

Introduction to Operating System

An OS is an integrated set of system programs where major functions is to manage resources (like CPU, memory, disk etc.); control input and output, schedule job, handle error, provide security and act as an interface b/w user and machine.

Basic function of OS

1) Resource management

↳ time management (CPU and disk scheduling)

↳ space management (main memory and secondary memory)

↳ process synchronization (ordering of process to avoid ambiguity or resource access error)

↳ Accounting and status information (info. like process state, memory usage, I/O requirements, department etc.)

2) Virtual Memory management

Here the OS allocate a memory or a portion of memory as primary memory which can be utilised when some of the process are not executing (in ideal state).

Major goal of OS

1) Optimize the use of computer resources so as to maximize its throughput (no. of process executed per unit time).

2) Create a user friendly computing environment for accessing computer resources.

3) Execute user programs and make solving user problem easier.

Computer Startup

When the power of the computer is turned on the first program that runs is usually a set of instructions that is usually stored in ROM. That examines the system hardware that to check whether all hardware device funcⁿ properly or not.

This will check CPU, memory, basic I/O for error and store the result in special memory location. This is known as power on self test (POST).

Once the POST has successfully executed the software loaded in ROM (firmware) will begin to activate the disk drive of the computer.

When the computer activate the hard drive, then the first piece of system program it finds is the bootstrap loader. The bootstrap loader is a small program whose single function is to load operating system into memory and allow it to begin operations.

Type of OS

Operating system can be classified into ~~two~~ categories based on processing, process scheduling, memory management, I/O management and file management.

① Serial Processing

When programmes are executed in a serial manner the processing is known as serial processing.

Advantages

- It has a big improvement over bare machine (i.e. a machine without a OS).
- OS was limited to only a loader and some I/O devices.

Disadvantages

- Slow processing
- At a particular instance of time very limited resource could be used. This ^{results} ~~decreases~~ ^{poor} the system utilisation and throughput (poor throughput).

② Batch OS

To speed up the processing, the jobs with similar needs were batch together and run through a computer as a group.

Adv

- It has a better throughput compared to serial processing.

Disadv.

- No interaction is possible with the user ~~by~~ while program is being executed.
- CPU set idle during the ~~transition~~ ^{transition} from one job to another job.

NOTE : • In OS process are the programs in execution (it is also termed as job).

- A program is a set of instructions to solve a particular task whereas the process is the program in execution which contains the state information like ready, new, running, idle and terminated.

NOTE The amount of memory and its location allocated by the OS, number of resource required and allocated to the process.

Performance Improvement of an OS

1) Buffering: It is a method of overlapping and processing of a single task. After the data has been read and the CPU is about to start operating on it, the I/O device is instructed to begin next I/O immediately. The CPU and I/O device are now busy at the same instance of time. By the time the CPU is ready for the next data item, the I/O device will have finished reading.

2) Spooling (simultaneous peripheral operations online)

In this technology rather than the card being read from the card reader directly into the memory and then job being processed. Cards are directly read from the card reader into the disk (spool). The location of the card image is recorded in a table by the OS. When a job is executed the OS satisfies its request for card reading I/O operation by reading from the disk.

Similarly when the job request to print a line that line is copied to the system buffer and then that is written to the disk.

Multi programming OS

- Multitasking OS → n no. of user interface with the machine (mainframe)
- multitasking OS → if n no. of job can be executed simultaneously (windows, mac, linux)
- multiprocessing OS

↓
if n no. of processor are used to execute one or more job

Distributed OS

In this case multiple systems located at various location are able to perform user's task cumulatively.
e.g. cloud computing

Real Time OS - (RTOS)

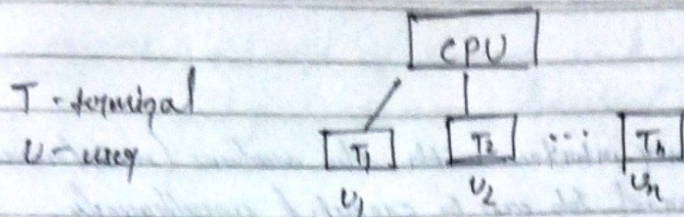
It is a type of OS where the op of a particular task must be generated after a fixed amount of time.

ex. flight simulator

- ↳ Hard Real time OS when the op is expected at a constant remnant of time delay will trigger penalty for the machine.
- ↳ SRTOS in soft RTOS if the target amount of time can be ^{missed} ~~used~~ without any penalty.

Time Sharing OS

It is a form of multiprogramming (multiprogram operating system) OS which operate in an interactive mode with a quick response time. Each interactive user is assigned a time slice. During this time slice user gets control of CPU.



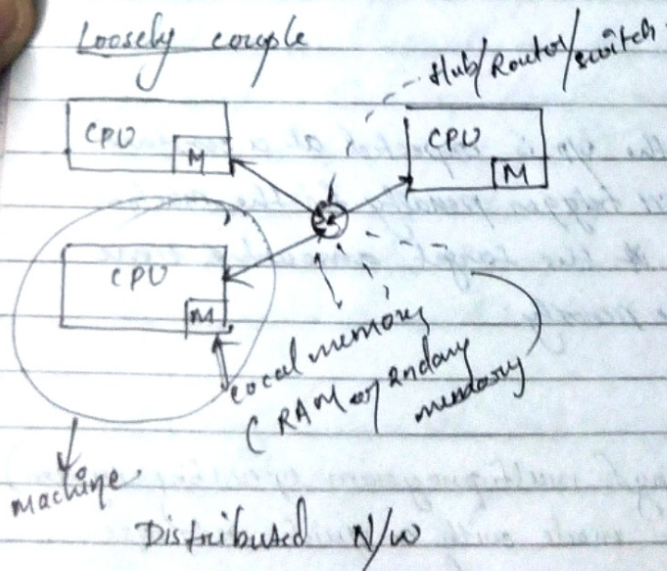
Network Operating system

→ It is defined as a collection of software and associated protocols that allow a set of autonomous computers; which are interconnected by a computer network to be used together in a convenient and cost effective manner.

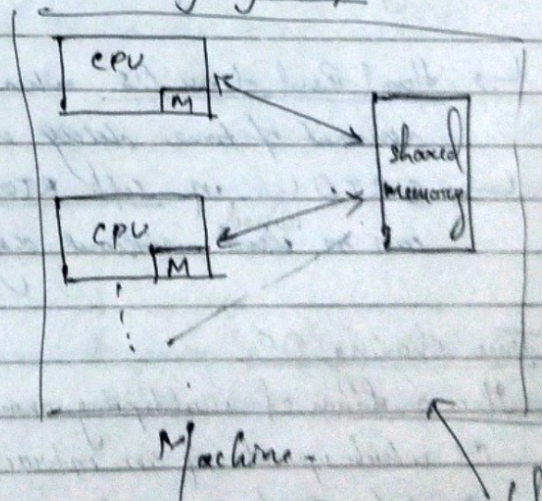
→ Here the users are aware of the existence of multiple computers.

→ They can login to remote machine and copy file from one system to another system.

Loosely couple



tightly couple



→ local memory

Machine -
multiprocessor or
multiprocessing or
Super m/c
→ local + shared
connected with high speed bus

Interrupt (i/s)

→ Occurrence of an event is usually signal by an interrupt from either software or hardware.

→ TRAP (exception) is a software generated interrupt caused due to any error.

→ I/O Interrupt - to start an I/O operation the CPU loads the appropriate register within device controller. The device controller examines the content of register to determine what action to take. If it finds a read request the controller will start the transfer of data from device to its local buffer.

Once the transfer of data is completed the device controller informs the CPU that it has finished its operation.

I/O operations are of two types:-

1) Synchronous I/O -

I/O is started - I/O request is started and then I/O completion control is returned to the user process.

2) Asynchronous I/O -

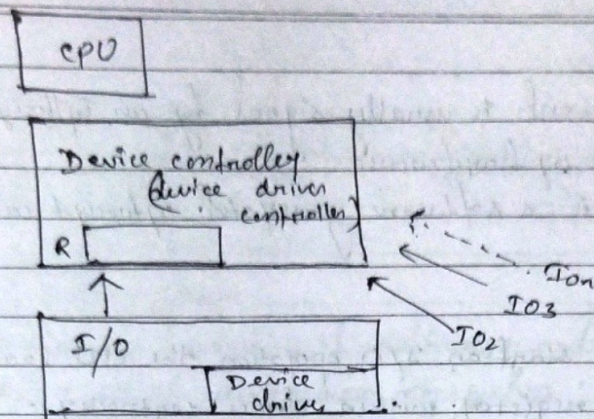
It returns the control to the user program without waiting for I/O completion.

Requesting process

↓
device driver controller

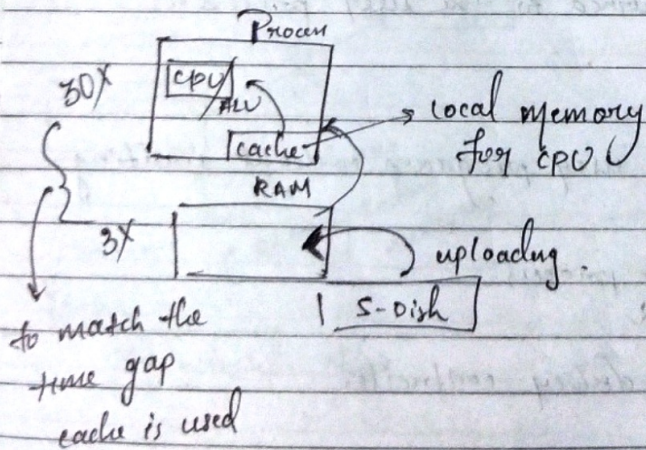
↓
interrupt handler

↓
h/w transfer.



Cohency and consistency

In multiprocessor environment where in addition to maintain internal registry, each of the CPU also contain a local cache. In such an environment a copy of 'A' may exist in several cache simultaneously. Since the various CPU can not execute concurrently, hence an update to a value of 'A' in one cache is immediately ~~reflected~~ reflected in all other caches where 'A' resides. This situation is called cache cohenacy.



Hardware Protection

The OS provides the hardware protection of a system by avoiding unauthorized access to I/O devices and the memory space.

If there is an illegal or unauthorized access of resources, the OS will generate an interrupt or TRAP to notify the user and stop further execution of the system.

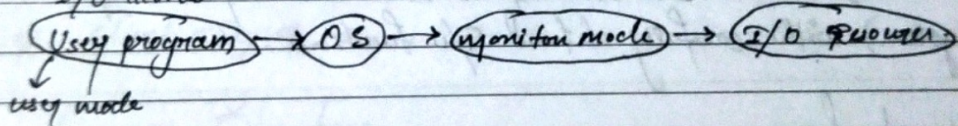
Dual mode operation.

1) User mode

2) monitor mode (privileged mode)

(I/O protection)

I/O operation - During the I/O operation all the user application or program must go through monitor mode in-order to access I/O devices.



Memory Protection

OS providing a protection to the memory management operation by limiting the process to access memory space.

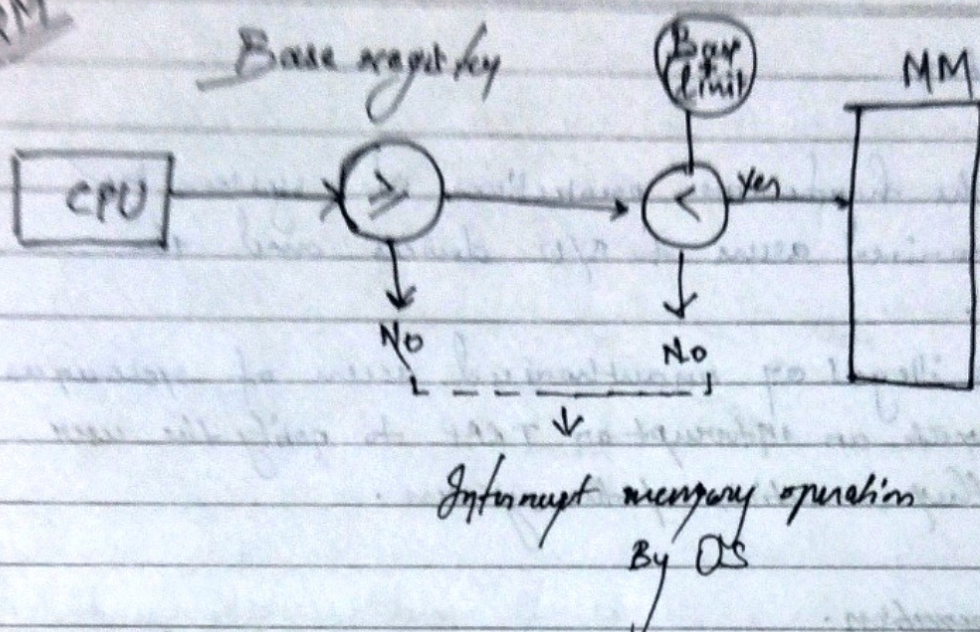
Using base and limit registers OS providing access to the memory space for user process/processes

main memory	
0	OS
1000	process 1
1050	process 2
1150	...
...	process n
2048	

logic addr $\rightarrow$ physical addr $\rightarrow$ load into mem

1000 100 $\rightarrow$ max m allocated
↑ ↓
BA limit space for B
to add new mem.

2M

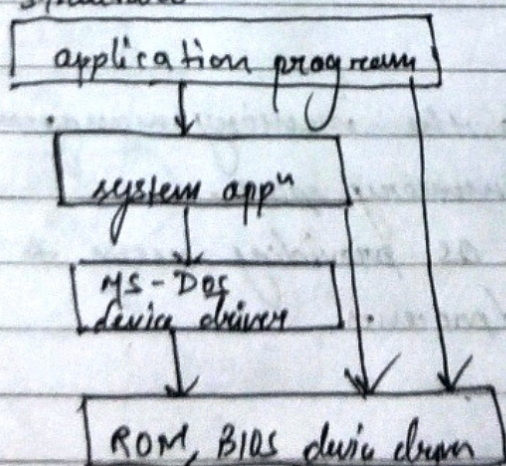


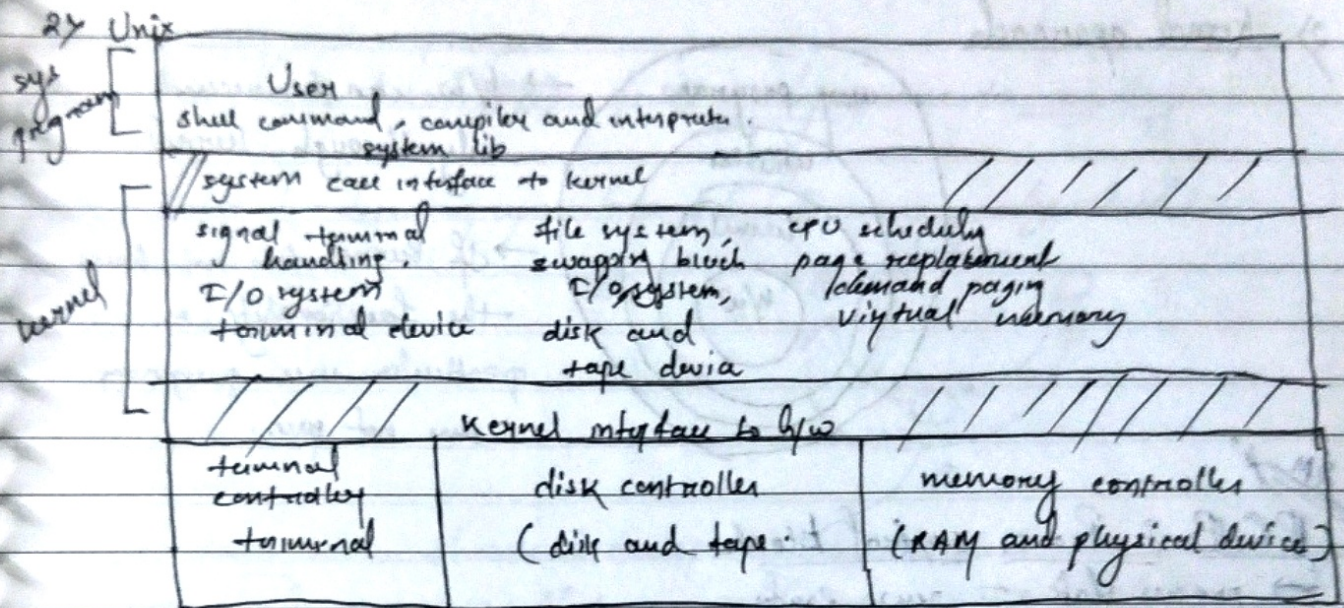
OS Architecture

If a system is to function properly and be modified easily a common approach is to partition the task into small components rather than having monolithic system.

→ Each modules should be a well defined structure (portion of the system) with carefully defined I/O functions.

1) Simple structure

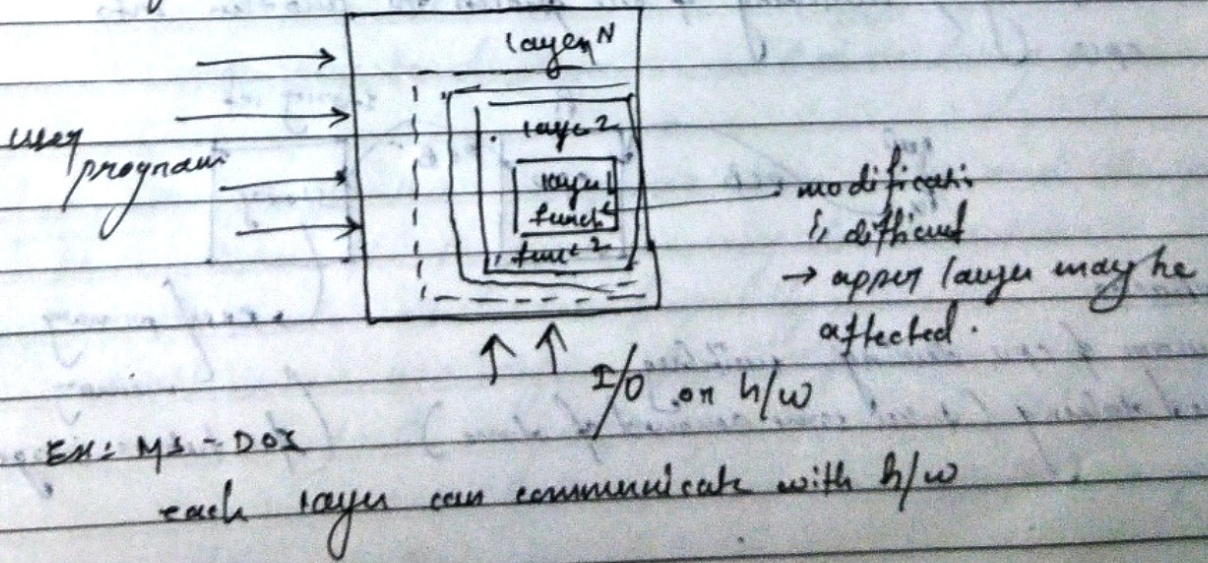




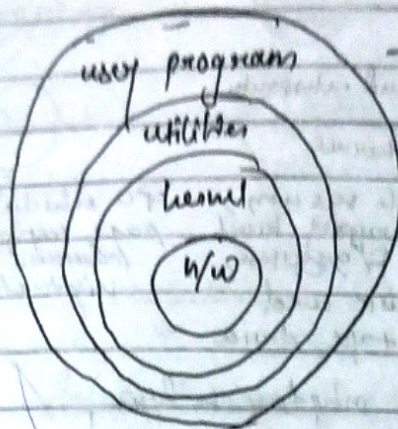
There are four type of os structure -

- layer structure
- kernel approach
- virtual memory machine
- client-server

(i) Layer Structure



2) kernel approach



→ h/w can be accessed only through kernel.

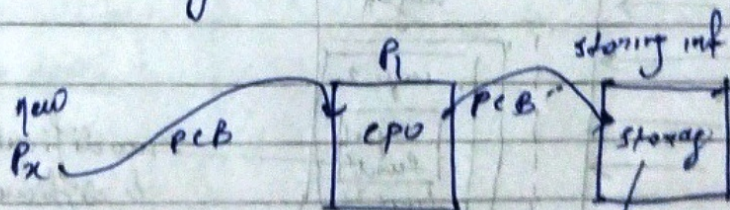
→ If kernel does not have the functionality of a particular user program, it can not run.

2M

PCB : Process control block

- ① → process state - new, ready
 - program counter - next address of an instrⁿ
 - CPU register - stack, Accumulator, 8 bit / 16 bit
 - CPU scheduling information - which algo/method
 - memory management info - swap space, virtual Actual memory
 - Accounting info - CPU usage, time limit of process no. (priority, scheduling)
 - I/O status info - allocating resources. process no. (no. of core, process no.)
- e.g (file open.)

2M // Context switching → switching of one process to another into CPU



→ RAM / primary memory

(for future usage)

Drawbacks

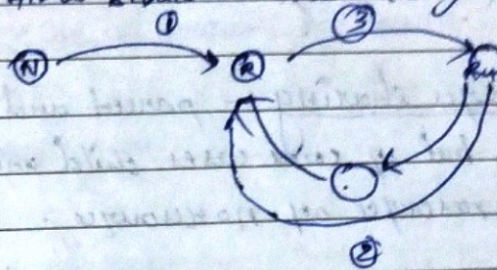
- > no usage of CPU during switching
- > time taking (takes some amount of time)

* Dispatcher

process
→ move data from ready queue to running.

Scheduler (depending upon time scheduler are categorised).

- 1) short-term
- 2) medium-term
- 3) long-term.



Short term: The short term CPU scheduler selects a process among the processes that are ready to execute and allocates the CPU.

→ the short term scheduler must select a new process for the CPU frequently.

medium-term → It schedules the processes at intermediate level of scheduling.

→ process can be described as either

- (i) I/O bound process - spend more time during I/O, then computation.
- (ii) CPU bound process - spend more time during computation.

Long term - It selects the process from disk and loads them into the main memory for execution.

It executes much less frequently.

→ It controls the degree of multiprogramming.

→ because of the longer interval b/w execution, the long term scheduler takes more time to select a process for execution.

Process Creation

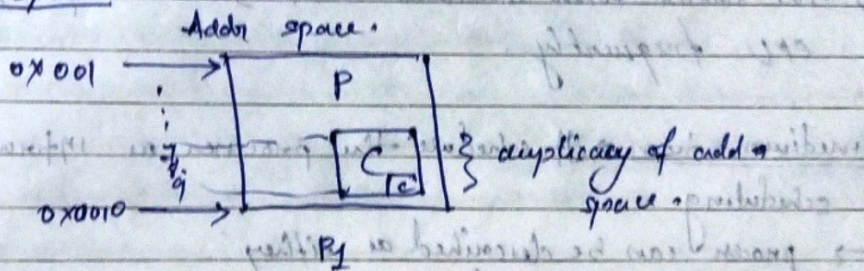
A parent process can create a child process using system call of an OS.

Ex: `fork()` in Linux

→ Resource sharing - parent and child process share all the resources but in some cases child may share a subsection of parent resource or no resource.

→ Execution - parent will wait for the termination of child process.

→ Address space



Process termination - The process can be terminated normally by executing last statement of the program or it may call `_exit()` system call to terminate the program.

→ If data from child to parent can be transferred via `wait()` system call.

→ If the program or process terminates abnormally the exception handler will raise a `abort()` system call.

⇒ Co-operating process

Scheduling Algorithm

→ Basic concept

→ CPU utilisation

→ Increase throughput → no. of process per second

- CPU burst time

- CPU I/O Burst cycle

criteria of scheduling

→ high throughput - $\frac{\text{no. of process}}{\text{per unit time}}$

→ turn around time - total time from arrival to complete.

→ waiting time - total time wait ~~during its execution~~ till

→ Response time completion of process.

(→ first response from CPU after arrival (when it starts execution))

turn around time = $CT - AT$ → arrival

waiting time = $AT - ET$

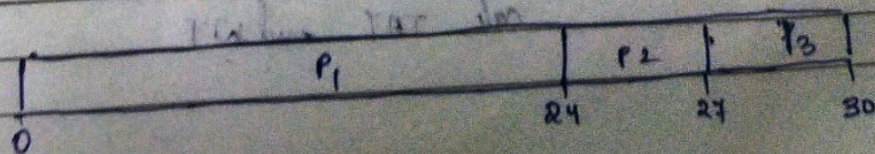
Scheduling algo

1) FCFS (first come first serve)

In this scheduling algo, the process which arrive first in the ready queue will execute first

process	Burst time (ms)
P ₁	24
P ₂	8
P ₃	8

Gantt chart



$$\begin{array}{r} 30 \\ 27 \\ 24 \\ \hline 81 \end{array}$$

Process	Burst time	TAT	WT
P ₁	24	24	0
P ₂	3	27	24
P ₃	3	30	27

$\Sigma TAT = \frac{81}{3} = 27$
 $WT = TAT - Burst\ time$

2) shortest job first (SJF) (non-preemptive)

Process	Burst time	TAT	WT
P ₁	6	9	3 (9-6)
P ₂	8	24	16 (24-8)
P ₃	7	16	9 (16-7)
P ₄	3	3	0 (3-3)

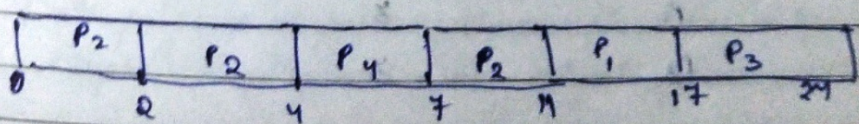
$\Sigma TAT = 4$ $\Sigma WT = 4$

```

    0   3   9   16  24
    |---|---|---|---|
    P4 P1 P3 P2
          
```

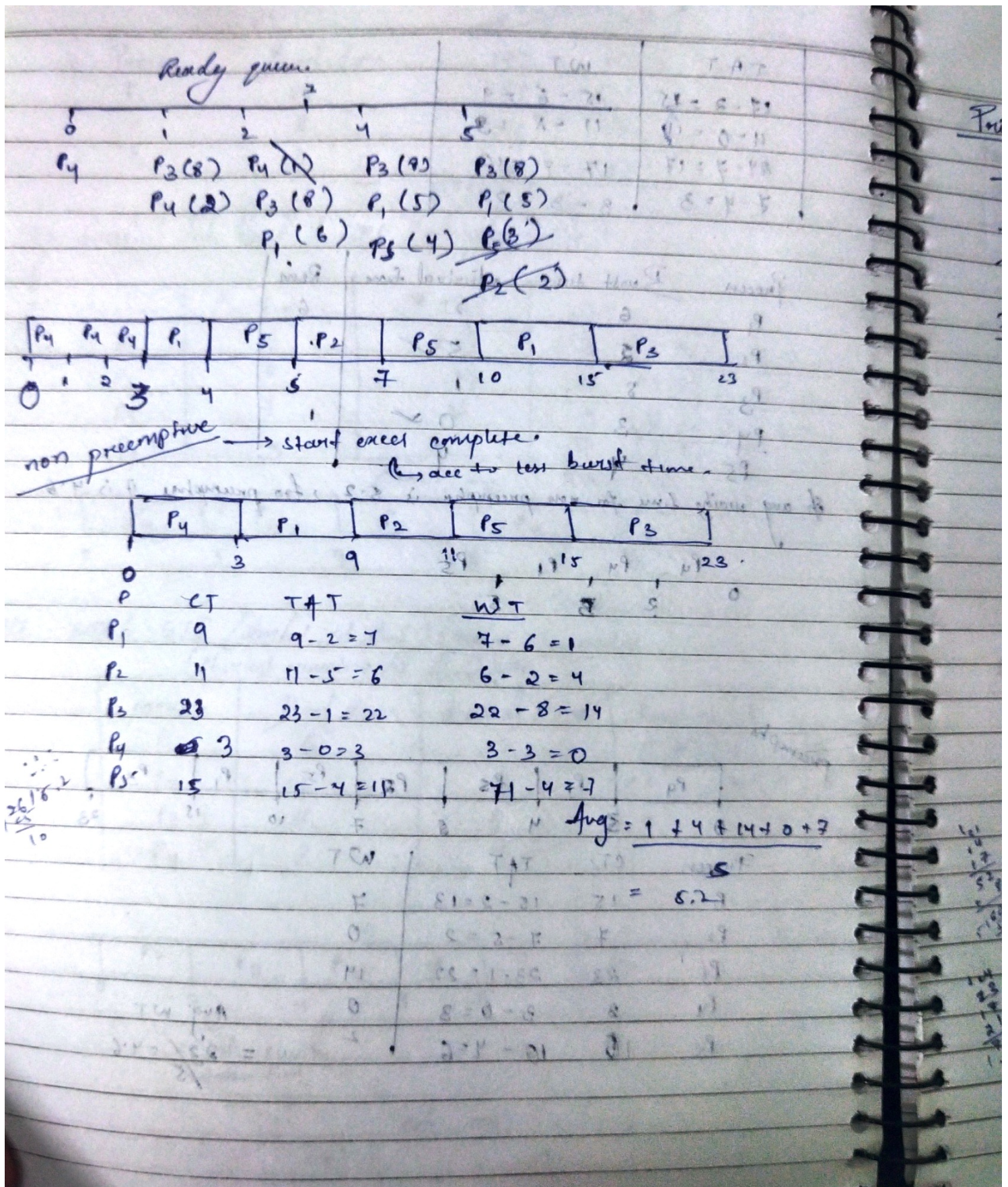
3) ~~SJF~~ SJF (shortest job first) (preemptive)
(shortest remaining job first) SRJF

Process	Burst time	Arrival time	Remaining
P ₁	6	2	5 (6-1)
P ₂	8	0	6 (8-2)
P ₃	7	4	3 (7-4)
P ₄	3	4	0



context switch
taken time

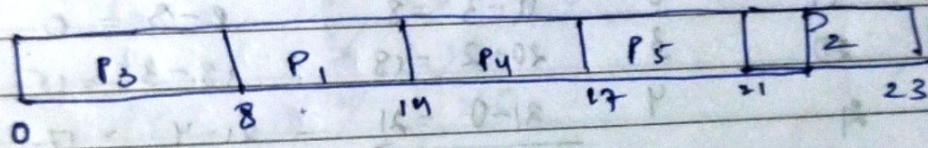
calc TAT and WT



Priority scheduling

This scheduling selects the process with high priority. The smallest value indicates high priority and higher value shows low priority.

Process	B.T	Priority
P ₁	6	2 ✓
P ₂	2	5
P ₃	8	1 ✓
P ₄	3	3
P ₅	4	4



P	CT	TAT	WT
P ₁	14	14 - 0 = 14	14 - 6 = 8
P ₂	23	23 - 0 = 23	23 - 2 = 21
P ₃	8	8	8 - 8 = 0
P ₄	17	17	17 - 3 = 14
P ₅	21	21	21 - 4 = 17

$$\sum WT = 8 + 21 + 14 + 17 + 0 = 60$$

$$\sum TAT = 14 + 23 + 8 + 17 + 21 = 83$$

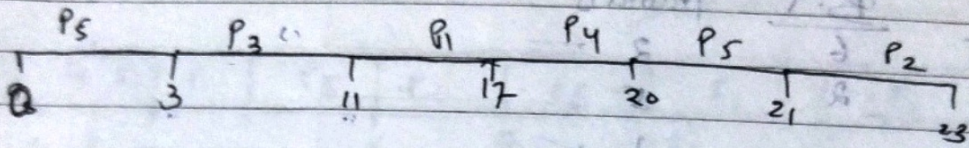
$$\begin{array}{r} 14 \\ 23 \\ 8 \\ 17 \\ 0 \\ \hline 62 \end{array}$$

$$\begin{array}{r} 14 \\ 23 \\ 8 \\ 17 \\ 21 \\ \hline 83 \end{array}$$

HW

Preemptive

Process	Burst	Priority	Arrival time
P ₁	6	2	1
P ₂	2	5	4
P ₃	8	1	3
P ₄	3	3	2
P ₅	4	4	0



Process	AT	CT	BT	STAT	WT
P ₁	1	17	6	17-1 = 16	16-6 = 10
P ₂	4	23	2	23-4 = 19	19-2 = 17
P ₃	3	11	8	11-3 = 8	8-8 = 0
P ₄	2	20	3	20-2 = 18	18-3 = 15
P ₅	0	21	4	21-0 = 21	21-4 = 17

$$\sum STAT = \frac{82}{5} = 16.4$$

$$\sum WT = 11.0$$

$$S = \frac{1}{n} \sum_{i=1}^n (STAT_i + WT_i)$$

$$S = \frac{1}{5} (16.4 + 11.0) = 5.8$$

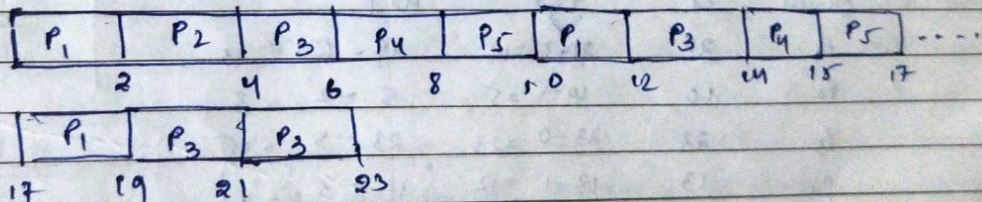
Round Robin scheduling (RR)

~~Process~~ ~~Burst time~~ (non-preemptive)

In this algo. all the process which are present in the ready queue are allocated a fixed cycle time for the execution. If a process is not able to complete its execution during given cycle time then the process wait for next cycle time.

It is same as first come first serve scheduling algo. with a fixed cycle time for each process (for the utilisation of CPU).

Process	Burst time (ms)	cycle time = 2ms.
P ₁	6	4 x 0
P ₂	2	0
P ₃	8	6 x 0
P ₄	3	1 x 0
P ₅	4	1 x 0



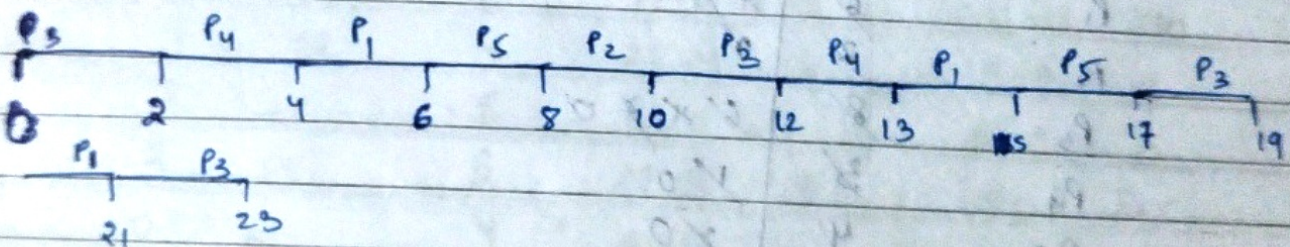
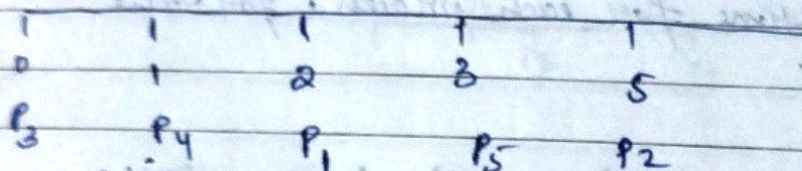
Process	CT	TAT	WT
P ₁	14	14 - 0 = 14	14 - 6 = 8
P ₂	4	4	4 - 2 = 2
P ₃	21	21	21 - 8 = 13
P ₄	15	15	15 - 3 = 12
P ₅	17	17	17 - 4 = 13

$\frac{21}{13}$

4W

use queue.

Process	B.T (ms)	Arrival time	(Arrival time - delay only does not affect)
P ₁	6 2	2	
P ₂	8 0	5	
P ₃	3 10	0	
P ₄	4 20	1	
P ₅	5 0	3	



Process	CT	TAT	WT
P ₁	21	$21 - 2 = 19$	$19 - 6 = 13$
P ₂	10	$10 - 5 = 5$	$5 - 2 = 3$
P ₃	23	$23 - 0 = 23$	$23 - 3 = 20$
P ₄	13	$13 - 1 = 12$	$12 - 3 = 9$
P ₅	17	$17 - 3 = 14$	$14 - 4 = 10$

Multi-level (2M)

In multilevel scheduling multiple scheduling methods are used to schedule different type of task (such as one scheduler for foreground process (interactive process) and other scheduler will be for background process)

Preemptive scheduling

Here once a CPU allocated to a process can be taken away because of the scheduling method or due to high priority task.

non-preemptive CPU

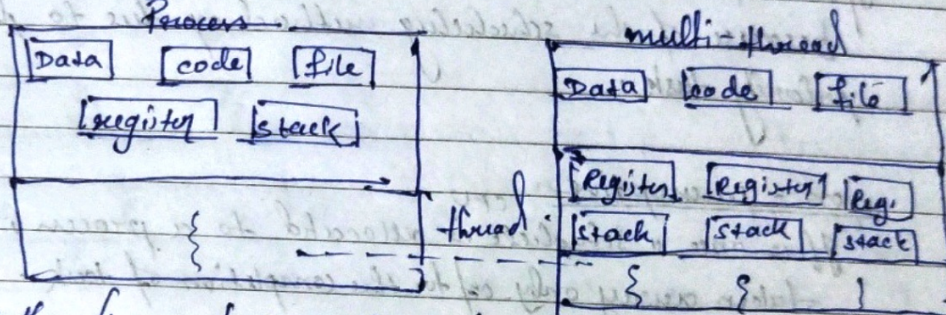
Here once a scheduler is allocated to a process, it can be taken away only after the completion of task.

Questions assignment (1)

- 1) Features of an operating system?
- 2) Differentiate b/w kernel OS and DOS.
- 3) What is the diff b/w multiprogramming and multiprocessing systems?
- 4) Diff b/w batch processing and multiprocessor OS.
- 5) Diff b/w distributed OS and centralised system.
- 6) What are the main services ^{of} that can be accessed using system call?
- 7) Diff b/w child and the parent process.
- 8) What is scheduling. Describe the different type of scheduler.
- 9) What are the diff type of scheduling algo present in OS to schedule the processes.
- 10) Which algo is best and why? (choose any one)

Thread

- A thread is known as light weight process (LWP).
- It is a basic unit of CPU utilization.
- A thread contains thread id, program counter, set of registers and stack.
- It ~~may~~ share code section, data section and other operating system resources such as open file and signals.

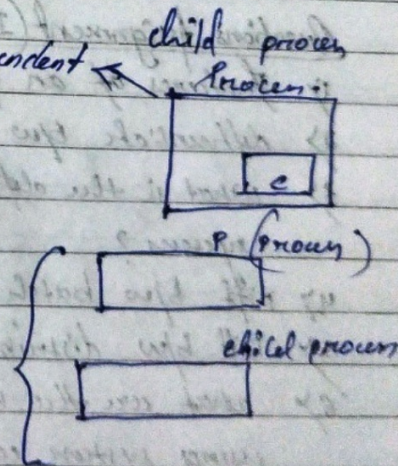


→ threads never block each other's execution.

Benefit

- 1) Responsiveness → less time for context switching
- 2) Resource sharing
- 3) Economy → memory allocation
- 4) Utilization and multiprocess architecture.

independent
(in terms of
resources)



User level thread implement in user level libraries rather than via system call. So thread switching does not need to call OS and do cause interrupt to the kernel.

Advantages

- 1> user level thread does not require modification to OS.
- 2> simple representation (each thread is represented by PC, registers, stack and a small control block are all stored in the user process address space).
- 3> simple management (thread creation, thread switching, synchronization b/w the thread can be done without intervention of the kernel).
- 4> fast and efficient.

Disadv

- 1> There is a lack of co-ordination b/w thread and OS ~~calls~~ kernel.
- 2> ULT require no blocking system calls in multithread kernel.

Kernel level thread

- 1> In this method kernel know about and manage thread.
- 2> Instead of thread table in each process the kernel has a thread table that keep track of all the thread in the system.
- 3> OS kernel provide system call to manage and create thread.

Adv

- 1) kernel level threads are especially good for appⁿ that frequently block.
- 2) Because kernel has full knowledge of all the threads, scheduler may decide to give more time to a process having large no. of threads than the process having smaller no. of threads.

Disadv

- 1) slow and inefficient (here kernel manages all the threads using thread control block and also schedules the thread).

Types of communication in IPC

- 1) Direct
- 2) Indirect

→ In direct communication the process may reach other (i.e. the process can be identified with a unique name).

Before start of the commⁿ process ~~are~~ must establish a commⁿ link with the another process.

→ there exist only a single link b/w the processes, and the link can be unidirectional or bidirectional.

Indirect

In this commⁿ messages are directed and received from mailbox (also referred to as port no.).

Each mailbox has a unique id and only communicate with the process if the mailbox is shared among the process.

Steps for establishing a connection

- link established only if processes share a common mailbox
- establish a physical link b/w the processes via mailbox
- Each pair of processes may share several commⁿ links if there exist more than 1 mailbox. Link may be unidirectional or bidirectional.

Synchronization

commⁿ b/w the process takes place by call to send and receive messages.

message passing may be either blocking or non-blocking. Here the sync. helps to co-ordinate b/w the processes to deliver messages to the respective shared memory and mailbox. Sync. also helps to avoid data inconsistency.

Blocking Send ⇒ the sending process is blocked until the msg is received by the receiving process or the mailbox.

Non-blocking send ⇒ the sending process send the msg and resume.

Blocking received ⇒ The ~~receiving~~ ~~block~~ receiving process blocks until a msg is available.

Non-blocking received ⇒ The receiver retrieves either a valid msg or null.

Buffer $\rightarrow$ In interprocess commⁿ the msg exchanged b/w the processes are stored in temporary storage known as buffer. Buffer can be of following types.

① capacity buffer - Here the buffer size is ① if msg^s as ~~the sender~~ soon as sender send the receiver must read it else other process are in block state.

Bounded buffer $\rightarrow$ Here the buffer has finite length and 'n'. n no. of messages can be sent and received by the processes. If the queue is full none of the process can send their data and if queue is empty then no receive operation can be performed.

Unbounded buffer $\rightarrow$ The buffer has ∞ no. of size.

Process Synchronization - ~~Process~~

Race condⁿ $\rightarrow$ when several process access and manipulate the same data concurrently and the o/p of execution depends on the particular order in which access takes place. is called race condition.

Producer - consumer Problem

↓
produce data

0	A
1	B
2	C
3	D
4	
5	