Advanced Process Management.

Starts at 9:06pm

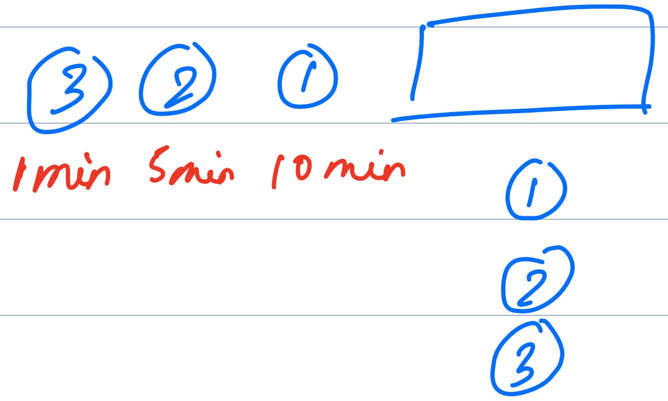
1. Process Scheduling
2. IPC
3. Signals and Signal Handling
4. Advanced Commands.

Process Scheduling.

All about which task to do when. by the O.S

Algorithms

1. FCFS



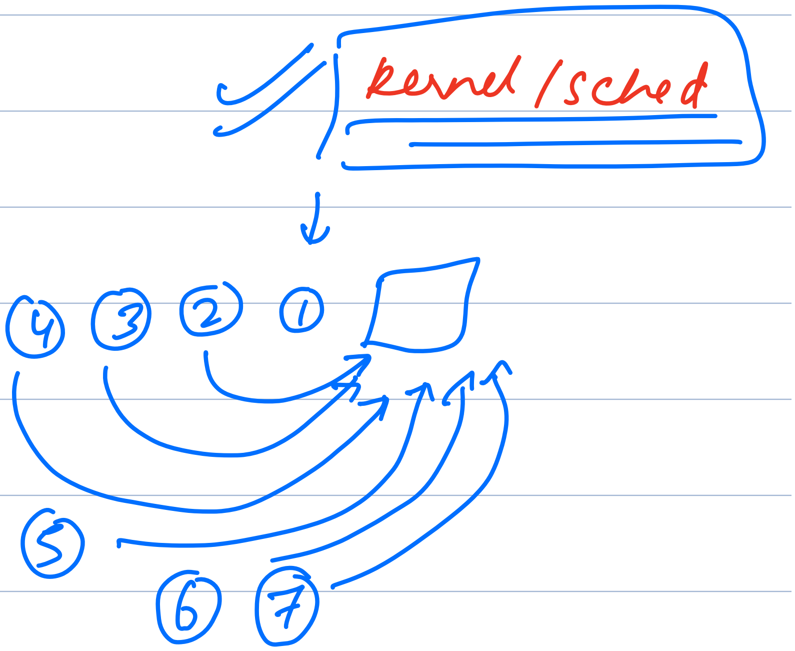
Drawback → Longer wait periods for short tasks.

② Shortest Job First.

100 Process

98 short P

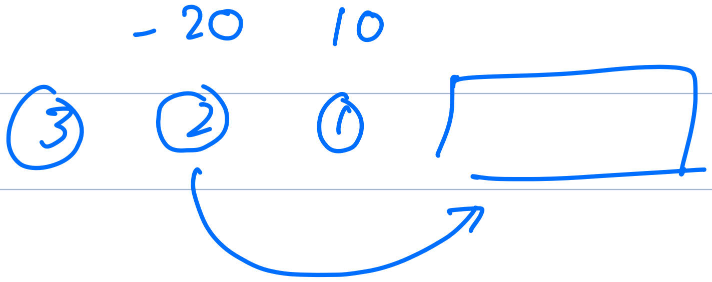
2 long P



Drawback →

Starvation of longer jobs.

③ Priority Scheduling.



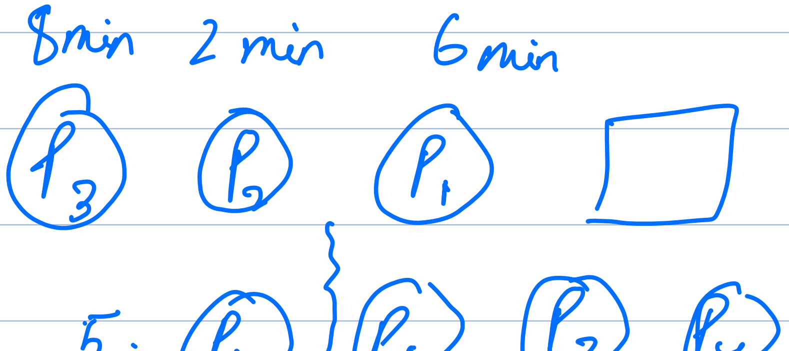
-20 → Highest

19 → lowest

④ Round Robin.

Time Quantum → Max amt. in the CPU. for a process.

T.Q. → 4min



Round 1 $P_1 \rightarrow 6-4 = \underline{\underline{2 \text{ min}}}$ Remaining.

$P_2 \rightarrow$ 2min → Finished

$P_3 \rightarrow 8-4 = \underline{\underline{4 \text{ min}}}$ Remaining

$P_4 \rightarrow 5-4 = \underline{\underline{1 \text{ min}}}$ Remaining.

P_1 → 2min
P_3 → 4min
P_4 → 1min

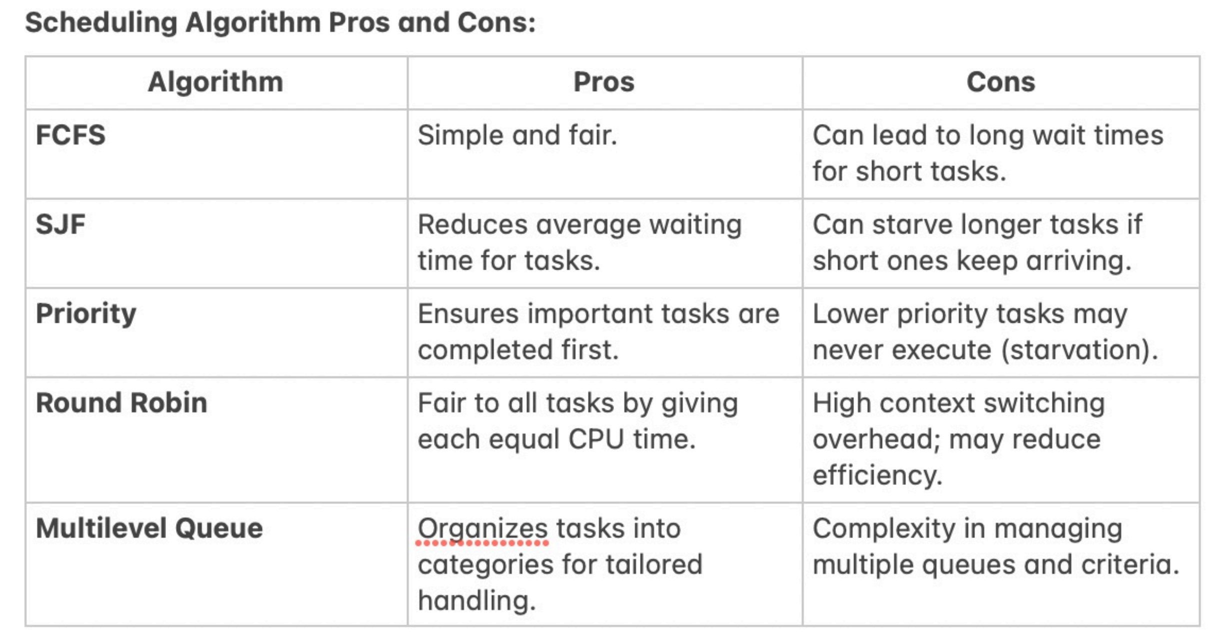
Ensure Fairness.

Every Process gets equal amount of time.

⑤ Multi Level queue Scheduling.

Q_0	Q_2	Q_3
P_1	P_1	P_1
P_2	P_2	P_2
P_3	P_3	P_3

↑ Emergency.
Appointment
Walk in.



Scheduling Algorithm Pros and Cons:

Algorithm	Pros	Cons
FCFS	Simple and fair.	Can lead to long wait times for short tasks.
SJF	Reduces average waiting time for tasks.	Can starve longer tasks if short ones keep arriving.
Priority	Ensures important tasks are completed first.	Lower priority tasks may never execute (starvation).
Round Robin	Fair to all tasks by giving each equal CPU time.	High context switching overhead; may reduce efficiency.
Multilevel Queue	Organizes tasks into categories for tailored handling.	Complexity in managing multiple queues and criteria.

→ Premptive Scheduling
- → Round Robin
- → Priority.

→ Non-Preemptive Sched.
- → FCFS
- → SJF

→ Batch OS. → maximise throughput.
- FCFS
- SJF

→ Interactive system.

Round Robin → dishes / hour

Priority.

→ process / hour.

→ Real time systems.

EDF (Earliest Deadline First)

P_3 P_2 P_1

$\underline{\underline{\text{1min.}}}$ 3min 2min

Throughput is how many tasks (or meals) are completed in a set period of time.

CPU utilization is how much of the CPU’s capacity (or the chefs’ working time) is being used actively.

You are a DevOps engineer responsible for managing a microservices-based application hosted on a cloud infrastructure. Recently, users have been complaining about slow response times when interacting with your application, especially during peak hours.

Upon investigating the monitoring dashboards, you observe the following:

* CPU usage on the servers hosting the services is consistently high, around 90–95%.

* However, the throughput (i.e., the number of processed requests per second) is lower than expected, even under the current traffic load.

Question:

1. What might be the possible causes for high CPU usage but low throughput in this scenario?

2. How would you go about diagnosing and troubleshooting this problem to improve the throughput while managing CPU usage?

Break → 10:16 pm.

Answer ->

Causes for High CPU Usage but Low Throughput:

* Inefficient Code or Resource-Intensive Operations: There could be inefficient algorithms or poorly optimized code that consumes excessive CPU cycles without completing tasks efficiently. For example, a service might be performing heavy computations, such as data parsing or processing large datasets unnecessarily.

* Thread Contention/Locking Issues: If the application uses shared resources (e.g., database connections or memory), there may be lock contention. Threads may be waiting for locks to release, causing delays while keeping the CPU busy with context switching or spinning on the lock.

* I/O Wait or Latency: If the system frequently waits on external resources (like databases, disk I/O, or network APIs), the CPU might be used inefficiently. High CPU utilization could be caused by processes stuck in an I/O wait state, leading to fewer completed tasks and hence low throughput.

* Too Many Active Threads or Processes: If the system is spawning too many threads/processes (e.g., due to improper

scaling or configuration), the CPU could be spending a lot of time on context switching rather than doing productive work, leading to low throughput.

* High Request Latency: The services could be experiencing delays due to dependent services or external APIs responding slowly, which causes the CPU to wait or retry operations, contributing to low throughput.

Steps to Diagnose and Troubleshoot the Problem:

* Analyze CPU Profiling: Use CPU profiling tools like top, htop to examine which processes or threads are consuming the most CPU resources. Look for any processes or functions that are unexpectedly consuming a large percentage of CPU cycles.

* Examine the Thread Model: Check whether too many threads are being spawned or if improper thread pool management is causing high CPU overhead due to excessive context switching. Tuning thread pools or limiting the number of active threads can help improve throughput.

* Optimize Code and Database Queries: Look for inefficient code paths or slow database queries that could be optimized. For example, reduce heavy computations, batch expensive operations, or improve query performance by adding indexes or caching results.

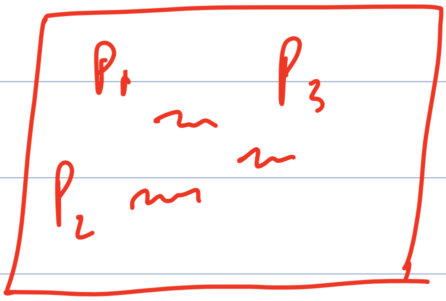
* Scale Infrastructure: If the traffic load is high and the system is reaching its CPU limits, consider scaling out by adding more servers, using autoscaling, or optimizing load balancing to distribute traffic more evenly across instances.

* Implement Caching: If the same data is being requested frequently, caching results in memory (e.g., using Redis or Memcached) can reduce the load on the CPU and external resources, improving overall throughput

Inter Process Communication.

IPC is the set of techniques and mechanisms that allow software processes to exchange information and synchronize their actions.

① Shared memory.



Advantages → Fast.

disadvantages → Security.

② Message Passing.

→ Packets. send and receive.

→ queues, Sockets.

Models.

① Direct Message Passing
using PID

② Indirect " "

using different objects

→ queues { Mediators }
→ Sockets

① → queues.

① unnamed pipes

apple.txt
1. apple.
2. banana
3. orange.

cat apple.txt | grep banana.

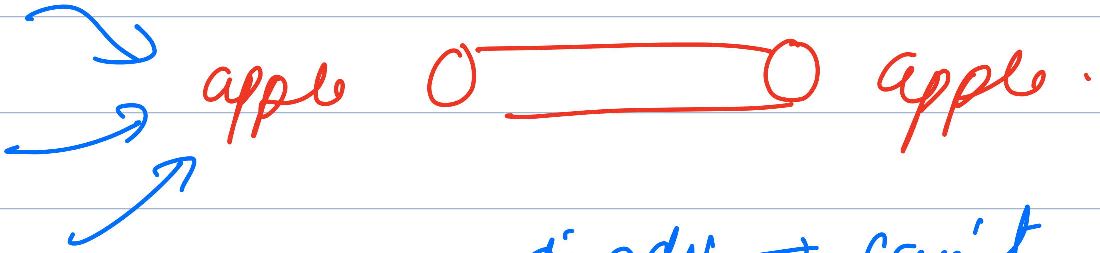
→ they are temporary.
→ communication b/w parent and child process
→ exist only when the process is running.

The pipe (|) creates a communication channel between these two processes. The shell creates this pipe, which is temporary and exists only for the duration of the command execution.

② Named pipes.

→ persisent
→ independent of process lifecycle.

mkfifo pipe



disadv → can't use for communication over a network.

② Sockets.

→ endpoints.

Server Client

→ opens a socket.
→ assigns IP
→ opens for comn.
→ comn.
→ closes.

→ creats a socket
→ connects
→ sends/receives.
→

Signals and Signal Handling.

running a script (10 mins)

↓

wrong parameters.

↓ ↓

ctrl + c   kill -9

Signals.

kill

(15)

Signal Handling

Signal handling is essentially equipping your program with a protocol for managing incoming signals.

Types of signals.

1. SIGHUP (signal hang up) (
   happens when disconnected.
2. SIGINT Interrupt.
   ctrl+C

→ stops execution of a process.

③ SIGKILL - 9

forceful

④ SIGTERM ⑮

kill

⑤ SIGSTOP Stopping (19)

⑥ SIGCONT Resuming (18)

kill -l → All the signals.

→ NICENESS.

-20 19

↓ ↓

highest lowest.

nice: Start a new process with a specified priority.

nice

renice: Change the priority of an already running process.

renice.

```
→ nice -n 19 ./sleep.sh &
```

```
→ renice -n -15 -p pid.
```

```
ps  -o  pid, nice, cmd  | grep sleep.
```