OPERATING SYSTEM-SCHEDULING ALGORITHMS

WHY SCHEDULING?

- **Why do we need scheduling?**
  A typical process involves both I/O time and CPU time.
- In a uni programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time.
- In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

SCHEDULING CRITERIA

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

SCHEDULING ALGORITHMS

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

- These algorithms are either **non-preemptive or preemptive**.
- Non-preemptive algorithms - once a process enters the running state, it cannot be preempted until it completes its allotted time.
- Preemptive scheduling -based on priority

CPU SCHEDULER

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **non-preemptive**
- All other scheduling is **preemptive - implications for data sharing between threads/processes**

DISPATCHER

- Dispatcher module gives control of the CPU to the process selected by the scheduler;
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

FIRST-COME, FIRST-SERVED (FCFS) SCHEDULING

Process	Burst Time
P_1	24
P_2	3
P_3	3

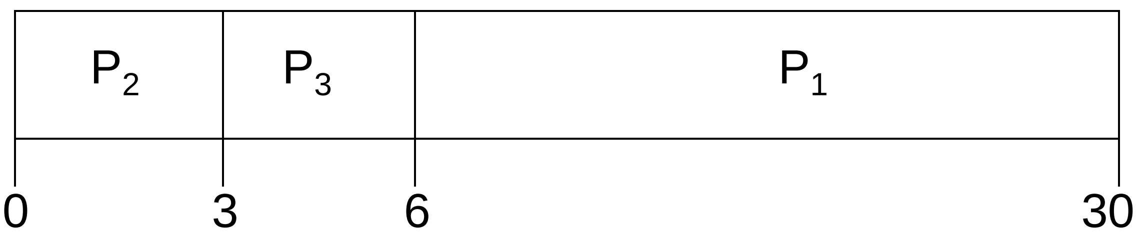
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order:

$$P_2\ ,\ P_3\ ,\ P_1$$

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case

ROUND ROBIN (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- We can predict wait time: If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - *Time quantum(q)* large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ too many process /context switching

EXAMPLE OF RR WITH TIME QUANTUM = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

- Typically, higher average turnaround than few other, but better *response*.