Operating Systems

Short notes for B.Tech CSE / IT · BCA · MCA · GATE CS

Uncodemy Handwritten Notes · free for students

I. Introduction

- **Operating System (OS)** = software that sits between the user / programs and the hardware.
- It manages CPU, memory, files, and I/O devices, and gives programs an easy way to use them.
- Examples: Windows, Linux, macOS, Android, iOS.

➤ Main jobs of an OS

1. Process management (running programs)
2. Memory management (RAM)
3. File management (storage)
4. I/O device management (keyboard, disk, printer)
5. Security and protection
6. User interface (CLI / GUI)

➤ Types of OS

Type	Idea	Example
Batch	Similar jobs run together, no user interaction	Old IBM systems
Multiprogramming	Many programs in RAM, CPU never idle	Early UNIX
Time-sharing	CPU time split in small slices between users	UNIX, Linux
Real-time (RTOS)	Must respond within a fixed deadline	Car airbag, pacemaker
Distributed	Many computers work as one	Cluster systems

EXAM POINT Multiprogramming = keep the CPU busy. Time-sharing = multiprogramming + fast switching so every user feels the system is theirs.

➤ Kernel, system call, dual mode

- **Kernel** = core of the OS, always in memory.

- **User mode** (mode bit = 1) and **kernel mode** (mode bit = 0). Risky instructions run only in kernel mode.
- **System call** = the way a program asks the kernel for a service. Example: `fork()`, `exec()`, `read()`, `write()`, `open()`.
- Kernel types: **Monolithic** (all services in kernel, fast, e.g. Linux), **Microkernel** (only basic services in kernel, safer, e.g. Minix, QNX).

2. Process

- **Program** = passive code on disk. **Process** = program in execution (active).
- A process in memory has: Text (code), Data (globals), Heap (dynamic memory), Stack (function calls).

➤ Process states

```
new --admit--> ready --dispatch--> running --exit--> terminated
               ^   <--interrupt--    |
               |                     | I/O or event wait
               +---- waiting <-------+
                  (I/O done)
```

- **PCB (Process Control Block)** stores everything about a process: PID, state, program counter, registers, priority, memory info, open files.
- **Context switch** = save the PCB of the old process, load the PCB of the new one. It is pure overhead (no useful work).

➤ Schedulers

Scheduler	Decides	Speed
Long-term (job)	Which jobs enter RAM (controls degree of multiprogramming)	Slow
Short-term (CPU)	Which ready process gets the CPU next	Very fast
Medium-term	Swaps processes out of / back into RAM	Medium

➤ Threads

- **Thread** = light-weight process. Threads of one process share code, data and files, but each has its own stack, registers and program counter.
- Benefits: faster creation, cheaper switching, better use of multi-core CPUs.

- `fork()` creates a child process. If you call `fork()` n times, total processes = `2^n` (child processes = `2^n − 1`).

FORMULA n fork() calls → 2^n processes in total, 2^n - 1 of them children.

3. CPU Scheduling

➤ Important terms

- **Arrival Time (AT)**: when the process comes. **Burst Time (BT)**: CPU time it needs.
- **Completion Time (CT)**: when it finishes.
- **Turnaround Time (TAT)** = CT − AT.
- **Waiting Time (WT)** = TAT − BT.
- **Response Time** = time of first run − AT.

FORMULA TAT = CT - AT, WT = TAT - BT. Average = sum / number of processes.

➤ Algorithms

Algorithm	Rule	Preemptive?	Note
FCFS	First come, first served	No	Convoy effect (short jobs wait behind a long one)
SJF	Shortest burst first	No	Minimum average WT, but needs burst time in advance
SRTF	Shortest remaining time first	Yes	Preemptive SJF
Priority	Highest priority first	Both	Starvation, fixed by aging
Round Robin	Each process gets a time quantum	Yes	Best for time-sharing, good response time
Multilevel queue	Separate queues (system, interactive, batch)	Yes	Multilevel feedback queue lets processes move between queues

➤ Solved example (FCFS)

Process	AT	BT	CT	TAT	WT
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	8	16	14	6

- Gantt chart: `| P1 0-5 | P2 5-8 | P3 8-16 |`
- Average WT = (0 + 4 + 6) / 3 = 3.33, Average TAT = (5 + 7 + 14) / 3 = 8.67

> **TIP** Round Robin with a very big quantum behaves like FCFS. With a very small quantum, too many context switches happen.

4. Process Synchronization

- **Race condition**: the result depends on the order in which processes run while they share data.
- **Critical section (CS)**: the part of code that touches shared data.

➤ A correct CS solution needs 3 things

1. **Mutual exclusion**: only one process in the CS at a time.
2. **Progress**: if no one is in the CS, a waiting process must be able to enter (no needless blocking).
3. **Bounded waiting**: a limit on how many times others can enter before me (no starvation).

➤ Tools

- **Peterson's solution**: for 2 processes, uses `flag[2]` and `turn`. It satisfies all 3 conditions.
- **Semaphore** S = an integer used with two atomic operations:
  - `wait(S)` (P): while S <= 0 wait; S = S − 1
  - `signal(S)` (V): S = S + 1
- **Binary semaphore** (0/1), which works like a mutex lock, and **counting semaphore** (any value, for n resources).
- **Monitor**: a high-level construct where only one process is active inside at a time.

> **EXAM POINT** Classic problems: Producer–Consumer (bounded buffer), Readers–Writers, Dining Philosophers. Learn the semaphore code of Producer–Consumer: mutex = 1, full = 0, empty = n.

```
Producer:                          Consumer:
 wait(empty); wait(mutex);          wait(full); wait(mutex);
   add item to buffer                 remove item from buffer
 signal(mutex); signal(full);       signal(mutex); signal(empty);
```

5. Deadlock

- **Deadlock**: a set of processes wait forever, each holding a resource that another one needs.

➤ 4 necessary conditions (all must hold together)

1. **Mutual exclusion**: a resource can be used by only one process at a time
2. **Hold and wait**: a process holds one resource and waits for another
3. **No preemption**: a resource cannot be taken away by force
4. **Circular wait**: P1 waits for P2, P2 for P3, ..., Pn for P1

> **REMEMBER** "M-H-N-C": break any one condition and deadlock cannot happen (prevention).

➤ Handling deadlock

- **Prevention**: make sure one condition can never hold (e.g. order the resources to stop circular wait).
- **Avoidance**: Banker's algorithm, which grants a request only if the system stays in a **safe state**.
- **Detection and recovery**: let it happen, find it (wait-for graph), then kill a process or preempt a resource.
- **Ignore** (Ostrich algorithm): used by Windows and Linux.

➤ Banker's algorithm (key formulas)

- Need = Max − Allocation
- Process Pi can run if Need(i) <= Available. After it finishes: Available = Available + Allocation(i).
- If all processes can finish in some order, the state is **safe** and that order is the **safe sequence**.

> **FORMULA** Deadlock-free condition: n processes, each needs at most k units of one resource type, R total units. No deadlock if R >= n(k - 1) + 1.

6. Memory Management

- **Logical address**: made by the CPU. **Physical address**: the real address in RAM. The **MMU** converts one into the other.
- **Fragmentation**:
  - **Internal**: wasted space inside an allocated block (fixed partitions, paging).
  - **External**: enough total free space, but not in one piece (variable partitions, segmentation). Fixed by **compaction**.
- Allocation strategies: **First fit** (first hole that is big enough), **Best fit** (smallest hole that fits), **Worst fit** (largest hole).

➤ Paging

- Logical memory is split into pages, physical memory into frames of the same size.
- Logical address = (page number p, offset d). A page table maps p to a frame number f.
- No external fragmentation, but a little internal fragmentation in the last page.
- **TLB** = a fast cache for page table entries.

FORMULA EMAT with TLB = $h(t + m) + (1 - h)(t + 2m)$, where h = TLB hit ratio, t = TLB time, m = memory access time.

FORMULA Number of pages = logical address space / page size. Page table size = number of pages x entry size.

➤ Segmentation

- Memory is split by logical units (code, stack, data) of different sizes. Address = (segment number, offset). It can suffer external fragmentation.

7. Virtual Memory

- It lets a program run even when it is not fully in RAM. Pages are loaded only when needed (**demand paging**).
- **Page fault**: the page is not in RAM, so the OS loads it from disk.

FORMULA Effective access time = $(1 - p)$ x memory access + p x page fault service time, where p = page fault rate.

Page replacement algorithms

Algorithm	Replace which page	Note
FIFO	The oldest page	Suffers Belady's anomaly (more frames can give more faults)
Optimal (OPT)	The page used furthest in the future	Fewest faults, cannot be built in practice, used as a benchmark
LRU	The page not used for the longest time	Good in practice, no Belady's anomaly

Example: reference string 7 0 1 2 0 3 0 4, 3 frames

- FIFO → 7 faults. LRU → 6 faults. OPT → 6 faults.
- **Thrashing**: the CPU spends more time swapping pages than running programs. Cause: too many processes and too few frames. Fix: working-set model, fewer processes.

8. File System and Disk Scheduling

- File allocation methods: **Contiguous** (fast, but external fragmentation), **Linked** (no fragmentation, slow random access, e.g. FAT), **Indexed** (index block holds pointers, e.g. UNIX inode).
- Directory structures: single-level, two-level, tree, acyclic graph.

Disk scheduling

Algorithm	Idea
FCFS	Serve requests in order of arrival
SSTF	Serve the request nearest to the head (can starve far requests)
SCAN (elevator)	Go to one end serving requests, then reverse
C-SCAN	Go to one end, jump back to the start, serve in one direction only
LOOK / C-LOOK	Like SCAN / C-SCAN but turn at the last request, not at the disk end

FORMULA Disk access time = seek time + rotational latency + transfer time.
Average rotational latency = 1/2 of the rotation time.

9. Most Asked Questions (practice)

1. Difference between process and thread.
2. Explain process states with a diagram. What is a PCB?
3. Solve FCFS, SJF, SRTF and Round Robin for given processes (find avg WT and TAT).
4. What are the 4 necessary conditions for deadlock? Explain Banker's algorithm with an example.
5. What is a semaphore? Solve Producer–Consumer using semaphores.
6. Paging vs segmentation. Internal vs external fragmentation.
7. Find page faults with FIFO, LRU and Optimal. What is Belady's anomaly?
8. What is thrashing and how is it controlled?
9. Explain SCAN and C-SCAN disk scheduling with an example.
10. What is a system call? Monolithic vs microkernel.

TIP In GATE, numericals come from scheduling, page replacement, EMAT, fork() and Banker's. Practise 10 questions of each.