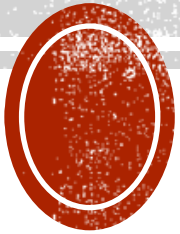
SCHEDULING

Chapter 2.3



PROCESS SCHEDULING

- Round Robin Scheduling
- Priority Scheduling
- Multiple Queues

SCHEDULING

- Multiprogram - Multiple process
- Two or more simultaneously in ready state
- If only one CPU is available, a choice has to make which process to run next.
- The part of the operating system which makes decision is called **scheduler**.
- And the algorithm it uses is called the **scheduling algorithm**.
- By switching the CPU among processes, the operating system can make the computer **more productive**.

CPU BURST VS. I/O BURST

- CPU Bound
  - The process that uses CPU until Quantum expire
- I/O Bound
  - The process that use CPU briefly then generate I/O
- CPU-bound processes have a long CPU-burst while I/O-bound processes have short CPU burst.
  - *when I/O bound process wants to run, it should get a chance quickly.*

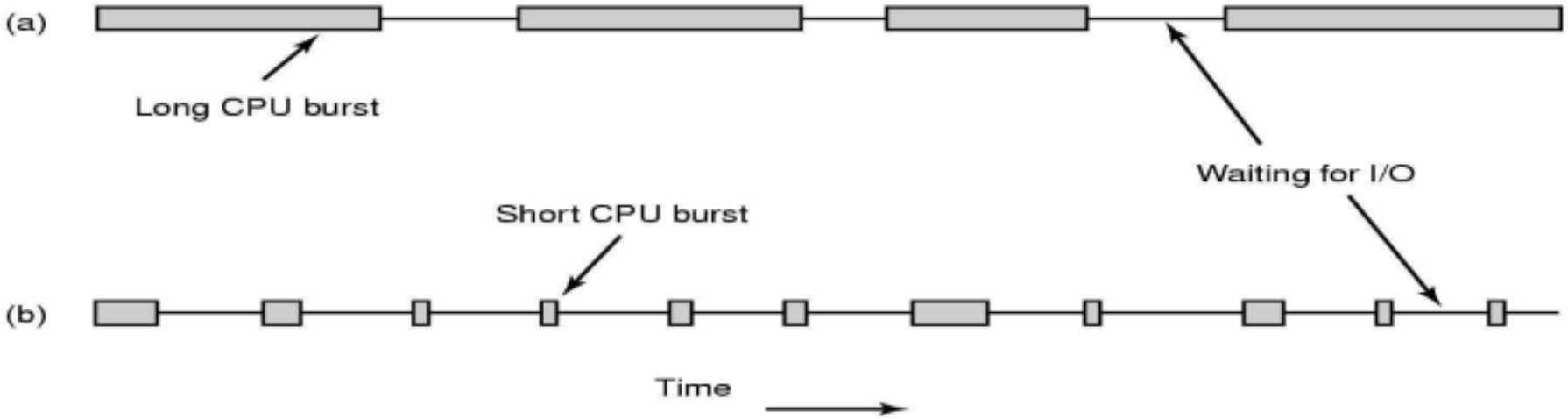


Fig: Process behavior

WHEN TO SCHEDULE

- 1. When a process switches from the **running state to the waiting state** (for. example, I/O request, or invocation of wait for the termination of one of the child processes).
- 2. When a process switches from the **running state to the ready state** (for example, when an interrupt occurs).
- 3. When a process switches from the **waiting state to the ready state** (for example, completion of I/O).
- 4. When a process **terminates**.

PREEMPTIVE VS. NONPREEMPTIVE

- **Nonpreemptive:**
  • Once a process has been given the CPU, it runs until blocks for I/O or termination.
  • Treatment of all processes is fair.
  • Response times are more predictable.
  • Useful in real-time system.
  • Shorts jobs are made to wait by longer jobs - no priority
- **Preemptive:**
  • Processes are allowed to run for a maximum of some fixed time.
  • Useful in systems in which high-priority processes requires rapid attention.
  • In timesharing systems, preemptive scheduling is important in guaranteeing acceptable response times.
  • High overhead

DISPATCHER VS. SCHEDULER

- Dispatcher
  - Low level mechanism.
  - **Responsibility:** Context switch
  - Save execution state of old process in PCB.
  - Load execution state of new process from PCB to registers.
  - Change the scheduling state of the process (running, ready, blocked)
  - Switch from kernel to user mode.
- Scheduler
  - Higher-level policy.
  - Responsibility: Which process to run next.

SCHEDULING CRITERIA

- Many criteria have been suggested for comparing CPU scheduling algorithms.
- ***Throughput***
  - number of processes completed per unit **time**.
- ***Turnaround time***
  - average time from the moment that a batch job is submitted until the moment it is completed.
  - **Turnaround time = Burst time + Waiting time or**
  - **Turnaround time = Exit time - Arrival time**
- ***CPU utilization***
  - ***Measures efficiency of CPU***
  - the number of processes executed by the CPU in a given amount of time.
- ***Waiting time***
  - Waiting time is the total time spent by the process in the ready state waiting for CPU.
  - **Waiting time = Turnaround time - Burst time**
- ***Response time***
  - *Required for interactive systems, t*he time between issuing a command and getting the result.

Process	Arrival time	Burst time
P1	0 ms	8 ms
P2	0 ms	7 ms
P3	2 ms	10 ms

Gantt Chart

P1		P2		P3	
0 ms	8 ms	8 ms	15 ms	15 ms	25 ms

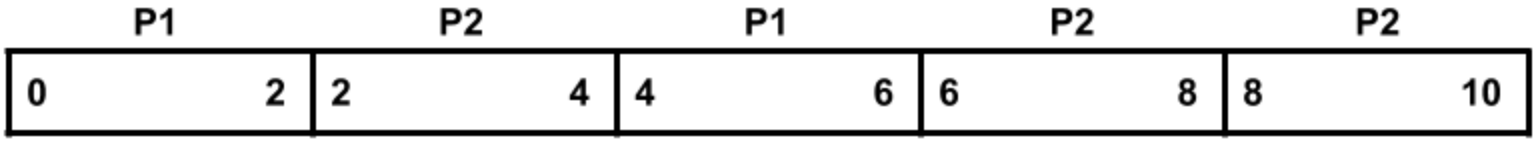
AfterAcademy

- waiting time for all the 3 processes will be:
- **P1:** 0 ms
- **P2:** 8 ms because P2 have to wait for the complete execution of P1 and arrival time of P2 is 0 ms.
- **P3:** 13 ms becuase P3 will be executed after P1 and P2 i.e. after 8+7 = 15 ms and the arrival time of P3 is 2 ms. So, the waiting time of P3 will be: 15-2 = 13 ms.

Process	Arrival time	Burst time
P1	0 ms	4 ms
P2	0 ms	6 ms

Time Quantum = 2ms

Gantt Chart



AfterAcademy

- response time of the process P2 is 2 ms because after 2 ms, the CPU is allocated to P2 and the waiting time of the process P2 is 4 ms i.e turnaround time - burst time (10 - 6 = 4 ms).

SCHEDULING ALGORITHMS

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Shortest-Remaining-Time-First (SRTF)
- Round Robin (RR)
- Priority
- Multiple Queues

FIRST COME FIRST SERVED (FCFS)

- Processes are assigned the CPU in the order they request it.
- Once the process has the CPU, it runs to completion – **Nonpreemptive.**
- Easily implemented, by managing a simple queue or by storing time the process was received.
- Ready jobs are inserted at the last of queue.
- When jobs are blocked, new job from the queue is fetched.
- If blocked jobs gets ready, it would be inserted at the end of the queue.
- Fair to all processes.
- **Problems:**
  No guarantee of good response time.
  Large average waiting time.
  Not applicable for interactive system.

SHORTEST JOB FIRST (SJF)

- The processing times are known in advanced.
- selects the process with shortest expected processing time.
- In case of the tie FCFS scheduling is used.
- The decision policies are based on the CPU burst time.
- **Advantages:**
  Reduces the average waiting time over FCFS.
  Favors short jobs at the cost of long jobs.
- **Problems:**
  Estimation of run time to completion. Accuracy?
  Not applicable in timesharing system.

SJF PERFORMANCE

Scenario: Consider the following set of processes that arrive at time 0, with length of CPU-bust time in milliseconds.

Processes	Burst time
P1	24
P2	3
P3	3

if the processes arrive in the order P1, P2, P3 and are served in FCFS order,

The average waiting time is (0 + 24 + 27)/3 = 17.

if the processes are served in SJF

The average waiting time is (6 + 0 + 3)/3 = 3.

SHORTEST-REMAINING-TIME-FIRST (SRTF)

- Preemptive version of SJF.
  Any time a new process enters the pool of processes to be scheduled, the scheduler compares the expected value for its remaining processing time with that of the process currently scheduled. If the new process's time is less, the currently scheduled process is preempted.

- Merits:
  Low average waiting time than SJF.
  Useful in timesharing.

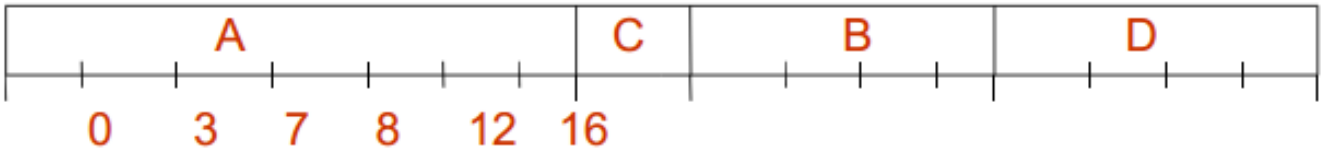
- Demerits:
  Very high overhead than SJF.
  Requires additional computation.
  Favors short jobs, longs jobs can be victims of starvation.

SRTF PERFORMANCE

Scenario: Consider the following four processes with the length of CPU-burst time given in milliseconds:

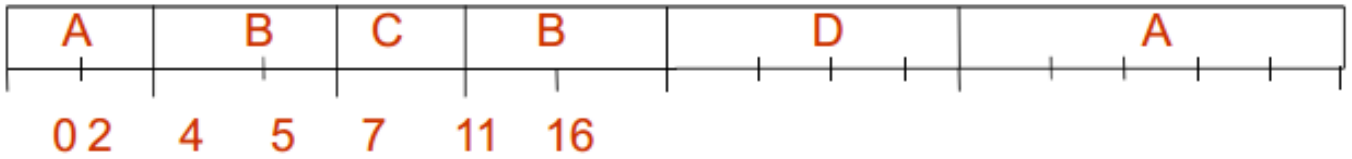
Processes	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

SJF:



Average waiting time = (0 + 6 + 3 + 7)/4 = 4

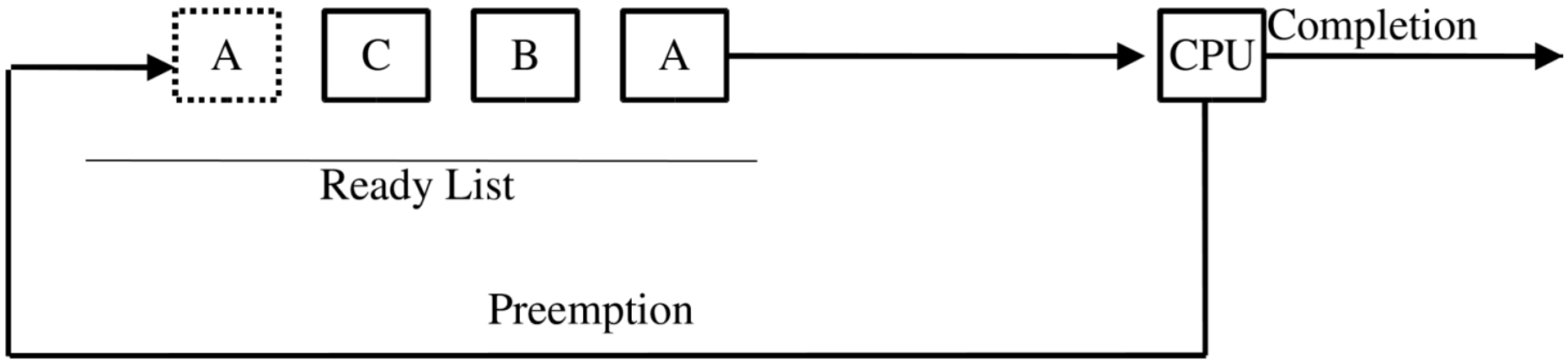
SRTF:



Average waiting time = (9 + 1 + 0 +2)/4 = 3

ROUND ROBIN (RR)

- Preemptive FCFS
- Each process is assigned a time interval (quantum), after the specified quantum, the running process is preempted to the last and a new process is allowed to run.
- Advantages:
  Fair allocation of CPU across the process.
  Used in timesharing system.
  Low average waiting time when process lengths very widely.
- Optimum value for quantum:
  - setting the quantum too short causes too many process switches and lowers the CPU efficiency, but setting it too long may cause poor response to short interactive requests.



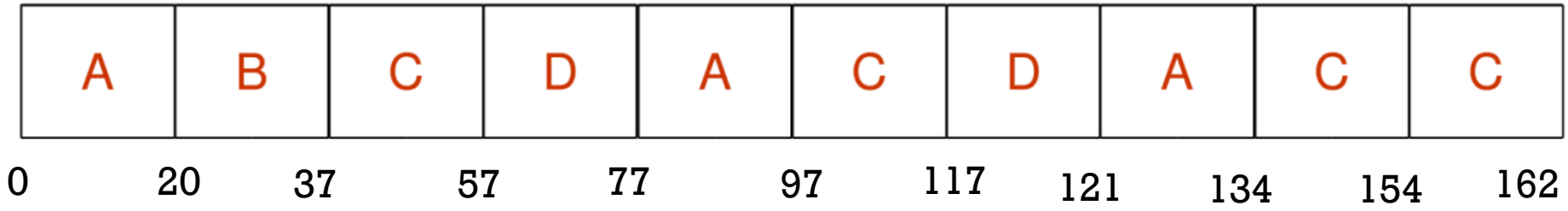
RR-PERFORMANCE

- Poor average waiting time when process lengths are identical.
  - Imagine 10 processes each requiring 10 msec burst time and 1msec quantum is assigned.
  - RR: All complete after about 100 times.
    FCFS is better! (About 20% time wastages in context-switching).
- Performance depends on quantum size.
- Quantum size:
  If the quantum is very large, each process is given as much time as needs for completion;
- If quantum is very small, system busy at just switching from one process to another process, the overhead of context-switching causes the system efficiency degrading.
- Optimal quantum size?
  Key idea: 80% of the CPU bursts should be shorter than the quantum.
  20-50 msec reasonable for many general processes.

RR WITH QUANTUM = 20

Process	Burst Time
A	53
B	17
C	68
D	24

The Gantt chart is:



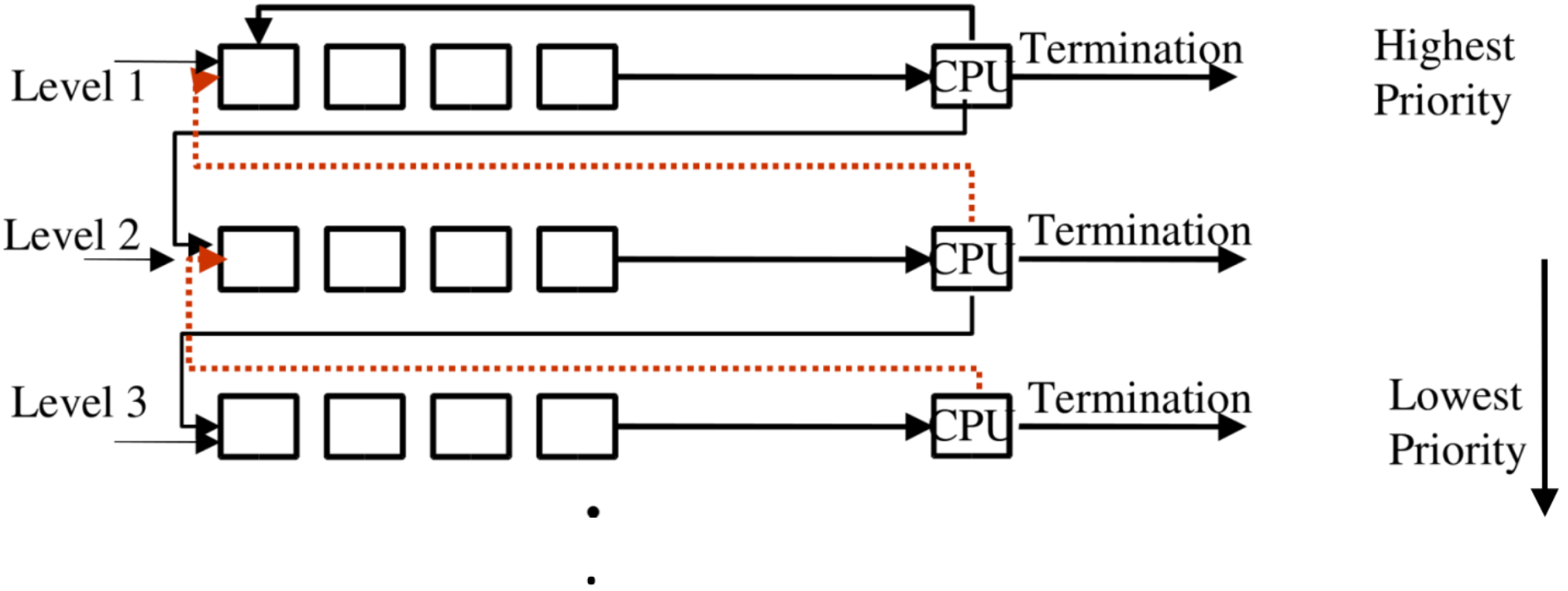
Typically, higher average turnaround than SJF, but better *response*.

PRIORITY

- Each process is assigned a priority value, and runnable process with the highest priority is allowed to run.
- FCFS or RR can be used in case of tie.
- To prevent high-priority process from running indefinitely, the scheduler may decrease the priority of the currently running process at each clock tick.
- Assigning Priority
  - Static:
    Some processes have higher priority than others.
    Problem: Starvation.
  - Dynamic:
    Priority chosen by system.
    Decrease priority of CPU-bound processes.
    Increase priority of I/O-bound processes.
    Many different policies possible…….
    E. g.: priority = (time waiting + processing time)/processing time.

MULTILEVEL FEEDBACK QUEUE(MFQ)

- CTSS machine (Compatible Time Sharing System 1961)
  - Process switching was very slow
  - So, Give large quantum to the CPU bound process



MFQ

- MFQ implements multilevel queues having different priority to each level (here lower level higher priority), and allows a process to move between the queues.
- If the process use too much CPU time, it will be moved to a lower-priority queue.
- Higher the level of queue(lower the priority) : larger the quantum size.
  - This leaves the I/O-bound and interactive processes in the high priority queue.

MFQ-EXAMPLE

- *Consider a MFQ scheduler with three queues numbered 1 to 3, with quantum size 8, 16 and 32 msec respectively.*
- The scheduler execute all process in queue 1, only when queue 1 is empty it execute process in queue 2 and process in queue 3 will execute only if queue 1 and queue 2 are empty.
- A process first enters in queue 1 and execute for 8 msec. If it does not finish, it moves to the tail of queue 2.
- If queue 1 is empty the processes of queue 2 start to execute in FCFS manner with 16 msec quantum. If it still does not complete, it is preempted and move to the queue 3.
- If the process blocks before using its entire quantum, it is moved to the next higher level queue.