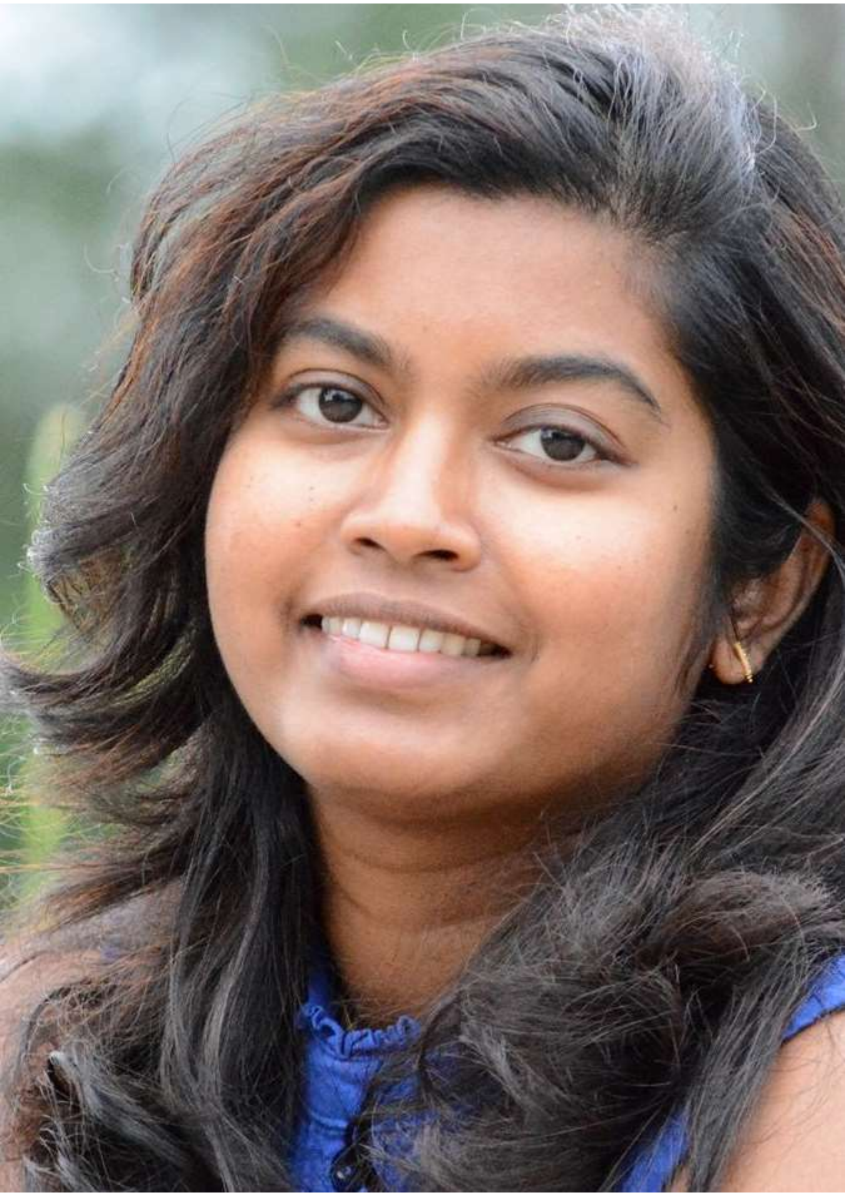
SCS 1312: Operating Systems

Dr. Dinuni Fernando, PhD

Based on Operating System Concepts by A.Silberschatz, P.Galvin, and G.Gagne

CPU Scheduling



Dr. Dinuni Fernando

- Holy Family Convent – Colombo 4
- Graduate from UCSC 2014 – CS major, 4 year degree, 1st class honors
- PhD – SUNY Binghamton, NY, USA 2019
- Joined UCSC 2019 as a Senior Lecturer

- Research Interests
  - Virtualization – Cloud computing
  - Blockchain and security
  - Software-defined networking

Learning Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems.
- To describe various CPU-scheduling algorithms.
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system.
- To examine the scheduling algorithms of several operating systems.

Single processor vs multi processor systems

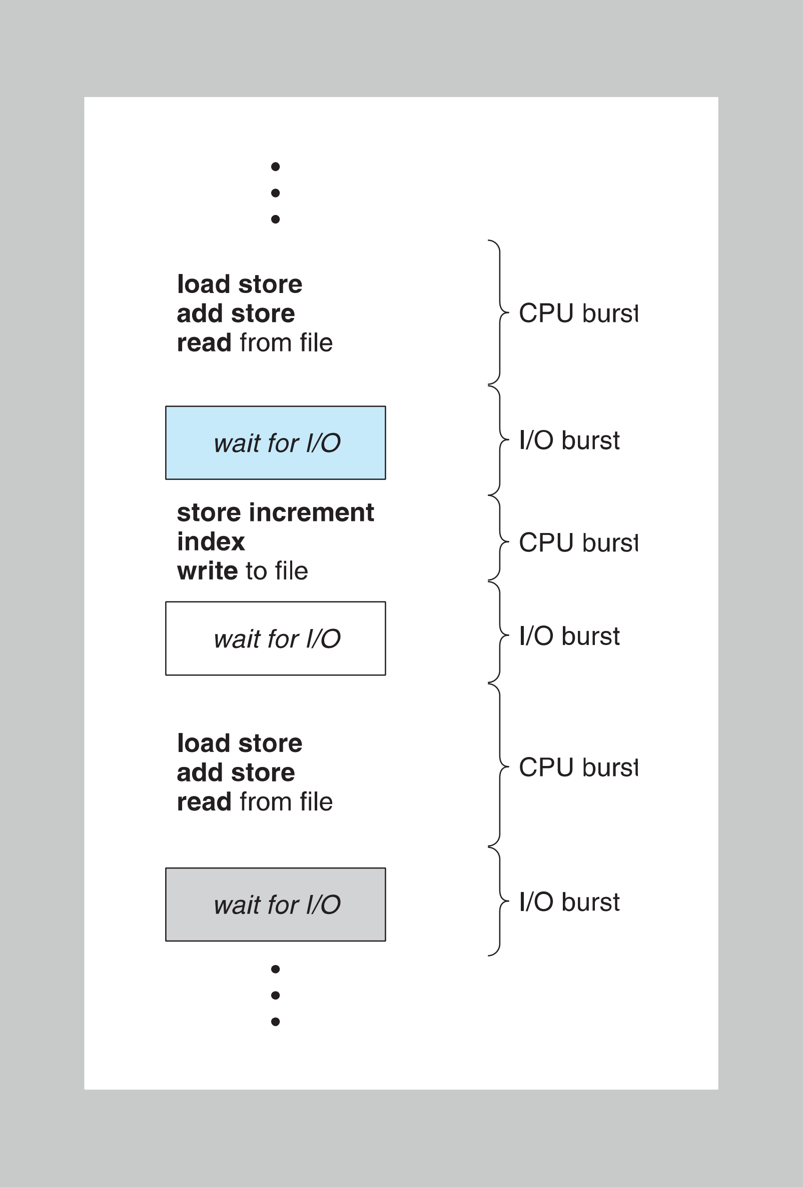
- Single processor systems : single process can run at a time; any others must wait until the CPU is free and can be rescheduled.
- Multiprogramming came to the picture with the objective of having some process running at all time to maximize CPU utilization.
- CPU utilization is the proportion of the total available processor cycles that are consumed by each process.
- CPU scheduling is the basis of multiprogrammed operating systems(OS).
  - By switching among processes, OS can make full of the computer.

Multiprogramming

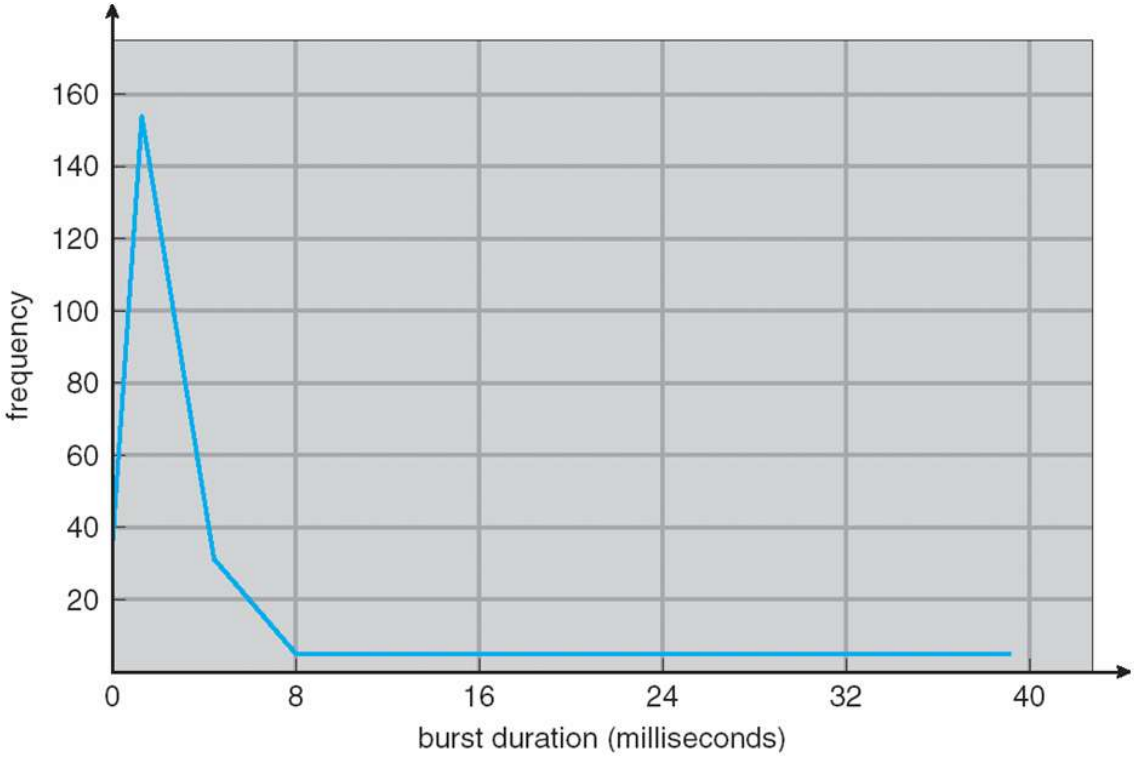
- Idea is relatively simple
  - A process is executed until it must wait, typically for completion of some I/O request. During this time CPU is idle. This waiting time is a waste.
  - With multiprogramming, we try to use this idle time productively by keeping several processes in memory. When one process has to wait, OS take the CPU away from process and give the CPU to another process.
- Therefore, scheduling is central to operating system.

CPU-I/O Burst Cycle

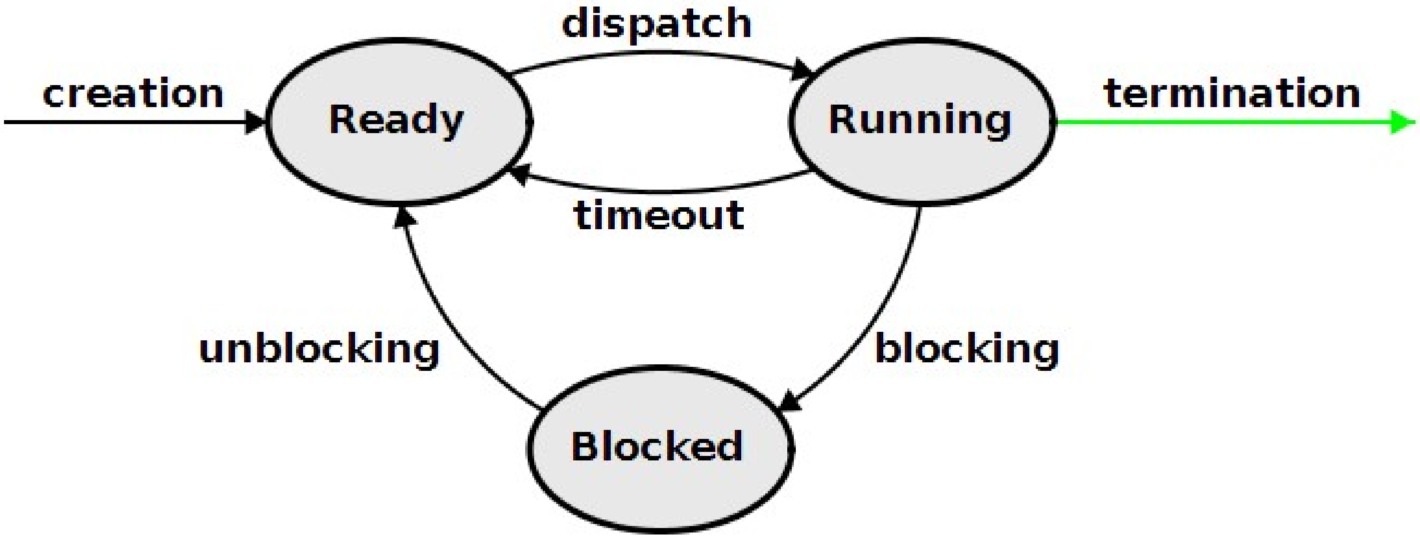
- Process executing consist of a cycle of CPU execution and I/O wait.
- Execution begins with a CPU burst and followed by an I/O burst, which is again followed by a CPU burst and so on.
- Eventually, the final CPU burst ends with a system request to terminate execution.
- CPU burst duration : vary from process to process. Curve is generally characterized as exponential or hyper-exponential with large number of short CPU bursts and a small number of long CPU bursts.
    - I/O-bound program – many short CPU bursts
    - CPU-bound program- few long CPU bursts



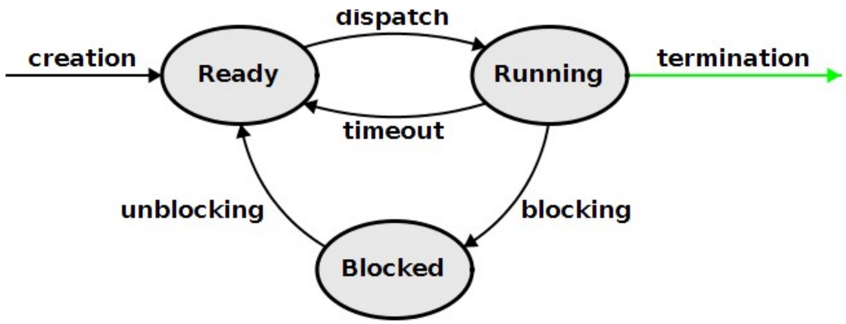
Histogram of CPU-burst durations



Life cycle of a process

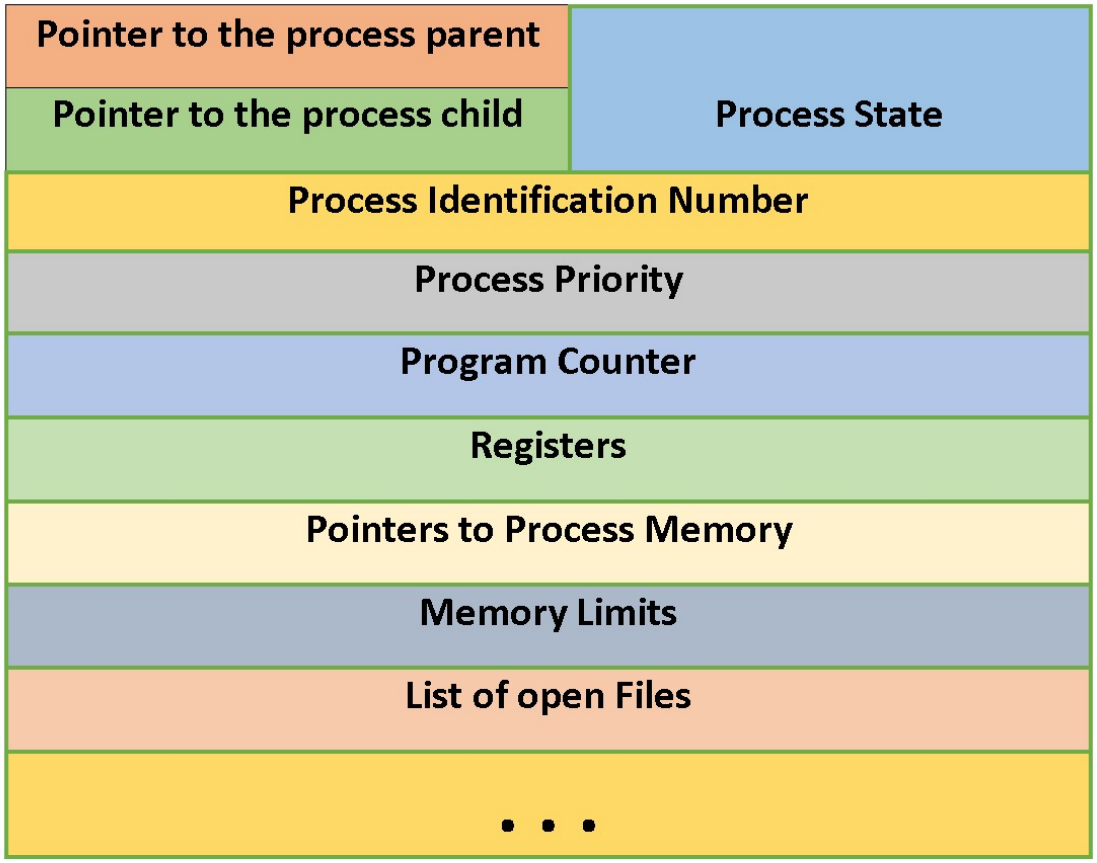


CPU Scheduler



- When CPU becomes idle, OS must select one of the processes in the ready queue to be executed.
- Selecting the next process is carried out short-term scheduler / CPU scheduler.
- Scheduler selects a process from the processes in memory that are ready to execute and allocates CPU to selected process.
- Ready queue – not necessarily a first-in, first-out (FIFO) queue.
    - Can be implemented as a FIFO queue, a priority queue, a tree or an unordered list.
- Conceptually, all processes in the ready queue are waiting for chance to run on CPU.
    - Process control block (PCB) contains related data of each process.

Process Control Block (PCB)



Pointer to the process parent

Pointer to the process child

Process State

Process Identification Number

Process Priority

Program Counter

Registers

Pointers to Process Memory

Memory Limits

List of open Files

. . .

Preemptive/ Non-preemptive Scheduling

- Non-Preemptive scheduling / cooperative scheduling

Once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state. There is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.

- Preemptive scheduling Process can be switched from running state to ready state upon receive higher priority task.

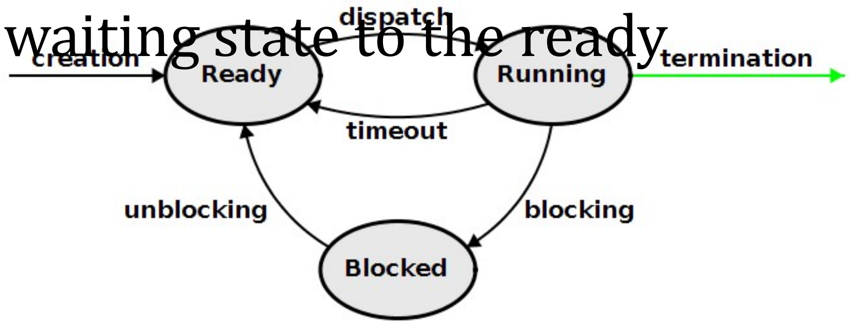
Preemptive/ Non-preemptive Scheduling

Non-Preemptive

1. When a process switches from the running state to the waiting state.
   - Eg: As the result of an I/O request or an invocation of wait for the termination of one of the child processes.
2. When a process terminates.

Preemptive

3. When a process switches from the running state to the ready state.
   - Eg: When an interrupt occurs.
4. When a process switches from the waiting state to the ready state.
   - Eg: at completion of I/O.



Non-preemptive vs. Preemptive scheduling

- Non-preemptive scheduling : introduced in Windows 3.x
- Preemptive scheduling : introduced in Window 95
  - Incurs cost to access shared data.
    - Eg: consider case of two processes that share data
    - While one is updating data, its preempted, so 2nd process can run.
    - Then 2nd process tries to read data : which is in an inconsistent state.
    - These shared data needs have coordinated access mechanisms, discuss under process synchronization.
  - Affects the design of OS kernel.
    - During a system call, kernel may busy with an activity of a process (that involves changing important kernel data i.e : eg: I/O queues), what if process get preempted in middle of changes and kernel needs to read or modify same data structure.
      - Eg: UNIX deals with such problems by waiting either for system call to complete or for I/O block to take place prior to a context switch.
  - Interrupts occurring during crucial OS activities

Dispatcher

- Involved in CPU scheduling.
- Is a module that gives control of the CPU's core to the process selected by the short-term scheduler.
- Should be fast
- Dispatch latency = difference of time it takes for dispatcher to stop one process and start another process running
- Functionalities
  - Context switching
  - Switching to user mode
  - Jumping to the proper location in the user program to restart the process.

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)

CPU scheduling algorithm optimization criteria

1. Max CPU utilization
2. Max throughput
3. Min turnaround time
4. Min waiting time
5. Min response time

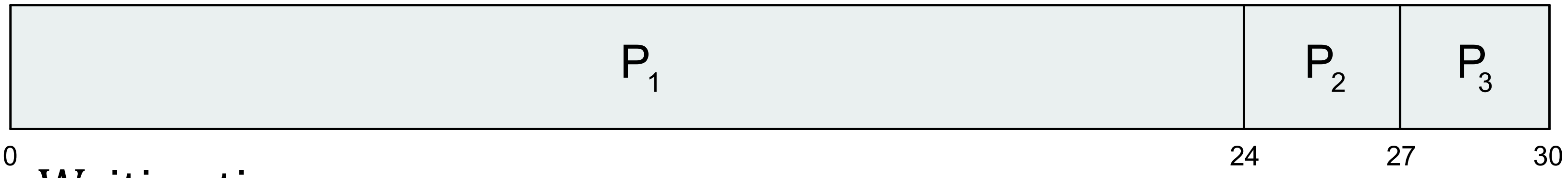
Scheduling algorithms : 1. First- Come, First-Served (FCFS) Scheduling

- It is a scheduling algorithm in which the process that arrives first is the one that gets executed first.
- The processes are executed in the order they arrive in the ready queue. The process that arrives first is selected for execution, and it continues until completion before the next process in the queue is chosen.
- When a process enters ready queue, PCB is linked on the tail of the queue. When CPU is free, tasks in the head of the queue is assigned.

Scheduling algorithms : 1. 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:



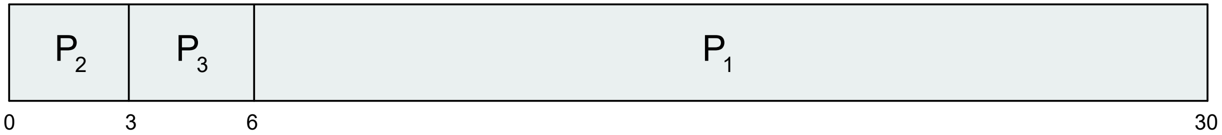
- Waiting time
  - for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time:
  - (0 + 24 + 27)/3 = 17

Scheduling algorithms : 1. First- Come, First-Served (FCFS) Scheduling (Cont.)

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
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

Convoy effect

- Occurs in scheduling algorithms, particularly in FCFS scheduling.
- Refers to a situation where shorter processes get delayed behind a long process.
- Significantly impact on the overall system performance when there is a mix of short and long processes.

Process	Burst Time (ms)
A	10
B	3
C	8

Order of execution would be A,B,C
A takes 10ms, then B takes 3 ms and C takes 8ms.
B is the shortest process; B needs to wait until longest A processes completes its execution. This waiting time for shorter process defines convoy effect.

Scheduling algorithms : 2. Shortest-job-first (SJF) Scheduling

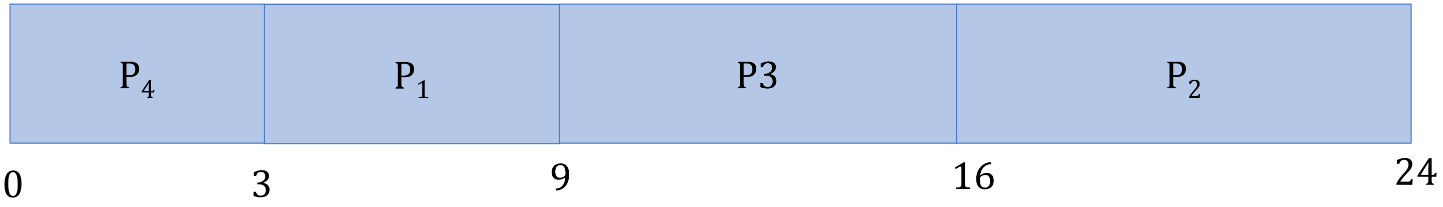
- Also known as Shortest Job Next (SJN)
- Selects the process with the shortest burst time for execution.
- It can be preemptive (Shortest Remaining Time First - SRTF) or non-preemptive.
- The idea is to minimize the waiting time and optimize the turnaround time by giving priority to processes with the shortest burst times.

Scheduling algorithms : 2. Shortest-job-first (SJF) Scheduling

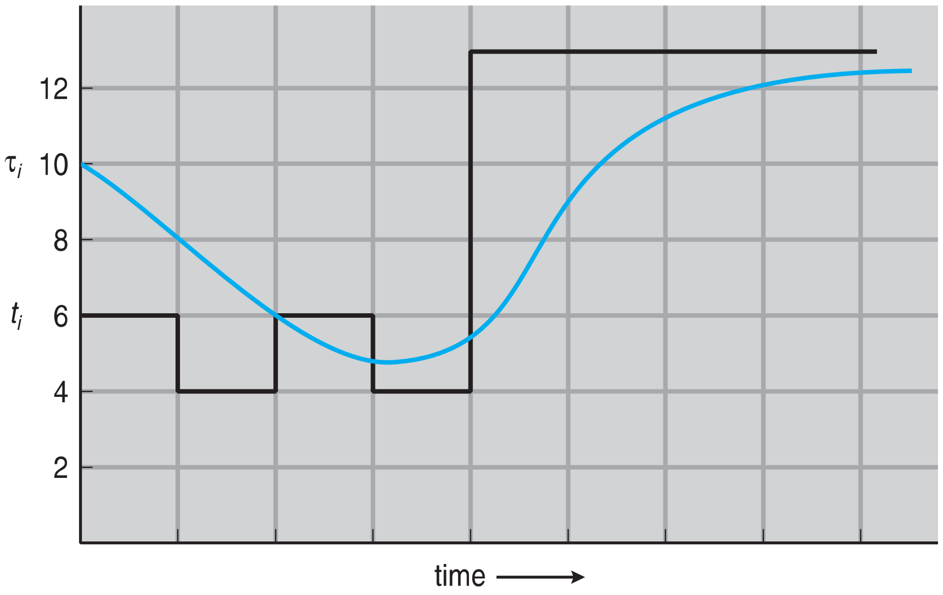
- Associates with each process the length of its next CPU burst
  - CPU is assigned to process with smallest next CPU burst
  - Preemptive version called **shortest-remaining-time-first**

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- Average waiting time = (0+3+9+16)/ 4 = 7 ms
- Gives minimum average waiting time.
- Difficult to know the length of the next CPU request.



Prediction of the length of the Next CPU burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Predicting the next CPU burst

1. Exponential Averaging
   - Use an exponential average to predict the next CPU burst time.
   - The formula for exponential averaging is:

   Predicted burst time = *α x Previous burst time + (1-α)x Previous prediction*
   - *α is a constant range between 0 and 1.*
2. Moving average
   - Use a moving average to predict the next CPU burst time.
   - The formula for moving average is: $predicted\ burst\ time = \frac{previous\ burst\ times}{n}$
   - n = #of previous burst time

Predicting the next CPU burst (Con’t)

3. Last CPU burst
    - Use the duration of the last observed CPU burst as the prediction for the next burst.
    - $Predicted\ Burst\ Time = Last\ CPU\ Burst$
4. Kalman filter

    More sophisticated technique, can be employed for dynamic prediction. Considers both current burst time and the prediction error.
5. Neural networks

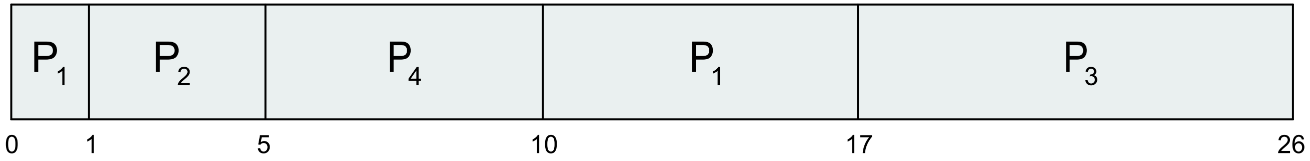
    Machine learning techniques such as NN can be trained on historical burst time data to predict future burst times.

Example of Shortest-remaining-time-first (SJF)

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 msec

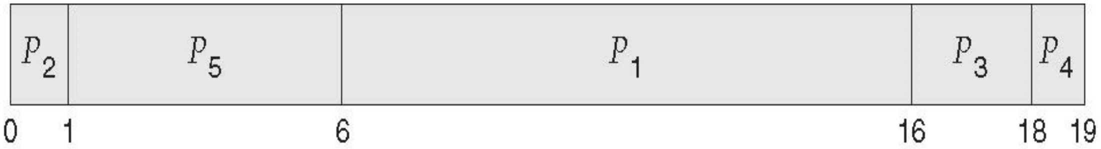
Scheduling algorithms : 3. Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time =((0+`+6+16+18)/5) = 8.2 msec

Scheduling algorithms : 4. Round Robin

- Preemptive scheduling algorithm
- Each process is assigned a fixed time slice / quantum
- CPU scheduler rotates among the processes allowing each to execute for a fixed amount of time.
- If process's burst time exceeds its time slide, its move back of the queue and next process in line gets a turn.

Process	Burst Time (ms)
A	8
B	5
C	10

Time quantum = 3ms
Execution order

A (3) 8-3	**B (3) 5-3**	**C (3) 10-3**	**A(3) 5-3**	**B(2) 2-2**	**C(3) 7-3**	**A(2) 2-2**	**C(5) 5**

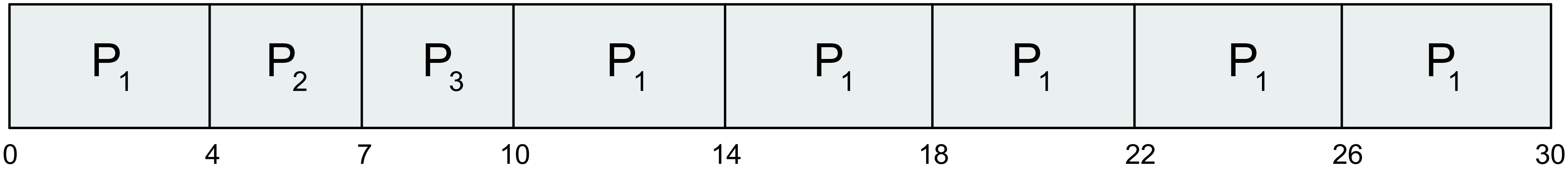
Scheduling algorithms : 4. Round Robin (cont'd)

- Each process gets a small unit of CPU time (**time quantum** q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with q = 4

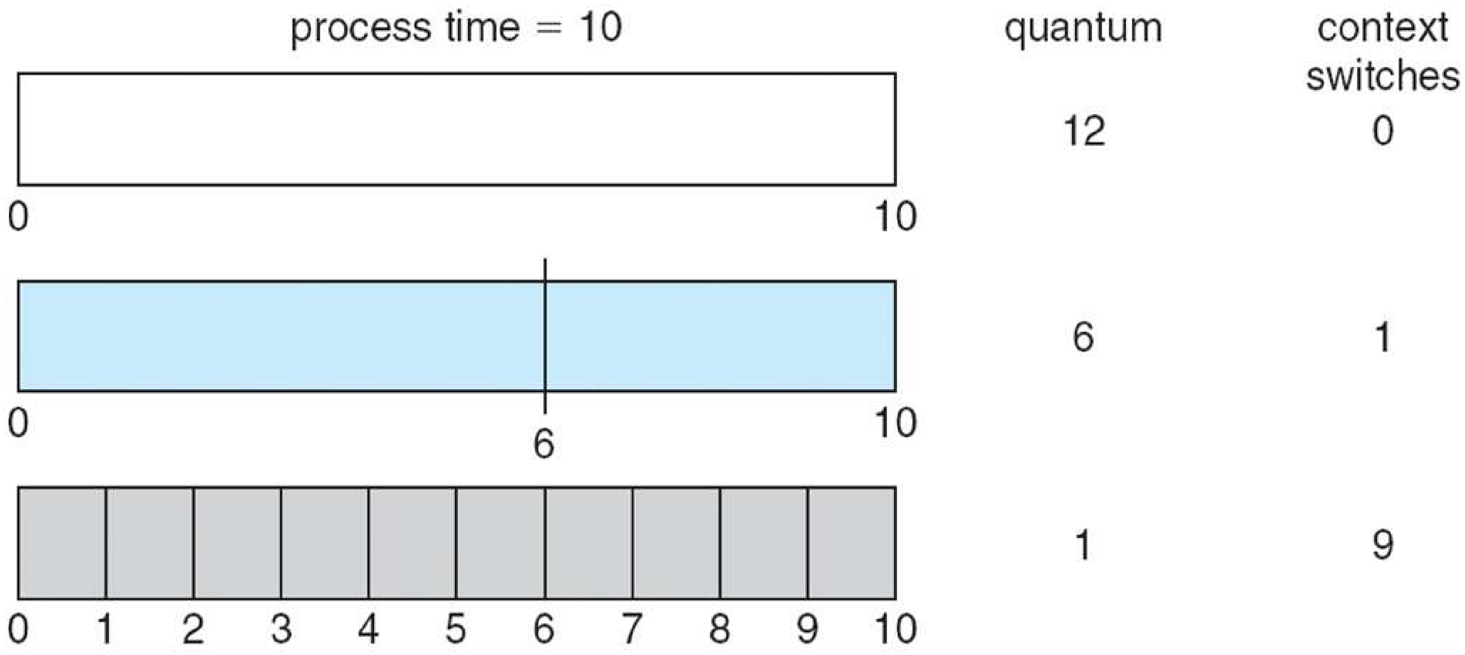
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

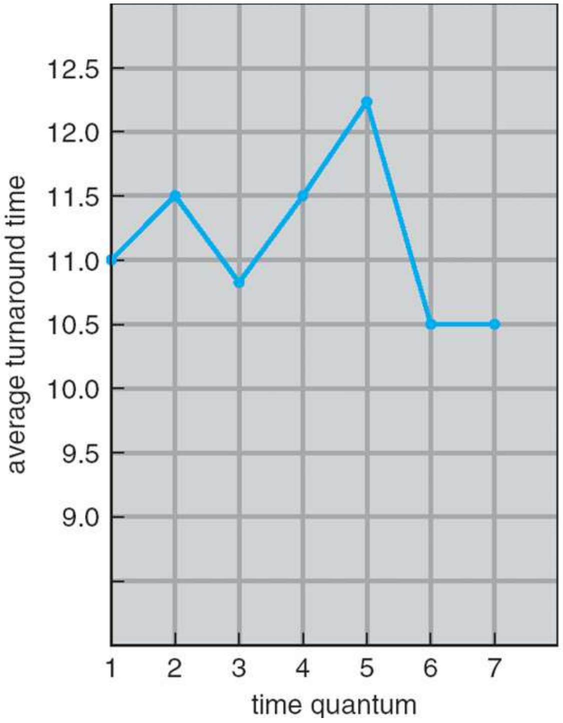


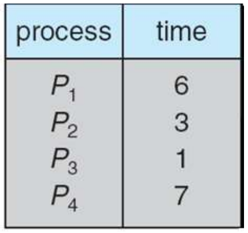
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time quantum and context switch time



Turnaround Time varies with the Time Quantum





process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q