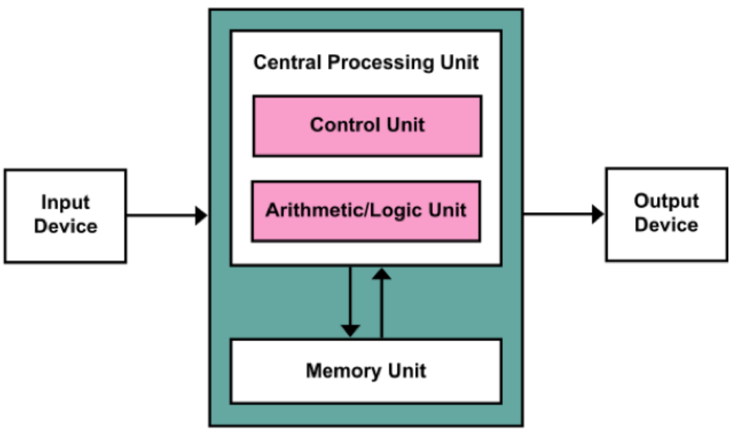
Operating systems

- Introduction :

*Features of an operating system : Design principle

- Multiprocessor } Design technique
- Multithread }
- Single/multicore → multicore is used for max^m eff.
- Centralised (Eg. resorts) → Autentication
- Distributed (Eg. Satellite kitchen) → Scalability

* Neumann Architecture : (Diagram - X.)



* Operating system :

- OS (system software) is a mandatory software that enables interactions between user and the hardware.

- Software : set of instructions written in specific programming language

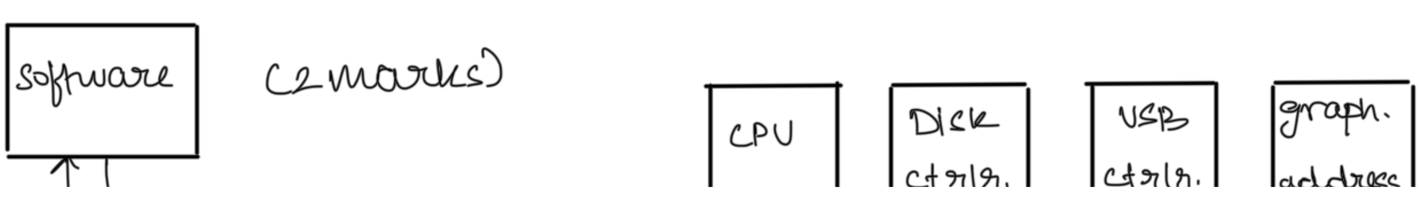
Two softwares :

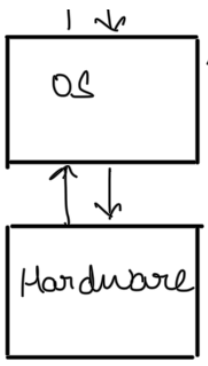
Application software → dedicated code to make computer active (Eg. calculator)

System software → Mandatory to make system functional or operational.

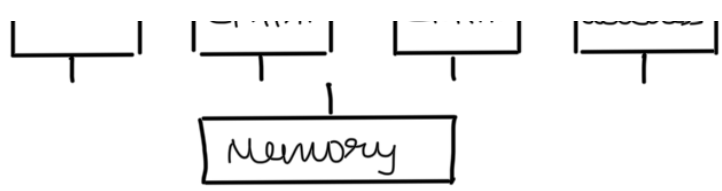
*Important features : Availability, Integrity, Security.

* (2 marks)





(Middle person)



* CUI: Command user interface ; GUI : Graphic User Interface

* 

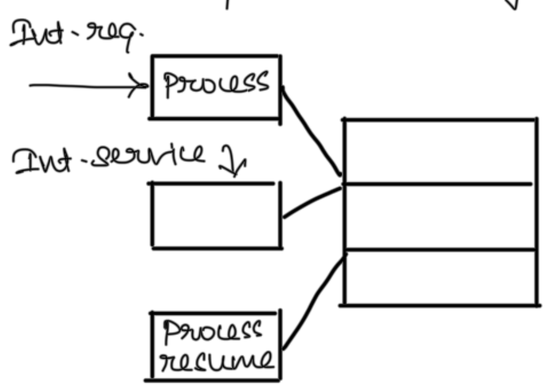
System call } S/W
Interrupts
Hardware

→ Interrupt request line and interrupt handler routine.

and Process communication types//

* Control, Data & Address lines

* Interrupt handling diagram.



Entails how the interrupts are managed (based upon relative priority)

* Memory Heirarchy:

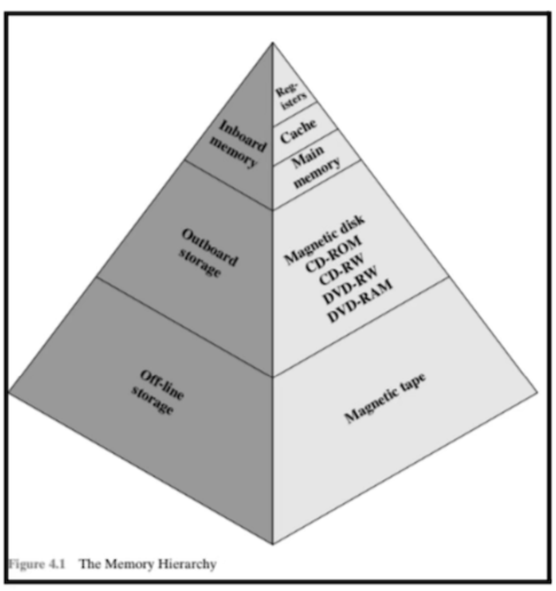


Figure 4.1 The Memory Hierarchy

} volatile

Difference between volatile and non volatile memory

RAM, ROM, SRAM and DRAM

* Important keywords:

Memory access, process driven, interrupt driven, DMA, Single process, multiprocess, time management, process management, Request, Service.

Process management, memory management, mass storage, file system, cache, I/O. → Resource mgmt.

* Functions of OS:

- Resource management
- Process management
- Create and delete
- Schedule threads
- Suspend and resume
- Process synchronization
- Process communication
- Memory management
- Mode of process creation
- Mode of execution
- Mode of comn and synn
- Mode of deployment & scalability
- Services:
  - User interface
  - program execution
  - I/O operation
  - File system management
  - Communication
  - Error detection

- File system management
- Mass storage management
- Cache management
- I/O system management
- Security and protection
- Vizualization
- Distributed systems
- Display and power options
- System view:
  - Resource allocation
  - Logging
  - Protection & security
  - Communal interpretation
  - GUI
  - System call & API
- Process execution
- Process sync
- TTL - Time to live
- Arriving time/comp.
- Wait time

* Process: id, time and priority

Algorithms: FCFS and SJF : Non preemptive
Priority & Round Robin } : Preemptive   10M

Note: Linux systems are generally safe from viruses as they do not allow self execution.

FCFS: when processes have similar burst time
SJF: when there are many "small" tasks
↳ problem of starvation.

① The Gaant Chart - 15M / 10M ↓

Process ID	Burst Time	Priority
P1	6	5
P2	4	1
P3	2	2
P4	6	4
P5	10	3

If no priority is mentioned, assume equal priority (as in round robin)

Gantt chart:

0	6	10	12	18	28
P1	P2	P3	P4	P5	

FIFO

0	2	6	12	18	28
P3	P2	P1	P4	P5	

SJF

0	4	6	12	22	28
P2	P3	P4	P5	P1	

Priority

0	3	6	8	11	14	17	18	21	24	27	28
P1	P2	P3	P4	P5	P1	P2	P4	P5	P5	P5	

RR

Note:
If priority and burst time are same, "oldest" process is considered
→ It has less ttl (time to live)

	R1	R2	R3	R4
P1	3-3	3-0	—	—
P2	3-1	1-0	—	-
P3	2-0	-	-	—
P4	3-3	3-0	—	-
P5	3-7	3-4	3-1	1-0

	w	e
P1:	14-3=11	17-0=17
P2:	3+17-6=14	18-3=15
P3:	6	8-6=2
P4:	8+7=15	21-8=13
P5:	11+10=21	28-11=17

	w / e	P1	P2	P3	P4	P5
FIFO		0	6	10	12	18
		6	10	12	18	28
SJF		6	2	0	12	18
		12	6	2	18	28
Priority		22	0	4	6	16
		28	4	6	16	22
RR		11	14	6	15	21
		17	15	2	13	17

(i) FIFO:
AWT: 9.2
AET: 14.8

(ii) SJF:
AWT: 7.6
AET: 13.2

(iii) Priority:
AWT: 9.6
AET: 15.2

(iv) RR:
AWT: 13.4
AET: 12.8

② Consider the following process:

Process	Burst	Arrival
P1	8	0
P2	4	1
P3	9	2
P4	5	3
P5	3	4

Calculate wait time and completion time for
(i) SJF
(ii) FCFS
(iii) RR → Quanta → 2.

(i) FCFS

	A	B	S	W	TA
P1	0	8	0	0	8
P2	1	4	8	7	11
P3	2	9	12	10	19
P4	3	5	21	18	23
P5	4	3	26	22	25

(W = S − A)
(TA = W + B)
W = 57
AW = 11.4
TA = 86
ATA = 17.2

Gantt:

0–8	8–12	12–21	21–26	26–29
P1	P2	P3	P4	P5

(ii) SJF:

	A	B	S	W	TA
P1	0	8	0	0	8
P5	4	3	8	4	7
P2	1	4	11	10	14
P4	3	5	15	12	17
P3	2	9	20	18	27

W = 44
AW = 8.8
TA = 73
ATA = 14.6

0–8	8–11	11–15	15–20	20–29
P1	P5	P2	P4	P3

(iii) Round Robin → Quanta = 2

	R1	R2	R3	R4	R5
P1	2-6	2-4	2-2	2-0	–
P2	2-2	2-0	–	–	–
P3	2-7	2-5	2-3	2-1	1-0
P4	2-3	2-1	1-0	–	–
P5	2-1	1-0	–	–	–

0 2 4 6 8 10 12 14 16 18 19 21 23 24 26 28 29
P1 P2 P3 P4 P5 P1 P2 P3 P4 P5 P1 P3 P4 P1 P3 P3

→ Use ready Queue

Questions:

1. What does OS do
2. User vs. system
3. Modules of resource mgt.
4. Sys process commⁿ
5. Hierarchy, multithread vs. multiprocessor.

Operating system services:

- User interface
- Program execution
- I/O operations
- File system manipulation

- File system manipulation

- Communication

- Error detection

- Resource allocation and logging

User interface:

CUI and GUI, Touchscreen.

System calls:

File copy:
get input name
write prompt to screen
accept input
Acquire target name
Accept input

(R and W) execute too

Open file in 'R' mode:

- if file does not exist ⇒ exit

Open target file 'W' mode:

- if file exists, abort loop

- Read from input file,

10 MK

- Until read, fail

- Close O/P file

- Work completion message in screen

- Terminate process

Application Programming Interface (API):

- Create process

- Instances (a.txt)

- RTE: Run Time Environment

Types of system calls:

- Process control: Create and terminate process
  load and execute
  get process attributes
  wait and signal
  allocate and free memory

File management:

- Create and delete
- open and close
- Read, write, reposition
- get and set file attributes

Device Management:

- Request and release
- Read, write, reposition
- get and set dev. attributes
- Attach and detach logical device.

Information Maintenance:

- get and set time and system data
- get and set device and process attributes
- shared message and message priority
- communication
- create and schedule messages, communication
- send and receive.

- transfer status is formatted
- attach and detach logical device
  ↳ reusable

System services:

- status in function
- file management
- file modification
- program language support
- program load and execution.

Protection:

- Get and set file and device permission.

Communication and background services:

- demon application program

Note:

9-bit protection mechanism:

Here, r = 4, w = 2, e = 1
↳ 2+1 = 3 ⇒ write + execute

O	G	E
rwe	rrr	rrr
7	4	4

MOST COMMON : 741

31/8/24

1) Types of system calls
Process Control
File management
Device management
Information maintenance
Communication
protection

2) System services:
File management
Status information
File mediation
Programming language support
Program loading & execution
Communication
Background services - App program & Deamons

(i) Linker
(ii) Loader < Absolute / Relocatable
(iii) Assembler
(iv) Translator / Interpreter
(v) Compiler

OS Design goals
user goals and system goals

Operating System Structure.

Monolithic

user

Shell commands and

System calls interface to Kernel

user

Kernel

Signal
File sys.

CPU scheduling
Terminal handling

Swapping block I/o
page replacement

Kernel interface to hardware

Hardware

Terminal controlled | device controller | Memory controller

↑
①

Layered approach:

application

Library function

System call	CPU scheduling
filesystem network	memory manager
block device	director devices

device driver

↓ ↓ ↓ ↓

Hardware

loosely coupled

tightly coupled —①

Micro Kernels.

Interpreter — memory — CPU scheduling

modules — Key
Loadable Kernel modules
Hybrid system – MacOS and IOS
User experience layer
Application processor,
Core framework Kernel

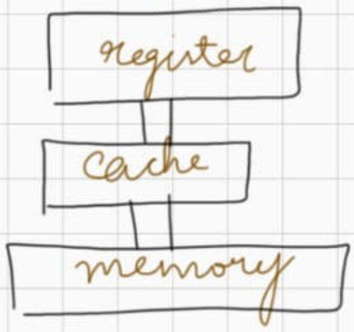
Android
Kernel abstraction and Kernel Expansion

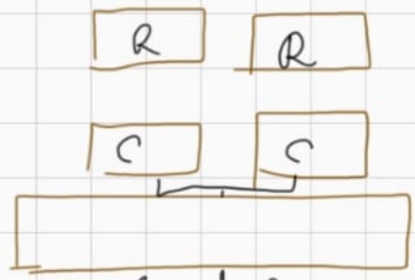
API and
Android Runtime – ART
Boot start

Date. 6/9/24

Single Core and multi core

→ user view
→ system view
→ multiprocess
multiprogram





Dual Core

Operating System	
Job 1	—
Job 2	Job 2
Job 3	Job 4
Job 4	Job 3
	Job 5

→ Time sharing hardware
→ swapping
→ logical and physical memory
→ system → user
↓ kernel

P, D, M, I/O } management

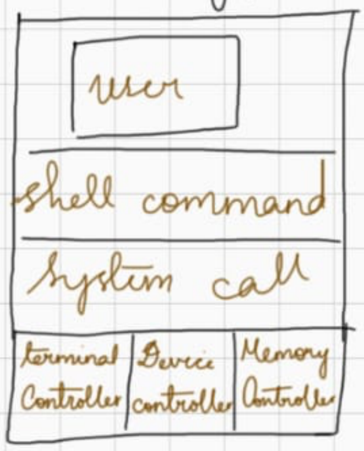
Operating System Services →
- UI (User interface)
- Program execution
- file system manipulation
- Communication
- Error Detection

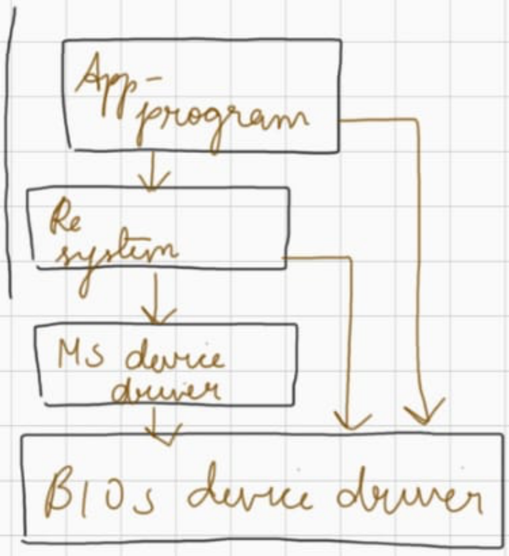
User operation
- command interpreter
- graphical user interface

Communication, Information — System calls →
- Process control
- file management
- Device management

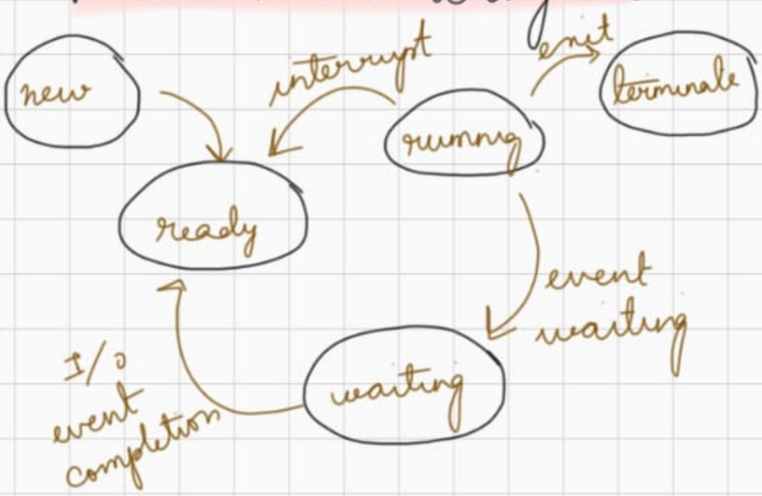
* Design Goals
* Mechanism and policies
* OS structure

Layered approach





Process State Diagram



- New
- Running
- Waiting
- Ready
- Terminated

PCB - Process control block

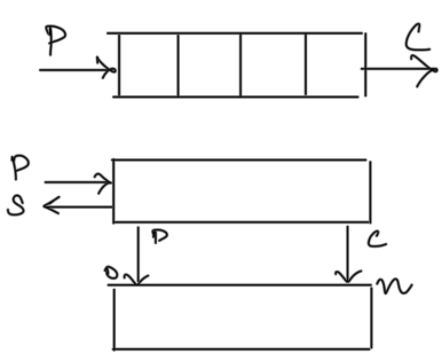
Process state
Process
Process cache
registers
memory lil
list of

* Process state
* Program control
* CPU registers
* CPU selection info
* mem management
* account information
* I/O state information

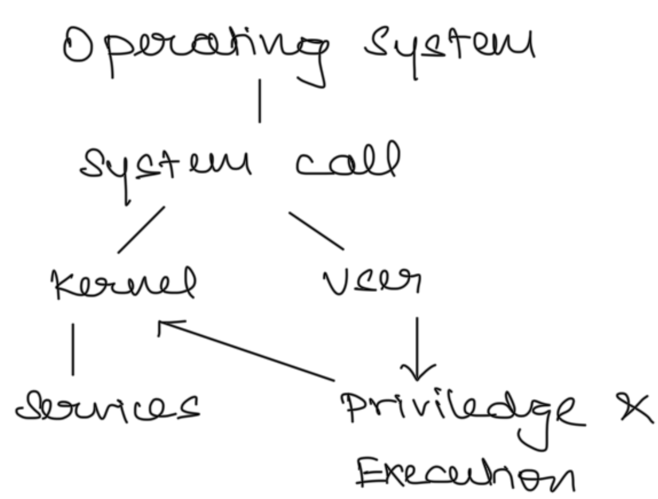
- thread
- scheduling queue
- FCFS, SJF, PS, RR - scheduler
- PID and PPID
- Long time, short time Job scheduler, cpu
- I/O bound process, CPU bound process
- mid scheduler / swapping sheduler
- content switching
- process termination

Operating System:

- system design and trade off
- Process model
- schedular organisation
- Process scheduling
- Pre-emptive and non-premptive
- Scheduling Algorithm: FCFS, SJF, Priority, RR
- Process co-operation
- Inter process communication (IPC)
- Process synchronisation
- synchronisation issues.
- Critical section
- Mutual exclusion
- Producer - Consumer
- Dining Philosopher Problem
- Array → produce, consume
- Race condition, semaphores



O → Empty ; n → Full



Semaphores

- Semaphore proposed by Edsger Dijkstra, is a technique to manage concurrent processes by using a simple integer value, which is known as a semaphore.

- Semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.
Helps in IPC
- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait () and signal ().

wait () → P [from the Dutch word **proberen**, which means "**to test**"]

signal () → V [from the Dutch word **verhogen**, which means "**to increment**"]

* Definition of wait()

Tests if any other process is using a resource

```
P (semaphore S)
{
 while (S<=0);
 S--;
}
```

* Definition of signal()

```
V (semaphore S)
{
 S++;   // Basically, resource is available @ this pt.
}
```

Note: Semaphore is common betⁿ processes.

All the modifications to the integer value of the semaphore in the wait () and signal() operations must be executed indivisibly.
That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Types:

(1) Binary / Mutex semaphores (Mutual Exclusion).

0 → process has to wait
1 → process can access resource

① Initial value is 1 → passes thro' while → changes to 0

② Another process comes up → here S is 0 →

process is stuck in while loop → "waits".

③ Upon exiting, first process calls signal fn.
→ S=1 @ this pt.

④ The 2nd process will break loop and execute now

(ii) Counting Semaphore:

- Unrestricted domain
- ctrl. access to a resource that has multiple instances

① say, a resource can be accessed by 2 resources.
↳ S=2
② First process enters ⇒ S=1
③ Second process enters ⇒ S=0
↓
when 3rd one enters, it has to wait.

- Process control blocks
↓
Process state
Program counter
CPU Register
CPU scheduling Information
Memory management Information
Accounting info
I/O status

- Threads
- I/O Bound Process
- CPU Bound Process
- Scheduling Queue
- CPU scheduler
- context switching
- Hardware support
- Process creation
- Fork and Pipeline

Process → correction, term.
Cascade termination
Zombie process
Fdfs & sjf.

* Service
|
Empty → Process → visible
foreground Background

IPC and Information sharing
Computational speedup
Modularity
Shared memory
Merge passing

Send, receive
Synchronisation
Blocking send
Non-blocking send
Block / N-B Receive

* Buffering:

Pipes - parent - child
↳ Generate and name pipes.

Client server communication, Socket and server socket

RPC: Remote productive calls

* Multiple programming → Data and task parallelism

Thread model: One to one, many to one
many to many.

* CPU Scheduling Algorithms:
↳
Can be pre-emptive / non-preemptive

Infinite wait → starvation and ageing of process.

* Other methods:

- Multilevel Queue Scheduling
- Realtime process
- System process
- Interface & batch process
- Multilevel feedback

* Load Balancing Scheduling
Push and pull migration
Process affinity: soft and hard

* Process synchronisation
Multiple - process, limited resource
Critical section process
Mutual exclusion, Binded waiting.

* Hardware support: strong / weak
  Priority inversion method

  Process scheduling → Memory access → Memory
  available → Request / Register.

* Addressing Binding
  Compile time
  Local time
  Execution time

  logical, physical and virtual.

  Memory allocation, dynamic allocation,
  Best / first / worst fit.

  Internal / External factors

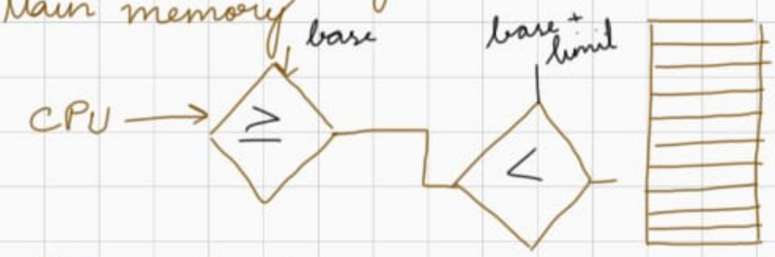
Date. 19/9/24

Process synchronisation
Process and resources
I/O + memory
Sender/receiver
Buffer size

The critical section
Mutual exclusion
Progress
Bounded wait
Pre-emptive kernel and
non-pre emptive kernel
Peterson Solution
memory barrier

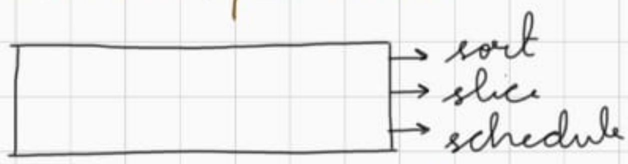
Mutex locks
monitor usage
signal and wait
signal and continue
Infinite wait (deadlock)
Synchronisation Problems
-bounded buffer problem
-Reader writer problem

Dening Philosopher problem
Concurrency control problem
Deadlock handling
Main memory

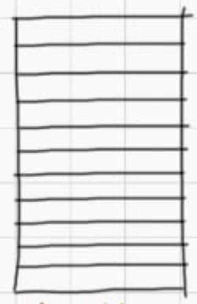


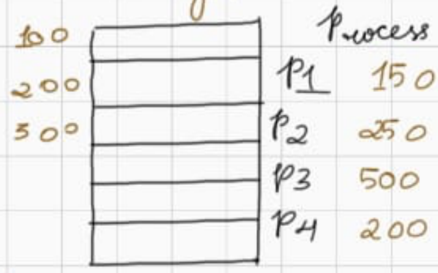
Address binding
Logical vs Physical vs Virtual
Dynamic loading
Dynamic linking and shared library
Load, link, compile, execute

Contiguous memory allocation
Memory Reallocation
Variable particular



Memory mem





best fit, worst fit,
first fit

Request → use →
Deadlock condition
Mutual exclusive
Hold and wait
No of exemption
Circular wait
Resource allocation graph

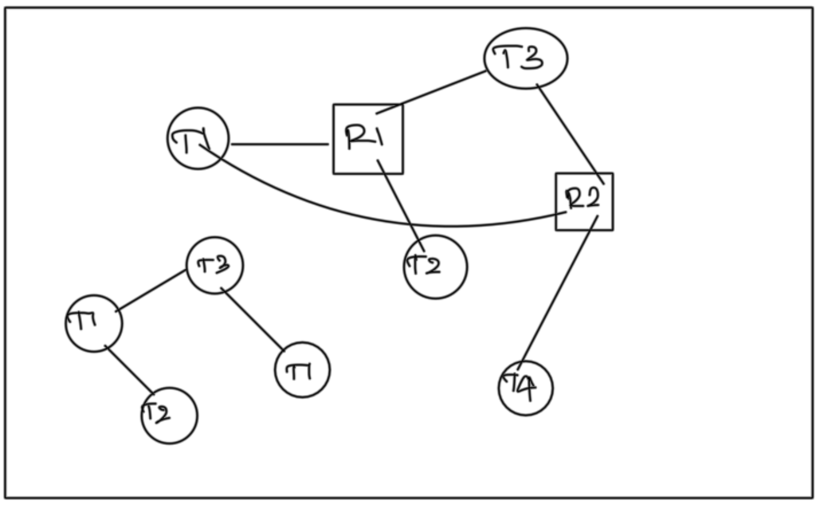
→ Request
$T_1 \to R_1 \to T_2 \to R_3 \to T_3 \to R_2 \to T_1$
$T_2 \to R_3 \to T_3 \to R_2 \to T_2$

E. $T_1 \to R_1, T_2 \to R_3, R\text{-}T_2, R_2\text{-}T_2, R_3\text{-}T_3$
MAC - Machine Authentication code
MAC address - Physical

T1 → R1 → T2

R1 → T3 → R2 → T1

R2 → T1 → T2 → R2



Qn: Consider the following set of processes with the following:

Pid.	Priority	Burst	Arrival
P1	40	20	0
P2	30	25	25
P3	30	25	30
P4	35	15	60
P5	5	10	100
P6	10	10	105
pidle	0		

Here, higher number indicates higher priority.

Time quantum size is 10 units.

If a process is pre-empted, it is placed at the end of a queue.

Sketch gantt chart in RR

Find waiting and arenue time

Sketch the process state diagram and explain it progressive path.

0 10 20 25 35 45 55 60 70 80 85 90 95 100 105 112 120
P1 P1 P5 P2 P3 P2 P3 P4 P3 P2 P4 P3 P5 P5 P0 P5

} priority has high value (write so)

0-10	P1	
10-20	P1	
20-25	P5	
	← P2	
25-35	P2	(15)
35-45	P2	(15)
45-55	P2	(5)
55-60	P3	(10)
60-70	P4	(5)
70-80	P3	(0)
80-85	P2	⟶
85-90	P4	

Portions:

Apply: sys call and scheduling, process state

Deadlock detection:

- How often deadlock occurs
- How many threads are involved.
- How many resources are in shortage.

Recovery:

- Abort all deadlock processes.

- Abort one process at a time till deadlock cycle RR loop is eliminated.

  ↳ Priority
  ↳ Level of completion
  ↳ No. of dependent processes.

- Resource preemption: select a victim, roll back and starvation.

Qn: which one of the six will lead to deadlock?
When it is not deadlocked, outline the sequence

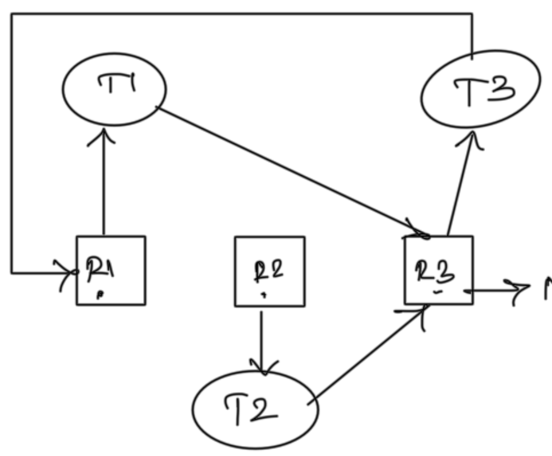
of thread execution.

Also, write the resource allocation and wait for execution of all six types.

Bankers' Algorithm ✗.

Wait-for graph: with arrows.
Resource allocation: without arrows.

□ → ○ Allocation
□ ← ○ Requires.



→ No. of dots = no. of resources.

T1 and T3 are in a circular dependency
T2 also depends on R3 ⇒ DEADLOCK.

PYQs:

(i) FCFS:

Burst time = Execution time.

P1	P2	P3	P4	P5

0 8 12 21 26 29

P	B/E	A	S	W = S − A	TAT = W + B
1	8	0	0	0	8
2	4	1	8	7	11
3	9	2	12	10	19
4	5	3	21	18	23
5	3	4	26	22	25

∴ Avg. wait time = 11.4 ms
Avg. TAT = 18 ms.

(ii) SJF

P	B	A	S	W = S−A	TAT = B + W

·	0	0	0	0	8
5	3	4	8	4	7
2	4	1	11	10	14
4	5	3	15	12	17
3	9	2	20	18	27

P1	P3	P2	P4	P5

0 8 11 15 20 29

∴ Avg. wait time = 8.8 ms
Avg. TAT = 14.6 ms

Various ways in which synchronisation problem can be solved.

Consider the following process scenario:

Process	(A, B, C, D) Allocation	(Demand) Max^m	= (M − A) Need	Common Available
P1	3014	5117	2103	0301
✓P2	2210	3211	1001	
✓P3	3121	3321	0200	
✓P4	0510	4612	4102	
P5	4212	6325	2113	

Banker's algorithm for the following: (10 M)

(i) Available: 0, 3, 0, 1

(ii) Available: 1, 0, 0, 2

Now, we shall plot Bankers' Algorithm and safe sequence.

Here,

work = | 0 | 3 | 0 | 1 |

finish [i] = false ∀ i = 1 ... 5

P3 → P2 → P4

finish [i] = false Need [i] <= work

1.
finish [i] = false ✓

Need [i] = | 2 | 1 | 0 | 3 |
work [i] = | 0 | 3 | 0 | 1 | ✗

2.
finish [i] = false ✓

Need[i] = 1 | 0 | 0 | 1
work [i] = 0 | 3 | 0 | 1 ✗

3. finish [i] = false ✓

Need [i] = 0 | 2 | 0 | 0
work [i] = 0 | 3 | 0 | 1 ✓

finish [i] = true

work [i] = 3 | 4 | 2 | 2 (work = work + allocation)

4. finish [i] = false ✓

Need [i] = 4 | 1 | 0 | 2
work [i] = 3 | 4 | 2 | 2 ✗

5. finish [i] = false ✓

Need [i] = 2 | 1 | 1 | 3
work [i] = 3 | 4 | 2 | 2 ✗

1. finish [i] = false ✓

Need [i] = 2 | 1 | 0 | 3
work [i] = 3 | 4 | 2 | 2 ✗

2. finish [i] = false ✓

Need [i] = 3 | 2 | 1 | 1
work [i] = 3 | 4 | 2 | 2 ✓

finish [i] = true

work [i] = 5 | 6 | 3 | 2

4. finish [i] = false ✓

Need [i] = 4 | 1 | 0 | 2
work [i] = 5 | 6 | 3 | 2

finish [i] = true

work [i] = 5 | 11 | 4 | 2

5. finish [i] = false ✓

Need [i] = | 2 | 1 | 1 | 3 |

Work [i] = | 5 | 11 | 4 | 2 | ×

1. finish [i] = false ✓

Need [i] = | 2 | 1 | 0 | 3 |

Work [i] = | 5 | 11 | 4 | 2 | ×

∴ Ans. : P3 → P2 → P4 → Deadlock
(P1, P5 are unsafe).

(iii) Given,

Need
2103
1001
0200
4102
2113

Work = | 1 | 0 | 0 | 2 |

finish [i] = false ∀ i = 1 ... 5

P2 → P3 → P4 → P5 → P1

1.
finish [i] = false ✓

Need [i] = | 2 | 1 | 0 | 3 |

Work [i] = | 1 | 0 | 0 | 2 | ×

2.
finish [i] = false ✓

Need [i] = | 1 | 0 | 0 | 1 |

Work [i] = | 1 | 0 | 0 | 2 | ✓

finish [i] = true

Work [i] = | 3 | 2 | 1 | 2 |

3.
finish [i] = false ✓

Need [i] = | 0 | 2 | 0 | 0 |

Work [i] = | 3 | 2 | 1 | 2 | ✓

finish [i] = true

Work [i] = | 6 | 3 | 3 | 3 |

4.
finish [i] = false ✓

Need [i] = | 4 | 1 | 0 | 2 |

work [i] = | 6 | 3 | 3 | 3 | ✓

finish [i] = true

work [i] = | 6 | 8 | 4 | 3 |

5.

finish [i] = false ✓

Need [i] = | 2 | 1 | 1 | 3 |

work [i] = | 6 | 8 | 4 | 3 |

finish [i] = true

work [i] = | 10 | 10 | 5 | 5 |

1.

finish [i] = false

Need [i] = | 2 | 1 | 0 | 3 |

work [i] = | 10 | 10 | 5 | 5 |

∴ Ans: P2 → P3 → P4 → P5 → P1

Request grant:

(i) $Request_i <= Need_i$

(ii) $Request_i <= Available_i$

Process:

$Available = Available - Request_i$

$Allocation_i = Allocation + Request_i$

$Need_i = Need_i - Request_i$

Process	Allocation	Max	Avail
P1	2 0 0 1	4 2 1 2	3 3 2 1
P2	3 1 2 1	5 2 5 2	
P3	2 1 0 3	2 3 1 6	
P4	1 3 1 2	1 4 2 4	
P5	1 4 3 2	3 6 6 5	

Resource-Request: 1D Array

Request < Need

Request < Available

if satisfied:

↓ if satisfied

Available = Available − Request$_i$
Allocation$_i$ = Allocation + Request$_i$
Need$_i$ = Need$_i$ − Request$_i$

Process	Allocation A B C D	Max A B C D	Need A B C D	Avail. A B C D
P1	2 0 0 1	4 2 1 2	2 2 1 1	3 3 2 1
P2	3 1 2 1	5 2 5 2	2 1 3 1	
P3	2 1 0 3	2 3 1 6	0 2 1 3	
P4	1 3 1 2	1 4 2 4	0 1 1 2	
P5	1 4 3 2	3 6 6 5	2 2 3 3	

finish [i] = false ∀ p = 1, 2, ... 5

work 3 3 2 1

① Need <= work:

work = work + allocation

finish [1] = true

work: 3 3 2 1
+ 2 0 0 1
= 5 3 2 2

P1 → P4 → P5 → P2 → P3

② Not satisfied

③ NO

④ finish [4] = true

work: 5 3 2 2
+ 1 3 1 2
= 6 6 3 4

⑤ work: 6 6 3 4
+ 1 4 3 2
= 7 10 6 6

② work: 7 10 6 6
+ 3 1 2 1
= 10 11 8 7

③ work: 10 11 8 7
+ 2 1 0 3

12 12 8 10

Request- checking:

P1: 1 1 0 0

Process	Allocation A B C D	Max A B C D	Need A B C D	Avail
P1	~~2~~ 1 0 1	4 2 1 2	1 1 1 1	~~3~~ ~~3~~ 2 1
P2	3 1 2 1	5 2 5 2	2 1 3 1	- 1 1 0 0
P3	2 1 0 3	2 3 1 6	0 2 1 3	2 2 2 1
P4	1 3 1 2	1 4 2 4	0 1 1 2	
P5	1 4 3 2	3 6 6 5	2 2 3 3	

Safe sequence:

P1 → P4 → P5 → P2 → P3

(1) Work: 2 2 2 1
Need <= 1

work = 2 2 2 1
  3 1 0 1
  ---
  5 3 2 2

(2) NO

(3) NO

(4) work = 5 3 2 2
  1 3 1 2
  ---
  6 6 3 4

(5) Work = 6 6 3 4
  1 4 3 2
  ---
  7 10 6 6

(2) work: 7 10 6 6
  3 1 2 1
  ---
  10 11 8 7

(3) Work: 10 11 8 7
  2 1 0 3
  ---
  12 12 8 10

(ii) Not possible

Page Replacement Algorithm

FIFO, Optimal, LRU

(i) FIFO:

		3	3	3	3	3	4	4	4	4
	2	2	2	2	5	5	5	5	7	7
7	7	7	1	1	1	1	1	6	6	6
M	M	M	M	Hit	M	Hit	M	M	M	Hit

1	1	1	4	4	4	3	3	3	4	4	4
7	7	5	5	5	2	2	2	1	1	1	0
6	0	0	0	6	6	6	0	0	0	6	6
M	M	M	M	M	M	M	M	M	M	M	M

1
0
6
M

Page fault : 21

(ii) LRU:

		3	3	3	5	5	5	6	6	6
	2	2	2	2	2	2	4	4	4	4
7	7	7	1	1	1	3	3	3	7	7
M	M	M	M	Hit	M	M	M	M	M	Hit

6	0	0	0	6	6	6	0	0	0
1	1	1	4	4	4	3	3	3	4
7	7	5	5	5	2	2	2	1	1
M	M	M	M	M	M	M	M	M	M

6	6	6
4	4	1
1	0	0
M	M	M

Page fault : 22

(iii) Optimal:

		3	3	3	3	3	4	6	7	7
	2	2	2	2	5	5	5	5	5	5
7	7	7	1	1	1	1	1	1	1	1
M	M	M	M	Hit	M	Hit	M	M	M	Hit

7	0	0	0	0	0	0	0	0	0	0	0	0
5	5	5	4	6	2	3	3	3	4	6	6	6
1	1	1	1	1	1	1	1	1	1	1	1	1
Hit	M	Hit	M	M	M	M	Hit	Hit	M	M	Hit	Hit

24 - 9 = 15 Page faults.

Memory T

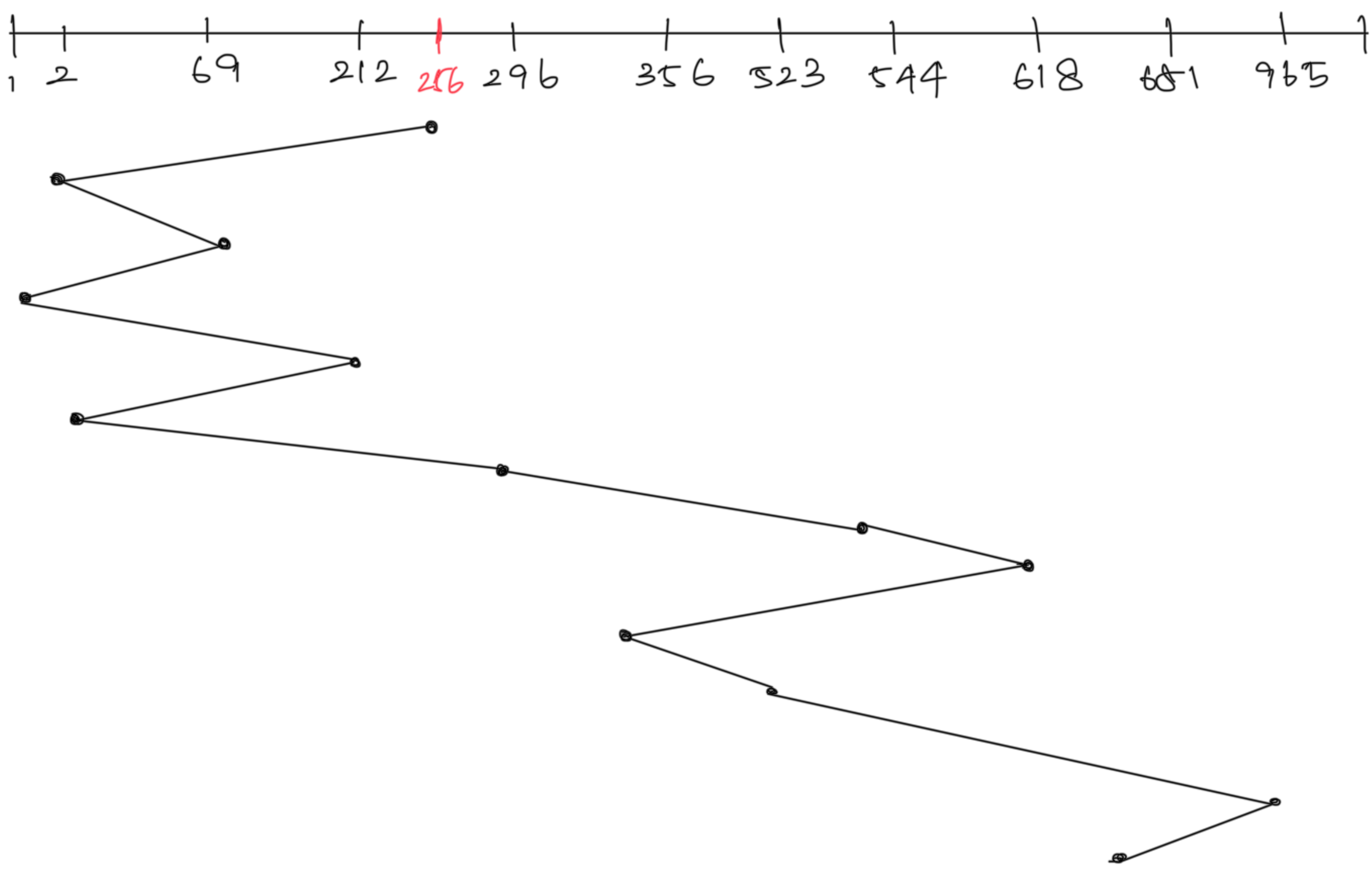
Time : (Hit ratio × Hit time) + (Miss ratio × Miss time)
52 102

TLB : Translation Lookaside Buffer

Total overhead movement → calc. first

Disc scheduling :

① FIFO

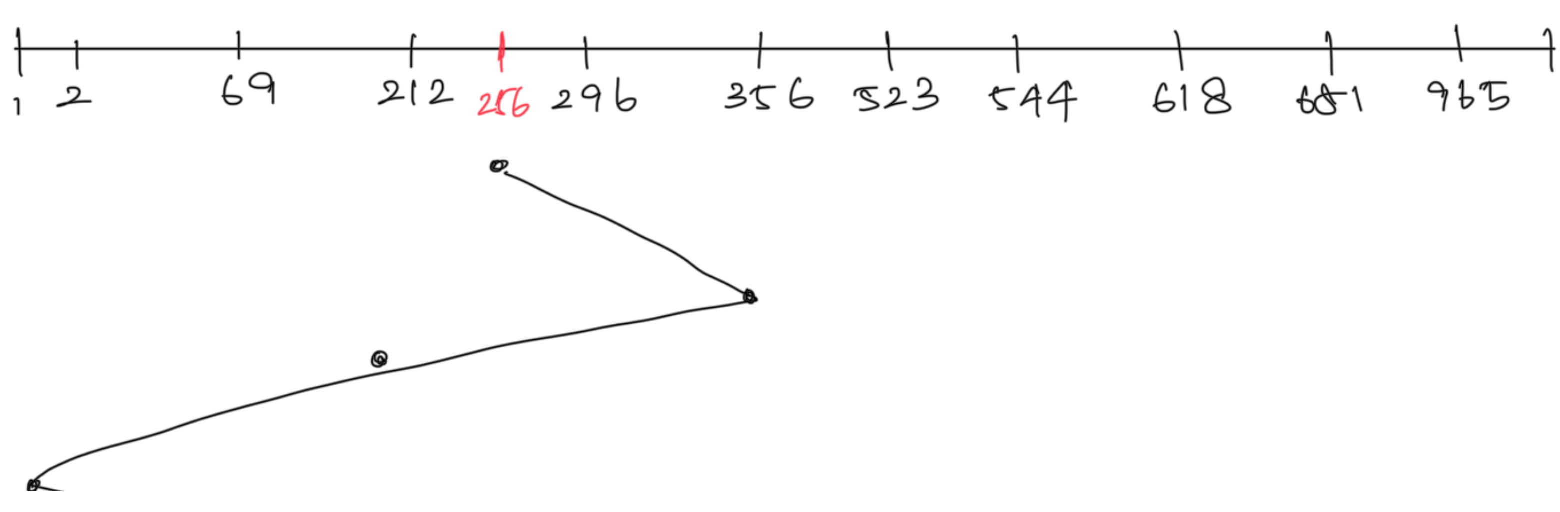


Total overhead movement :

254 + 67 + 68 + 211 + 210 + 294 + 248 + 74 + 262 + 167 + 442 + 284

= 2581

② SSTF



∴ Answer: 1419

③ SCAN:

(999 − 256) + (999 − 1) = 1741

④ CSCAN:

(999 − 256) + (999 − 0) + (212 − 0) = 1954

⑤ C Look:

(965 − 256) + (965 − 1) + (212 − 1) = 1884

⑥ Look:

(965 − 256) + (965 − 1) = 1673

Logical address → pg no., offset

↓ main memory

Phy. address. → frame no., offset