

Operating System (315319)

Model Question Paper — Set A · Model Answers

Q.1

a) Four services of operating system.

2 M

1. **Program execution** — loads a program into memory, runs it and ends it (normally or abnormally).
2. **I/O operations** — performs input/output for the user program (user cannot control devices directly).
3. **File-system manipulation** — create, delete, read, write, search and list files and directories.
4. **Communication** — exchange of information between processes (shared memory / message passing).
5. **Error detection**, resource allocation, accounting, protection and security.

(Any four, ½ M each)

b) Process and its states.

2 M

Process: A process is a program in execution. It is an active entity which has a program counter, stack, data section and the resources needed to run.

States of a process: New, Ready, Running, Waiting (Blocked), Terminated.

(Definition 1 M, states 1 M)

c) Throughput and waiting time.

2 M

i) Throughput: The number of processes that complete their execution per unit time (e.g. 10 processes/second). It should be maximum.

ii) Waiting time: The total time a process spends waiting in the ready queue. Waiting time = Turnaround time – Burst time. It should be minimum.

d) Thread and advantages of multithreading.

2 M

Thread: A thread is a lightweight process — the basic unit of CPU utilisation. It has its own thread ID, program counter, register set and stack, and shares code, data and open files with other threads of the same process.

Advantages: 1) Responsiveness — a program keeps running even if part of it is blocked. 2) Resource sharing and economy — threads share memory, so creating and switching threads is cheaper than processes. (Also: better use of multiprocessor architecture.)

e) Page fault.

2 M

Page fault: A page fault is an interrupt (trap) generated when a process tries to access a page that is **not currently present in main memory** (the page-table entry is marked invalid).

When it occurs: in demand paging, when the referenced page is still on the disk (backing store). The OS then brings the page from disk into a free frame (or replaces a page), updates the page table

and restarts the instruction.

f) Four attributes of a file.

2 M

1. **Name** — symbolic name of the file for the user.
2. **Identifier** — unique number (tag) that identifies the file within the file system.
3. **Type** — kind of file (text, executable, image ...).
4. **Location** — pointer to the device and location of the file on it.
5. **Size, Protection** (read/write/execute permissions), **Time, date and user identification**.

(Any four, ½ M each)

g) Linux vs Windows.

2 M

Linux	Windows
Open-source and free; source code is available.	Proprietary (Microsoft); paid licence, source code not available.
Command line (shell) is the main interface; GUI is optional.	Mainly GUI based; command prompt is secondary.
File names are case-sensitive; uses / as path separator.	File names are not case-sensitive; uses \ as path separator.
More secure; fewer viruses.	More prone to viruses and malware.

(Any two points, 1 M each)

Q.2

a) Real-time operating system.

4 M

A **real-time operating system (RTOS)** is an OS in which the correctness of a result depends not only on the logical result but also on the **time** at which it is produced. Every task has a fixed deadline and the OS must respond to events within that time limit. It is used when rigid time requirements are placed on the operation of a processor or the flow of data.

Types:

- **Hard real-time system:** deadlines must be met strictly; missing a deadline can cause failure or disaster. Secondary storage is limited, data is kept in ROM. Example: missile control, air-bag system.
- **Soft real-time system:** a critical task gets priority over other tasks, but missing a deadline only degrades performance. Example: multimedia, online video streaming.

Applications: industrial process control, air-traffic control, medical imaging systems, anti-lock braking (ABS), flight control, robotics.

(Explanation 2 M, types 1 M, two applications 1 M)

b) Types of system calls with examples.

4 M

Type of system call	Examples
1. Process control	fork(), exit(), wait(), exec(), kill(), load / execute

2. File management	open(), read(), write(), close(), create, delete
3. Device management	ioctl(), read(), write(), request / release device
4. Information maintenance	getpid(), alarm(), sleep(), time(), get/set system data
5. Communication	pipe(), shmget(), mmap(), send / receive message, create connection

(Types 2 M, examples 2 M)

c) Fixed (static) partitioning.

4 M

In **fixed partitioning**, main memory (other than the OS area) is divided into a number of partitions of **fixed size** at system start-up time. Partitions may be of equal or unequal size. Each partition can hold exactly **one process**, so the degree of multiprogramming is limited by the number of partitions. When a partition is free, a process from the input queue is loaded into it.

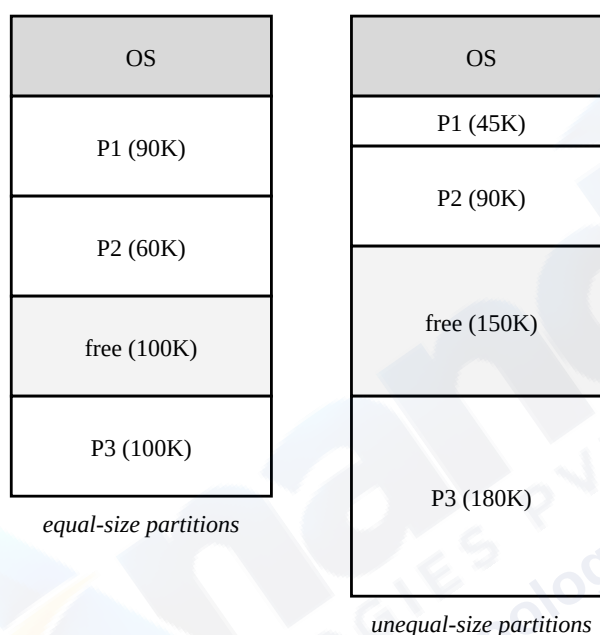


Fig.: Fixed partitioning of main memory

Disadvantages:

- **Internal fragmentation** — a small process in a large partition wastes the remaining space of that partition.
- Degree of multiprogramming is fixed (number of partitions).
- A process larger than the largest partition cannot be loaded.
- Memory utilisation is poor.

(Explanation 2 M, diagram 1 M, disadvantages 1 M)

d) Single-level directory structure.

4 M

In a **single-level directory**, all files of all users are kept in one directory. It is the simplest directory structure; every file is listed in the same directory and is accessed directly by its name.

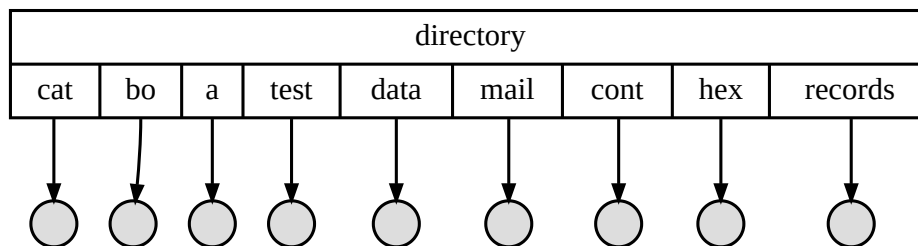


Fig.: Single-level directory

Advantages: 1) Simple to implement and understand. 2) Searching, creation and deletion of files is fast when the number of files is small.

Disadvantages: 1) **Naming problem** — all files must have unique names, two users cannot use the same file name. 2) **Grouping problem** — files cannot be grouped (no sub-directories); searching becomes slow when files are many. 3) No protection or privacy between users.

(Explanation 1 M, diagram 1 M, advantages 1 M, disadvantages 1 M)

Q.3

a) Process state diagram.

4 M

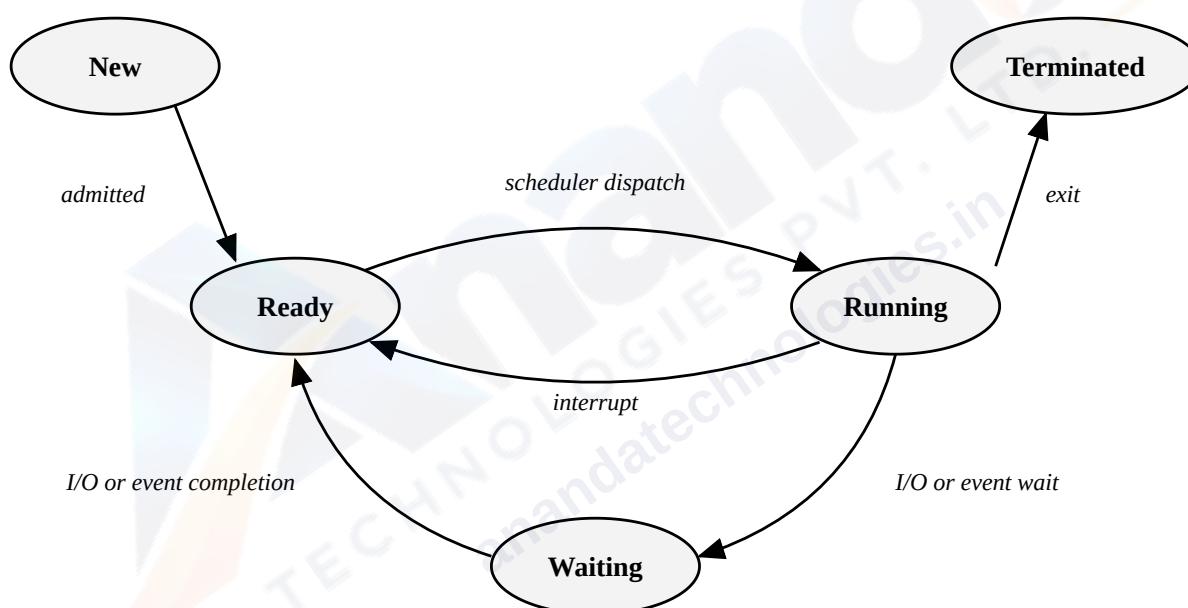


Fig.: Process state diagram (five-state model)

- **New:** the process is being created.
- **Ready:** the process is in main memory, waiting to be assigned to the CPU.
- **Running:** instructions of the process are being executed by the CPU.
- **Waiting (blocked):** the process is waiting for some event (I/O completion or a signal).
- **Terminated:** the process has finished execution.

Transitions: New → Ready (admitted by long-term scheduler); Ready → Running (dispatched by CPU scheduler); Running → Ready (interrupt / time slice over); Running → Waiting (I/O or event wait); Waiting → Ready (I/O or event completed); Running → Terminated (exit).

(Diagram 2 M, explanation 2 M)

b) One-to-one and many-to-many models.

4 M

1) One-to-one model: Each user thread is mapped to a separate kernel thread. If one thread makes a blocking system call, other threads can still run; threads can run in parallel on multiprocessors. Drawback: creating a user thread needs creating a kernel thread, so the number of threads may be limited. Example: Linux, Windows.

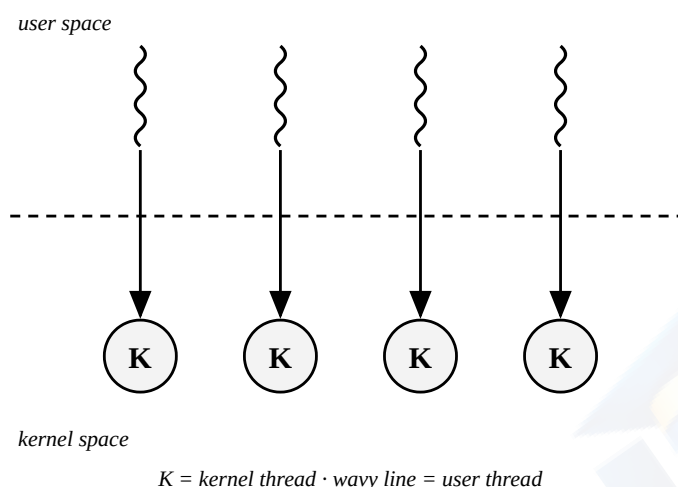


Fig.: One-to-one model

2) Many-to-many model: Many user threads are multiplexed onto a smaller or equal number of kernel threads. The user can create as many threads as needed; kernel threads run in parallel on a multiprocessor, and a blocking call does not block the whole process. Example: older Solaris, Windows with ThreadFiber package.

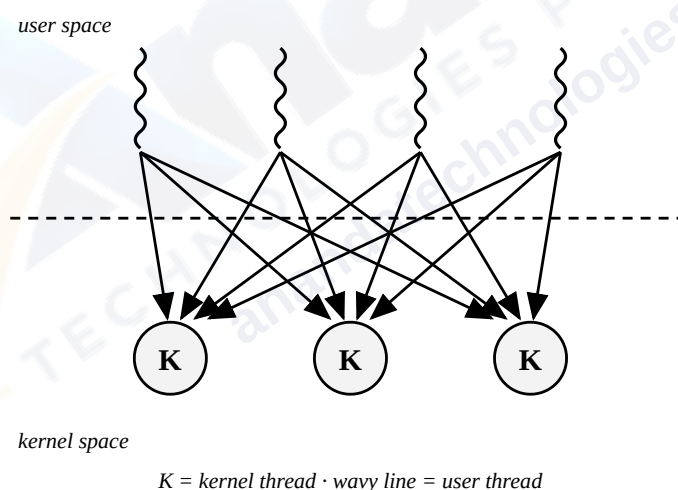


Fig.: Many-to-many model

(Each model with diagram 2 M)

c) Paging vs segmentation.

4 M

Paging	Segmentation
Memory is divided into fixed-size blocks called pages (logical) and frames (physical).	Memory is divided into variable-size blocks called segments.

Page size is decided by the hardware.	Segment size is decided by the user / compiler (logical units such as main, function, stack).
Logical address = page number + offset.	Logical address = segment number + offset.
Uses a page table (frame number for each page).	Uses a segment table (base and limit for each segment).
May cause internal fragmentation; no external fragmentation.	May cause external fragmentation; no internal fragmentation.
Invisible to the user (programmer).	Visible to the user; matches the user view of memory.

(Any four points, 1 M each)

d) Contiguous file allocation.

4 M

In **contiguous allocation**, each file occupies a set of **consecutive blocks** on the disk. The directory entry of a file stores its **starting block** and **length** (number of blocks). Both sequential and direct access are easy: block i of a file starting at b is block $b + i$.

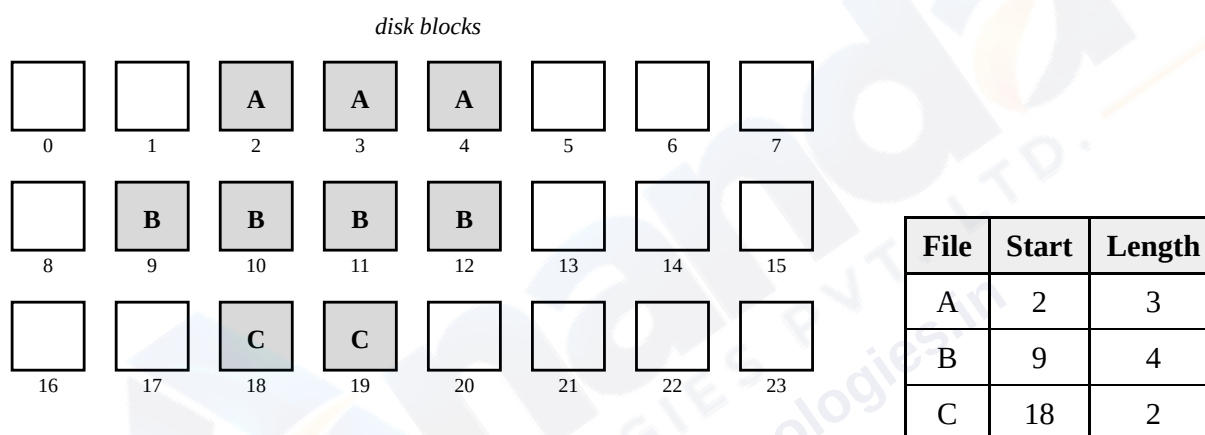


Fig.: Contiguous allocation with directory

Advantages: 1) Simple — only start and length are stored. 2) Supports sequential and direct access. 3) Fast access, minimum disk-head movement.

Disadvantages: 1) **External fragmentation** of free disk space. 2) File size must be known when the file is created. 3) A file cannot grow if the next blocks are occupied. 4) Compaction is needed, which is time-consuming.

(Explanation 1 M, diagram 1 M, advantages 1 M, disadvantages 1 M)

Q.4

a) First fit, best fit, worst fit (150K, 520K, 230K, 310K, 640K).

4 M

Partitions: 1 = 150K, 2 = 520K, 3 = 230K, 4 = 310K, 5 = 640K. Processes (in order): P1 = 225K, P2 = 480K, P3 = 140K, P4 = 600K. One process per partition.

First fit

Rule: allocate the first free partition that is big enough (search from the beginning).

Process	Partition allocated	Unused space (internal fragmentation)
---------	---------------------	---------------------------------------

P1 (225K)	Partition 2 (520K)	295K
P2 (480K)	Partition 5 (640K)	160K
P3 (140K)	Partition 1 (150K)	10K
P4 (600K)	Not allocated — must wait	—

Best fit

Rule: allocate the smallest free partition that is big enough (least leftover space).

Process	Partition allocated	Unused space (internal fragmentation)
P1 (225K)	Partition 3 (230K)	5K
P2 (480K)	Partition 2 (520K)	40K
P3 (140K)	Partition 1 (150K)	10K
P4 (600K)	Partition 5 (640K)	40K

Worst fit

Rule: allocate the largest free partition.

Process	Partition allocated	Unused space (internal fragmentation)
P1 (225K)	Partition 5 (640K)	415K
P2 (480K)	Partition 2 (520K)	40K
P3 (140K)	Partition 4 (310K)	170K
P4 (600K)	Not allocated — must wait	—

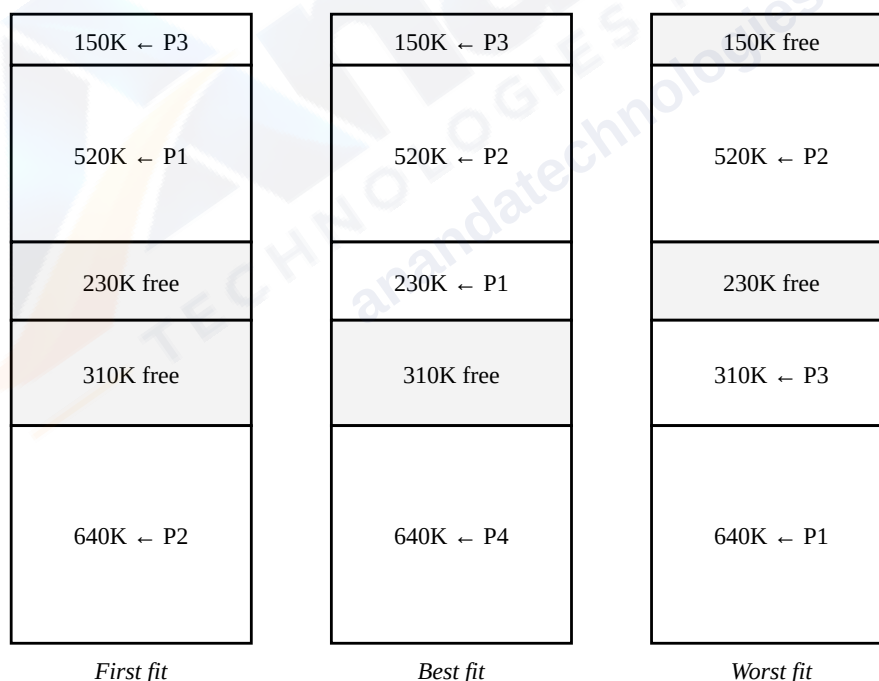


Fig.: Placement of processes (heights proportional to partition size)

Conclusion: Only **best fit** is able to allocate all four processes (P4 = 600K waits in first fit and worst fit). Hence best fit makes the most efficient use of memory here.

b) Necessary conditions for deadlock.

4 M

A deadlock can arise only if the following four conditions hold **simultaneously** (Coffman conditions):

1. **Mutual exclusion:** at least one resource must be held in a non-sharable mode — only one process can use it at a time.
2. **Hold and wait:** a process is holding at least one resource and is waiting to acquire additional resources held by other processes.
3. **No preemption:** a resource cannot be taken away forcibly; it is released only voluntarily by the process holding it, after the process completes its task.
4. **Circular wait:** a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes exists such that P_0 waits for a resource held by P_1 , P_1 waits for P_2 , ..., and P_n waits for P_0 .

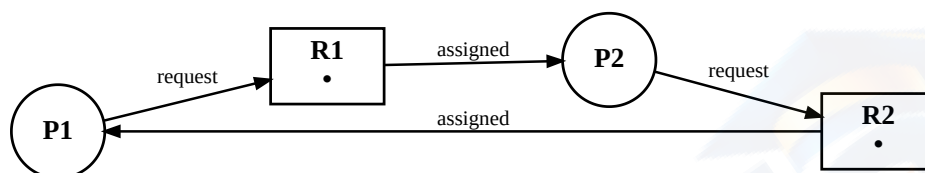


Fig.: Resource allocation graph with a deadlock (cycle $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_1$)

(Each condition 1 M)

c) Short-term and medium-term schedulers.

4 M

Short-term scheduler (CPU scheduler): selects one process from the **ready queue** and allocates the CPU to it (through the dispatcher). It runs very frequently (every few milliseconds), so it must be very fast. It decides which ready process runs next.

Medium-term scheduler: performs **swapping**. It removes (swaps out) a partially executed process from main memory to the disk to reduce the degree of multiprogramming or to free memory, and later swaps it back in so that it continues from where it stopped. It improves the process mix.

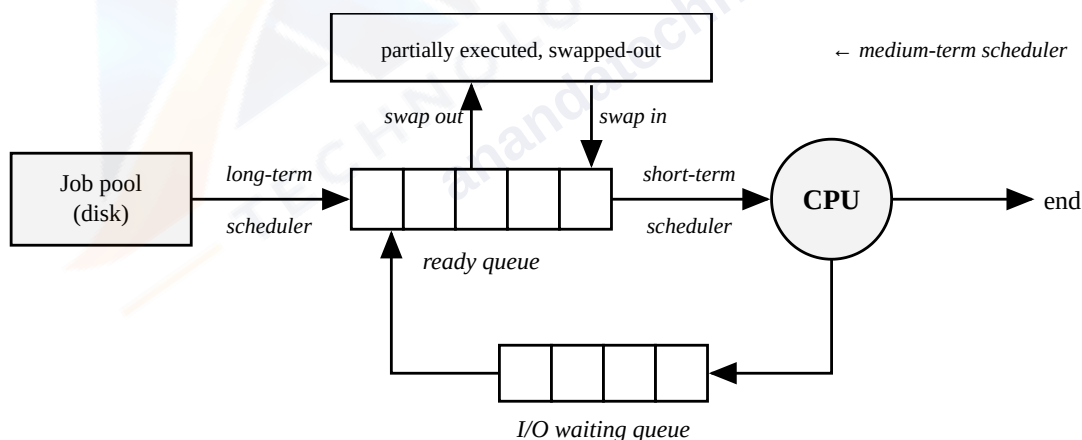


Fig.: Queueing diagram showing long-term, short-term and medium-term schedulers

(Short-term 1½ M, medium-term 1½ M, diagram 1 M)

d) Linked-list free-space management.

4 M

In this method, all **free disk blocks are linked together**. A pointer to the first free block is kept in a special location on the disk and cached in memory. Each free block contains a pointer to the next

free block. When a block is needed, the first block of the list is allocated and the head pointer moves to the next free block.

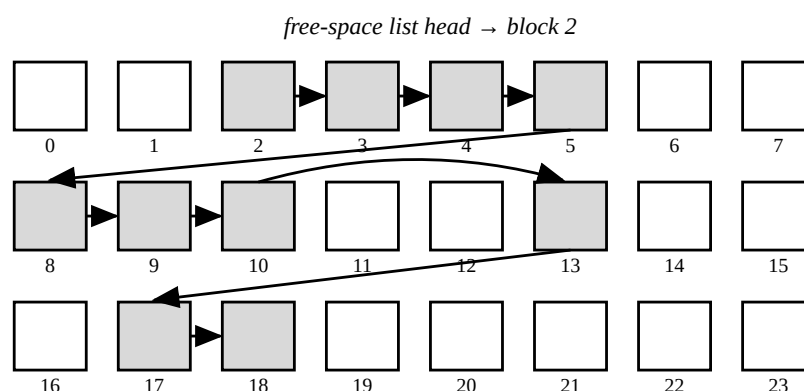


Fig.: Linked free-space list on disk (shaded = free blocks)

Advantages: no extra space is wasted (unlike a bitmap); easy to get one free block. **Disadvantage:** to traverse the list each block must be read from disk, which needs a lot of I/O time; finding contiguous free space is difficult.

(Explanation 2 M, diagram 2 M)

e) Process Control Block (PCB).

4 M

Each process is represented in the OS by a **Process Control Block (PCB)**, also called task control block. It is a data structure that stores all information about the process. The PCB is saved and reloaded during a context switch.

Pointer	Process state
	Process number (PID)
	Program counter
	CPU registers
	CPU scheduling information
	Memory-management information
	Accounting information
	I/O status information (list of open files)
	...

Fig.: Process Control Block (PCB)

- **Pointer:** links the PCB to the next PCB in a scheduling queue.
- **Process state:** new, ready, running, waiting, terminated.
- **Process number:** unique process ID (PID).
- **Program counter:** address of the next instruction to be executed.
- **CPU registers:** accumulator, index registers, stack pointer etc. (saved on interrupt).
- **CPU scheduling information:** priority, pointers to scheduling queues.

- **Memory-management information:** base and limit registers, page / segment tables.
- **Accounting information:** CPU time used, time limits, account numbers.
- **I/O status information:** list of I/O devices allocated, list of open files.

(Diagram 2 M, explanation 2 M)

Q.5

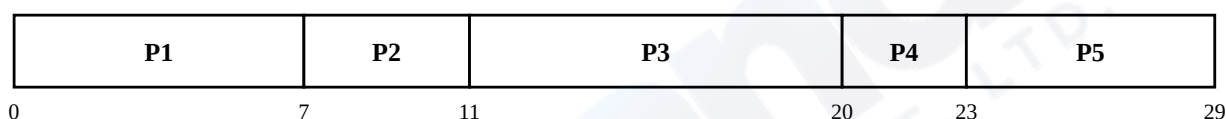
a) FCFS, SJF and RR ($q = 3$) for P1–P5.

6 M

All processes arrive at time 0, so TAT = Completion time and WT = TAT – Burst time.

Process	Burst time
P1	7
P2	4
P3	9
P4	3
P5	6

FCFS (First Come First Served)



Gantt chart

Process	BT	Completion time (CT)	TAT = CT	WT = TAT – BT
P1	7	7	7	$7 - 7 = 0$
P2	4	11	11	$11 - 4 = 7$
P3	9	20	20	$20 - 9 = 11$
P4	3	23	23	$23 - 3 = 20$
P5	6	29	29	$29 - 6 = 23$

Average turnaround time = $90 / 5 = 18$ ms

Average waiting time = $61 / 5 = 12.2$ ms

SJF (Shortest Job First, non-preemptive)



Gantt chart

Process	BT	Completion time (CT)	TAT = CT	WT = TAT – BT
P1	7	20	20	$20 - 7 = 13$

P2	4	7	7	$7 - 4 = 3$
P3	9	29	29	$29 - 9 = 20$
P4	3	3	3	$3 - 3 = 0$
P5	6	13	13	$13 - 6 = 7$

Average turnaround time = $72 / 5 = 14.4$ ms

Average waiting time = $43 / 5 = 8.6$ ms

Round Robin (time quantum = 3 ms)

P1	P2	P3	P4	P5	P1	P2	P3	P5	P1	P3	
0	3	6	9	12	15	18	19	22	25	26	29

Gantt chart

Process	BT	Completion time (CT)	TAT = CT	WT = TAT - BT
P1	7	26	26	$26 - 7 = 19$
P2	4	19	19	$19 - 4 = 15$
P3	9	29	29	$29 - 9 = 20$
P4	3	12	12	$12 - 3 = 9$
P5	6	25	25	$25 - 6 = 19$

Average turnaround time = $111 / 5 = 22.2$ ms

Average waiting time = $82 / 5 = 16.4$ ms

Algorithm	Average WT (ms)	Average TAT (ms)
FCFS	12.2	18
SJF	8.6	14.4
RR (q = 3)	16.4	22.2

SJF gives the minimum average waiting time.

(Each algorithm: Gantt chart 1 M, calculation 1 M)

b) IPC and message passing model.

6 M

Inter Process Communication (IPC) is a mechanism that allows **cooperating processes** to exchange data and information and to synchronise their actions. Reasons: information sharing, computation speed-up, modularity, convenience. The two fundamental models are **shared memory** and **message passing**.

Message passing model: processes communicate by exchanging messages through the kernel, without sharing an address space. Two operations are provided: **send(message)** and **receive(message)**. If processes P and Q want to communicate, a **communication link** is established between them and messages are exchanged using send/receive.

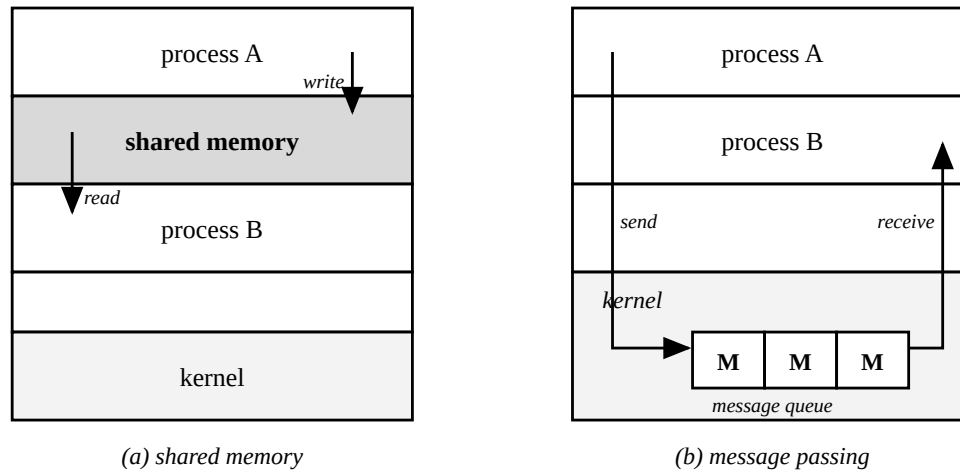


Fig.: Models of IPC — (a) shared memory, (b) message passing

- **Direct communication:** each process names the other explicitly — send(P, msg), receive(Q, msg).
- **Indirect communication:** messages are sent to and received from mailboxes (ports) — send(A, msg), receive(A, msg).
- **Synchronisation:** blocking (synchronous) or non-blocking (asynchronous) send and receive.
- **Buffering:** zero capacity, bounded capacity or unbounded capacity message queue.

Advantages: 1) Easy to implement and useful for exchanging small amounts of data. 2) No conflicts to avoid, since memory is not shared. 3) Works in distributed systems where processes are on different computers connected by a network.

(IPC 1 M, explanation 2 M, diagram 2 M, advantages 1 M)

c) Deadlock and Banker's algorithm.

6 M

Deadlock: A set of processes is in a deadlock state when every process in the set is waiting for an event (release of a resource) that can be caused only by another process in the same set. None of them can proceed.

Banker's algorithm is a deadlock-avoidance algorithm for resources with multiple instances.

When a process requests resources, the system grants them only if the system remains in a **safe state**; otherwise the process must wait. (Like a bank that never lends money in such a way that it cannot satisfy all its customers.)

Data structures (n processes, m resource types):

- **Available[m]:** number of free instances of each resource type.
- **Max[n][m]:** maximum demand of each process.
- **Allocation[n][m]:** number of instances currently allocated to each process.
- **Need[n][m]:** remaining need; $\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$.

Safety algorithm:

1. Let $\text{Work} = \text{Available}$ and $\text{Finish}[i] = \text{false}$ for all i.
2. Find an i such that $\text{Finish}[i] = \text{false}$ and $\text{Need}_i \leq \text{Work}$. If no such i exists, go to step 4.
3. $\text{Work} = \text{Work} + \text{Allocation}_i$; $\text{Finish}[i] = \text{true}$; go to step 2.

4. If $\text{Finish}[i] = \text{true}$ for all i , the system is in a safe state (the order of selection is the safe sequence); otherwise it is unsafe.

Resource-request algorithm: when P_i requests Request_i : (1) if $\text{Request}_i > \text{Need}_i$, error; (2) if $\text{Request}_i > \text{Available}$, P_i waits; (3) pretend to allocate ($\text{Available} -= \text{Request}$, $\text{Allocation} += \text{Request}$, $\text{Need} -= \text{Request}$) and run the safety algorithm — if safe, grant; else restore the old state and P_i waits.

(Definition 1 M, data structures 2 M, algorithm 3 M)

Q.6

a) Operating system; batch and multiprocessor OS.

6 M

Operating system: An operating system is a program (system software) that acts as an **interface between the user and the computer hardware**. It manages all hardware and software resources (CPU, memory, files, I/O devices) and provides an environment in which users can execute programs conveniently and efficiently.

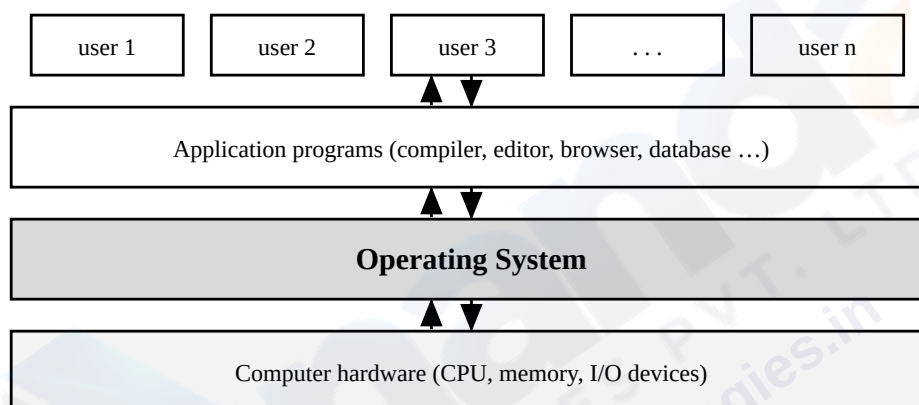


Fig.: Abstract view of a computer system

i) Batch operating system: Users do not interact with the computer directly. Each user prepares a job (program, data, control cards) and submits it to the operator. Jobs with similar needs are grouped into **batches** and run one after another by a resident monitor. Advantages: less idle time between jobs, good for large repetitive work. Disadvantages: no interaction, CPU idle during I/O, long turnaround time. **Example:** payroll / bank statement processing, IBM OS/360 (early batch systems).

ii) Multiprocessor operating system: A system with **two or more processors** in close communication that share the computer bus, clock, memory and peripheral devices (tightly coupled / parallel system). Advantages: increased throughput, economy of scale (shared peripherals), increased reliability (graceful degradation). Types: **symmetric multiprocessing (SMP)** — every processor runs the OS and user tasks; **asymmetric multiprocessing** — a master processor assigns work to slave processors. **Example:** Windows, Linux and macOS on multi-core processors (SMP).

(OS definition 2 M, each type with example 2 M)

b) FIFO and Optimal page replacement (3 frames).

6 M

Reference string (16 references): 4, 7, 6, 1, 7, 6, 1, 2, 7, 2, 4, 1, 6, 2, 7, 1; number of frames = 3.

FIFO (First In First Out)

Rule: replace the page that was brought into memory first (oldest page).

Ref.	4	7	6	1	7	6	1	2	7	2	4	1	6	2	7	1
F1	4	4	4	1	1	1	1	1	1	1	4	4	4	2	2	2
F2		7	7	7	7	7	7	2	2	2	2	1	1	1	7	7
F3			6	6	6	6	6	6	7	7	7	7	6	6	6	1
Fault	F	F	F	F	H	H	H	F	F	H	F	F	F	F	F	F

F = page fault, H = hit. **Number of page faults (FIFO) = 12** (hits = 4); page fault ratio = $12/16 = 0.75$

Optimal (OPT)

Rule: replace the page that will not be used for the longest time in future (if several pages are never used again, the oldest of them is replaced).

Ref.	4	7	6	1	7	6	1	2	7	2	4	1	6	2	7	1
F1	4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1
F2		7	7	7	7	7	7	7	7	7	4	4	6	6	6	6
F3			6	6	6	6	6	2	2	2	2	2	2	2	7	7
Fault	F	F	F	F	H	H	H	F	H	H	F	H	F	H	F	H

F = page fault, H = hit. **Number of page faults (OPT) = 8** (hits = 8); page fault ratio = $8/16 = 0.50$

Result: FIFO = 12 page faults, Optimal = 8 page faults. The **Optimal** algorithm gives fewer page faults (it gives the lowest possible page-fault rate, but needs future knowledge, so it is used mainly for comparison).

(Each algorithm 3 M)

c) Linux commands ps, kill, sleep.

6 M

i) ps (process status): displays information about the currently running processes — PID, terminal, CPU time and command name.

Syntax: **ps [options]** · Options: **-e** / **-A** all processes, **-f** full format (UID, PPID, start time), **-u** user processes of a user, **-l** long format.

```
$ ps
  PID TTY          TIME CMD
 2314 pts/0    00:00:00 bash
 2390 pts/0    00:00:00 ps
```

ii) kill: sends a signal to a process, normally to terminate it. Default signal is SIGTERM (15); **-9** (SIGKILL) terminates it forcibly. **kill -l** lists all signals.

Syntax: **kill [-signal] PID**

```
$ sleep 300 &
[1] 2401
$ kill -9 2401
[1]+  Killed                  sleep 300
```

iii) sleep: suspends (delays) execution of the shell / script for the given time. Suffix s = seconds (default), m = minutes, h = hours, d = days.

Syntax: **sleep** NUMBER[suffix]

```
$ sleep 5          # the prompt returns after 5 seconds
$ echo "start"; sleep 2; echo "end"
start
end                # printed 2 seconds later
```

(PID numbers differ on every system.)

(Each command: use 1 M, syntax with example 1 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in