Counting (-∞, +∞)
- Binary (0, 1)

i) COUNTING SEMAPHORE -

```
Down ( Semaphore S)
{
    S.value = S.value - 1
    if ( S.value < 0)
    {
        put process (PCB) in
        suspend list, sleep();
    }
    else
        return;
}
```

{ PCB — Process control block where its id, variables, etc are kept }

e.g entry code

if $(S=-4)$, 4 processes are in blocked state.

$S=0$, no process is in suspend/ blocked state.

```
UP ( Semaphore S)
{
    S.value = S.value + 1;
    if ( S.value <= 0)
    {
        Select a process from suspended list &
    WakeUp(); // (if S=-ve)
    }
}
```

exit code

Generally FIFO is followed for waking up.

if $\boxed{S=10}$, 10 processes can enter into critical section and complete their operations successfully.

(2) BINARY SEMAPHORE

```
Down ( semaphore s)
{
    if ( s.value == 1)
    {
        s.value = 0;    // Successful operation
    }
    else
    {
    Block the process and place it in
    suspend queue, sleep();
    }
}
```

entry

```
Up ( semaphore s)
{
    if ( suspend list is empty)
    {
        s.value = 1;
    }
    else
    {
        select a process from suspend list
    and wake up()
    } }
```

exit code/ section.

Counting Semaphore -

(1) It has multiple values of the counter The value can range over an unrestricted domain.

(2) It is a structure, which comprises a variable, known as semaphore variable that can take more than 2 values and a list of task or entity, which is nothing but a ~~value~~ process or the thread.

(3) The value of semaphore = no. of process or thread that are to be allowed inside critical section.

(4) The value of counting semaphore can range b/w 0 to N, where N = no. of process which is free to enter and exit the critical section.

(5) As mentioned earlier, a counting semaphore can allow multiple process

or threads to access the critical section, hence mutual exclusion is not guaranteed.

6) Since, multiple instances of process can access the shared resources at any time, counting semaphore guarantees bounded wait, implies that no process will starve.

★ Binary Semaphore

i) Value ranges over 0 and 1.

ii) It is nothing, but similar to a lock, with two values, 0 and 1.
Here 0 means busy while 1 is free.

iii) The idea behind using a binary semaphore is that, it allows only one process at a time to enter critical section.

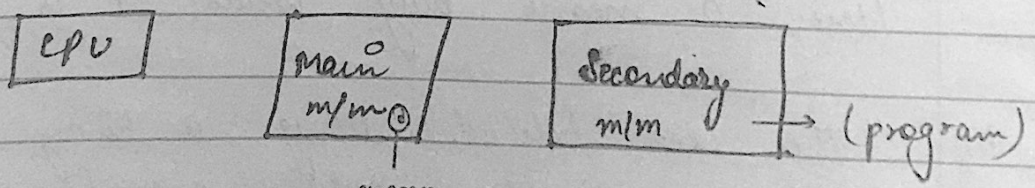
iv) Here, 0 represents that a process or a thread is in the critical section, while the other process or

thread should wait for it to complete, 1 means that no process is accessing the shared resource, and critical section is free

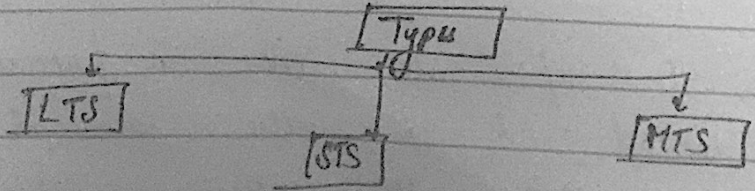
5) Guarantees mutual exclusion, since no two processes can use critical section at same time.

6) Since it is just a variable, which holds an integer value, it cannot guarantee bounded wait. It may happen, and this might cause starvation.

Types of Schedulers-



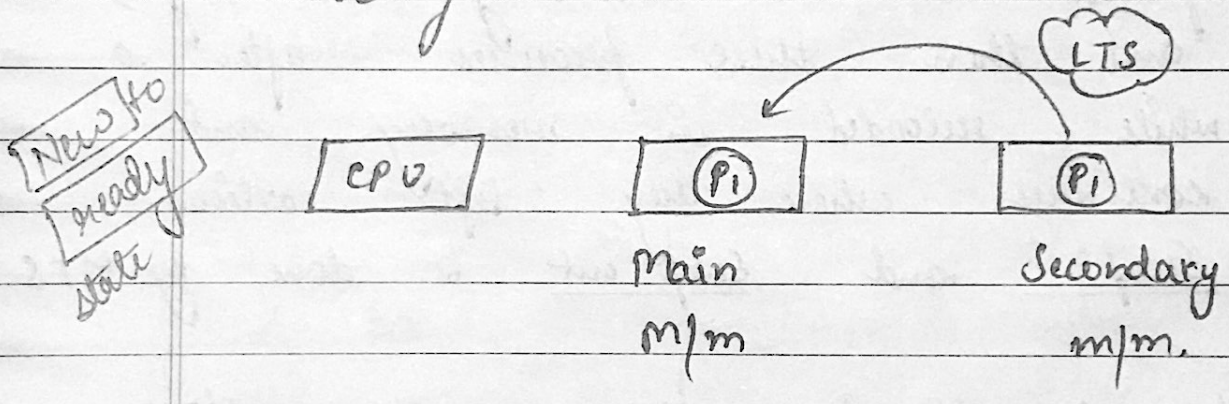
process — program under execution.



PAGE No. :
DATE : / /

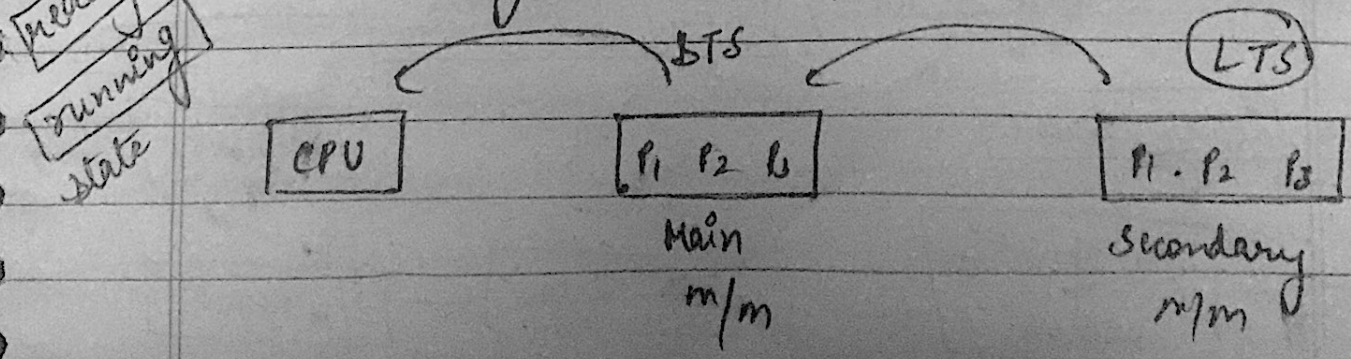
① Long Term Scheduler (LTS)

It selects processes from the queue and loads them in the main memory when the process changes the state from new to ready, then there is use of LTS.



② Short Term Scheduler (STS)

It is also known as CPU scheduler or dispatcher. STS selects a process among the processes that are ready to execute and allocates CPU to one of them. It means, STS/CPU scheduler make the decision of which process to execute next.

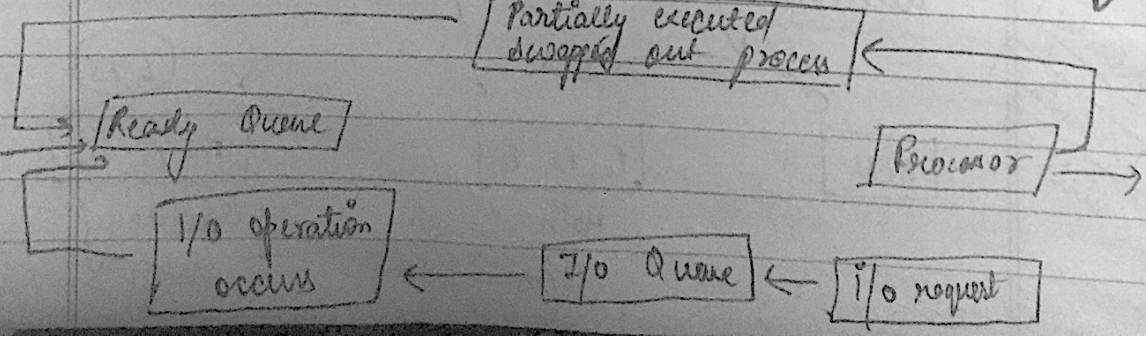


PAGE No. :
DATE : / /

3 Middle term Scheduler (MTS)

It is a part of swapping.
Some running process requires I/O operation. In this, condition, process suspends from main m/m and placed on the secondary memory, and then these processes after a while reloaded in memory and continues where they left earlier.
Swap in and swap out is done by MTS.

If a running process makes I/O request, it becomes suspended.
A ~~secondary~~ suspended process will not show any progress towards completion so it should be moved to secondary storage, hence providing space for new process.
It reduces the degree of Multiprogramming



Nand
Date
Page

☆ Process Synchronization

(1) Dining philosophers problem - using Semaphores

The dining philosophers problem is the classical problem of synchronization which says that five philosophers are sitting are around a circular table and their job is to think and eat alternatively
A bowl of noodles is placed at the center of the table along with 5 chopsticks for each of the philosophers

To eat a philosopher needs both their right and left chopstick. A philosopher can only eat if these are available. ~~It can~~
In case if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their (either left or right) chopstick and starts thinking again

```
void philosopher (void)
{
    while (true)
    {
        Thinking();
        take_fork(i);          <-- left fork
        take_fork((i+1) % N);  <-- Right fork
        Eat();
        put fork(i);
        put fork((i+1) % N);
    }
}
```

i = no. of that philosopher

// 2 philosophers can use the CS at the same time but the processes/philosophers should be (independent) (shouldn't use the same forks)

```
void philosopher (void)
{
    while (true)
    {
        Thinking;
        Wait ( take_fork( Si ))
        Wait ( take_fork( S(i+1) mod N )
```

Nand
Date
Page

0 = Busy
1 = free

```
EAT();
Signal ( put_fork(i));
Signal ( put_fork(i+1)%N)
}
}
```

{ In this, a deadlock can also occur, if one fork is occupied by every process and they can't get another one }

If all the semaphores have value '0', it will cause a deadlock.

The deadlock can be removed by reversing the sequence of taking fork of last philosopher.

for (N-1) philosopher-

```
void philosopher (void)
{
    while (true)
{
wait( take_fork(i))
wait( take_fork ( (i+1)%N)
EAT();
```

```
        signal ( put_fork (si);
        signal ( put_fork (i+1) % N)
    }
}
```

for Nth philosopher -

```
wait ( take_fork ( s(i+1) mod N));
wait ( take_fork ( si));
```

(2) ~~Reading~~ Reader-Writer problem -

A database can be accessed by 2 types of people - readers and writers

Readers - who only reads the data
Writers - who writes the data

If two readers are accessing the data or reading the data at the same time, then there is no problem.
But if two writers or a reader and a writer access the data / object at the same time, there may be problems.

To solve this problem, a writer should get exclusive access to an object, i.e, when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the eject at the same time.

```
int rc = 0;                          // readcount = 0   (no. of readers in Buffer)
Semaphore mutex = 1                  // mutualexclusion = 1
Semaphore db = 1                     // db = database (CS)
void Reader (void)
{
while (true)
{
down (mutex)
rc = rc+1;
if (rc==1), then down (db);
up (mutex)
DB
}
}
```

Binary semaphore

entry code

for writer

```
void writer (void)
{ while (true)
{
down (db);
DB
up (db);
}
}
```

exit code

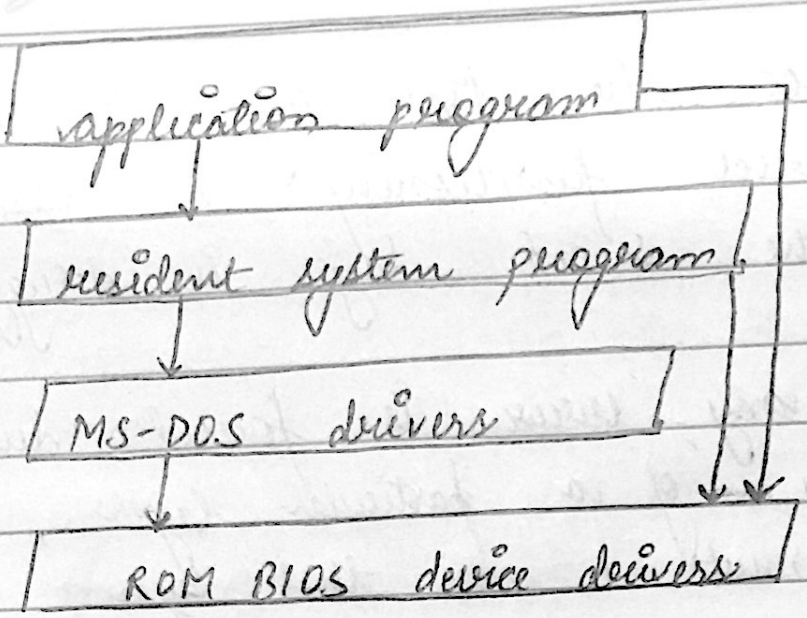
```
down ( mutex)
rc = rc-1
if (rc == 0) then up(db); // no reader in db.
up ( mutex)
Process_data
}
}
```

☆ Structure of OS -

There are 4 types of OS structure

1) Simple structure — oldest

a. Not well designed
b. Layers are not seperated
c. Not divided into modules
d. MS-DOS ~~and UNIX~~ is ~~are~~ example
e. changes or errors in any layer will effect another as all are interconnected
f. All layers are connected to device drivers, which means they can access the hardware directly.



(2) Layered Approach

- OS is broken into a number of layers (levels) (~~layer 0~~) each built on top of lower layers.
- Divided in Modules
- Bottom layer (layer 0) is the hardware and the topmost layer (layer n) is the user interface. The main advantage of this ~~lay~~ approach is modularity.
- This mainly simplified debugging and system verification, i.e, the first layer can be debugged without concerning the rest of the system.

Once the first layer is debugged, it's correct functioning is assumed while the second layer is debugged and so on.

If any error is found during the debugging of a particular layer, the error must be on that layer because the layer below it is already debugged.

layer 5	User Program
layer 4	I/O Management
layer 3	Operator Process Communication
layer 2	Memory Management
layer 1	CPU scheduling
layer 0	Hardware

Each layer can only implement the operations of layer below it. A layer does not need to know how these operations are implemented, it only needs to know what these operations do.

It is less effient than non-layered system

e.g UNIX

Nand
Date
Page

3) Kernel structure

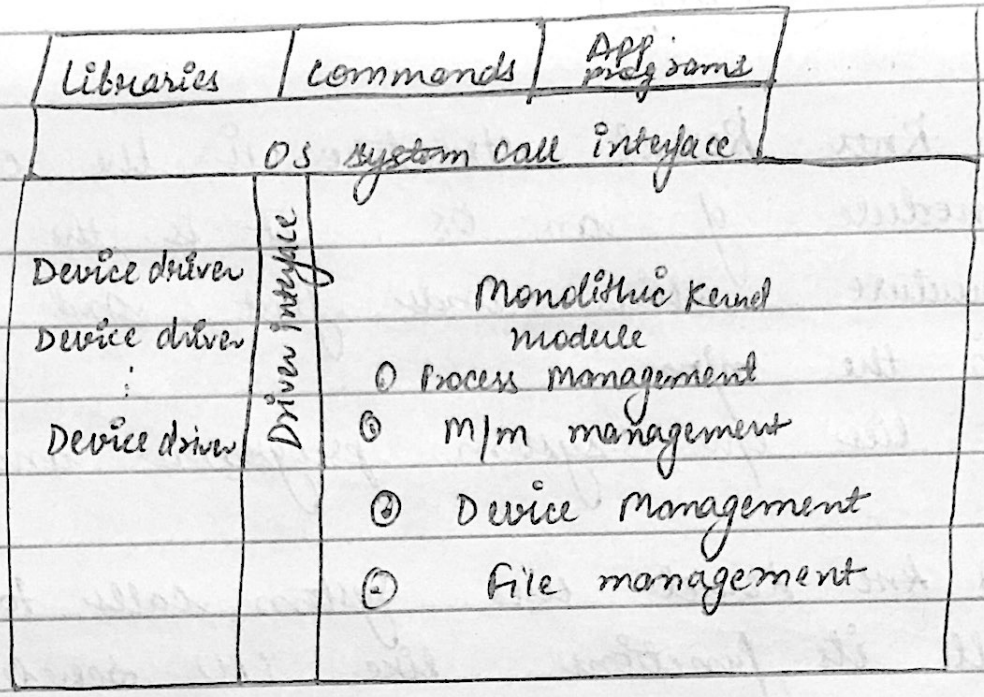
A ~~Kne~~ Kernel structure is the control module of an OS. It is the one structure which loads first and remains in the m/m.
It lies b/w system programs and hardware

This ~~kne~~ kernel uses system calls to perform all its functions like CPU scheduling, m/m management, etc.
It is also known as command line interpreters

① Provides Mechanism for inter process communication

② Provides mechanism for creation and deletion of process

③ Provides CPU scheduling, m/m management and I/O management



Monolithic Kernel structure

(iv) Micro Kernels

This structures the OS by removing all non-essential portions of the kernel and implementing them as system and user level programs

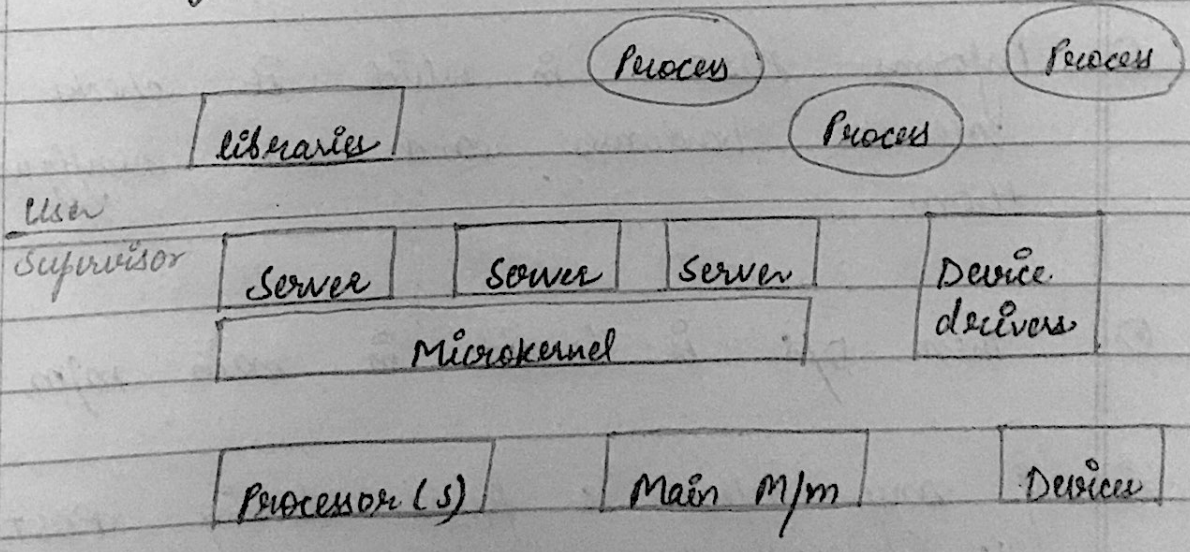
① Generally they provide minimal processes and m/m management, and a communication facility

② The communication b/w components of OS

is provided by message passing.

The benefits of microkernels are as follows

- Extending the OS becomes much easier
- Any changes to kernel tend to be fewer, since the kernel is smaller
- Provides more security and reliability
- Main disadvantage is poor performance due to increased system overhead from message passing



A Microkernel Structure