Operating Systems:

Chaps 2, 3, 4

Scheduling Algorithms

CS400

Week 8: 3rd March

Spring 2020

Oliver BONHAM-CARTER

What’s Been Covered

- We have learned about:
  - Processes
    - Creating
    - Destroying
    - Blocking
  - Threads
    - Creating
    - Destroying
  - Semaphores
  - Memory management institutions
    - Maps
    - Paging systems

Bursts of Usage

- A “burst”: A brief stretch of “run as fast as you can go until you cannot continue for some reason.”
- A CPU bursts when it is executing instructions;
  - An I/O system bursts when it services requests to fetch information.
  - Each component operates until it cannot continue (for some reason).

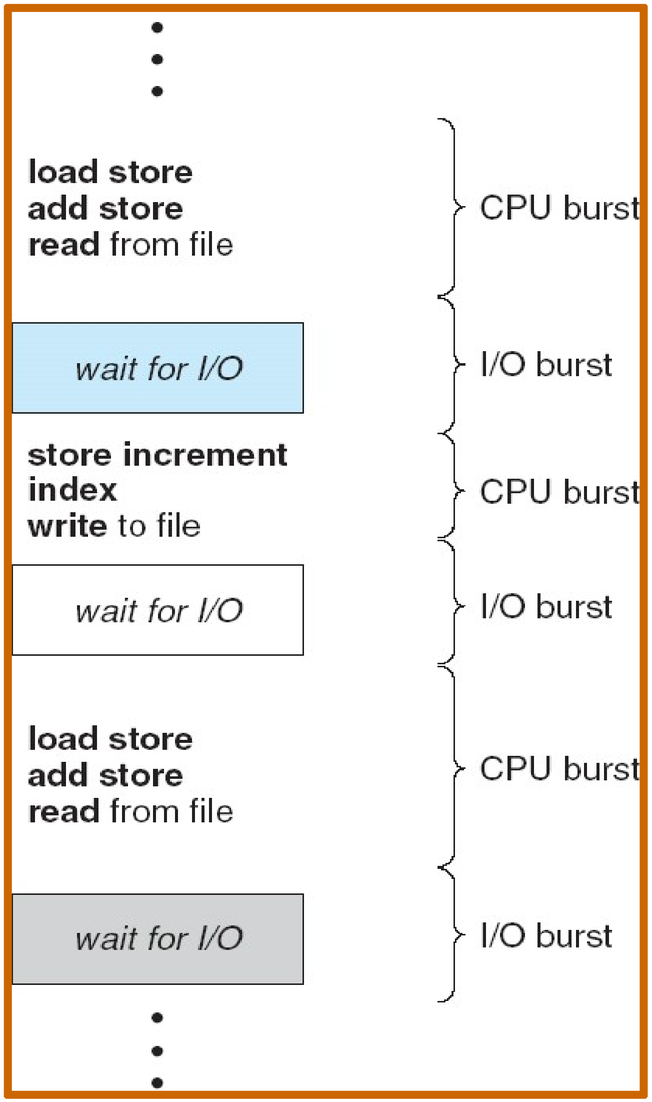
Bursts of CPU and IO Usage

- Start of CPU burst
  - A CPU can run instructions from cache until it needs to fetch more instructions or data from memory.
- Ends of CPU burst
- Starts the I/O burst.
  - The I/O burst read or writes data until the requested data is read/written or the space to store it cache runs out.
- End of an I/O burst.

Controlling *Bursts*:
The Magic of the OS

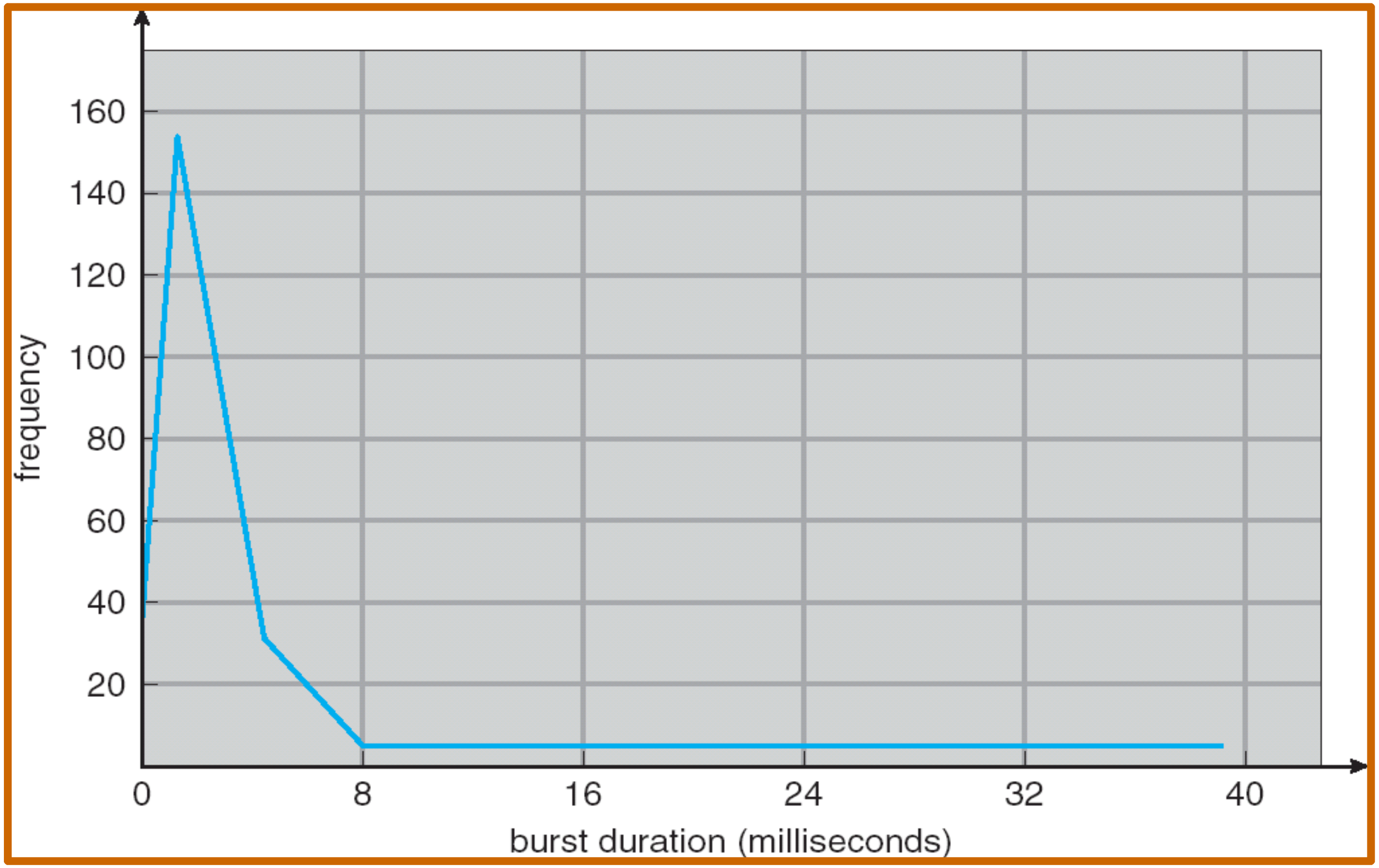
- "*The magic of an OS is the act of managing and scheduling these activities to maximize the use of the resources and minimize the wait and idle times for processes.*"

Alternating Sequence:
CPU and I/O ***Bursts***



- A typical execution of a program is made-up by bursts
- CPU: runs only one thing at a time
- I/O: Reads or writes to complete the instruction
- *All or nothing type of system*

Histogram of CPU-*Burst* Times

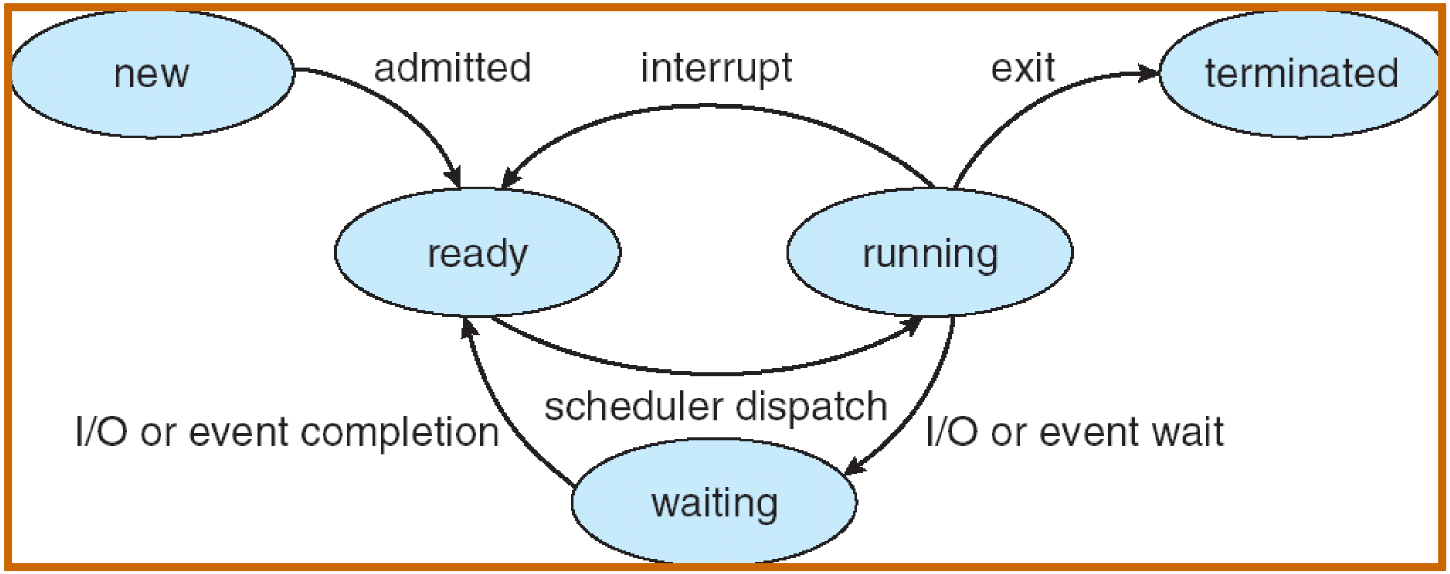


CPU bursts tend to have a frequency curve similar to the exponential curve shown above. It is characterized by a large number of short CPU bursts and a small number of long CPU bursts. An I/O-bound program typically has many short CPU bursts; a CPU-bound program might have a few long CPU bursts.

States of a Process

- **READY** - The process is waiting to be assigned to a processor.
- **RUNNING** - Instructions are being executed.
- **WAITING** - The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **TERMINATED** - The process has finished execution.

CPU Scheduler: Handling *Bursts*



- The CPU scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

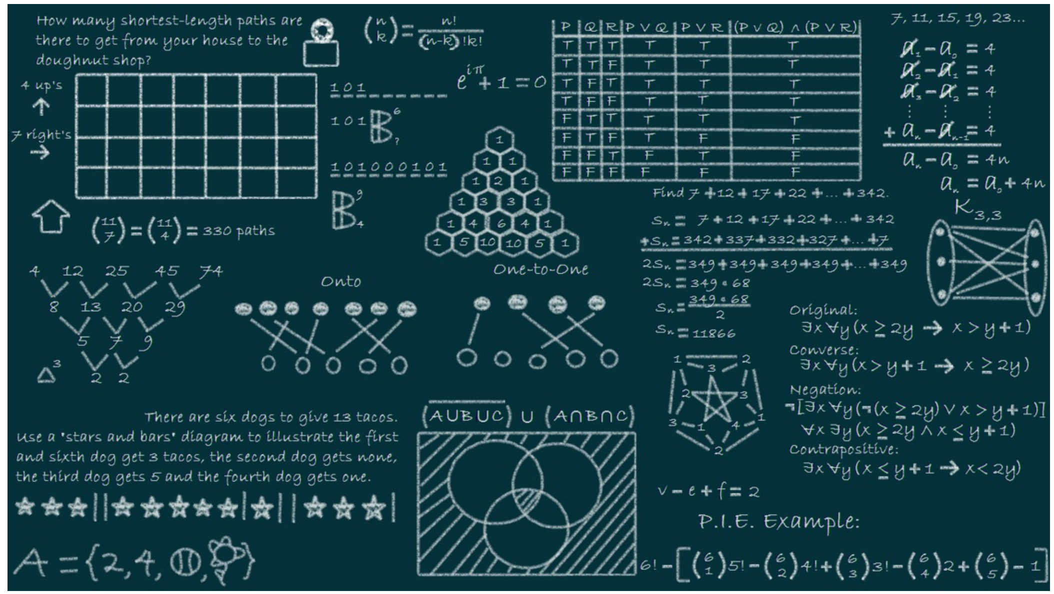
CPU Scheduler

- CPU scheduling is affected by the following set of circumstances:
  - A process switches from **running** to **waiting** state
  - A process switches from **running** to **ready** state
  - A process switches from **waiting** to **ready** state
  - A processes switches from **running** to **terminated** state
- Circumstances 1 and 4 are non-preemptive; they offer **no scheduling choice**
- Circumstances 2 and 3 are pre-emptive; they **can be scheduled**

Dispatcher

- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler; involving:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- The dispatcher needs to run as fast as possible, since it is invoked during process context switch
- The time it takes for the dispatcher to stop one process and start another process is called *dispatch latency*

Scheduling Criteria: Algorithms



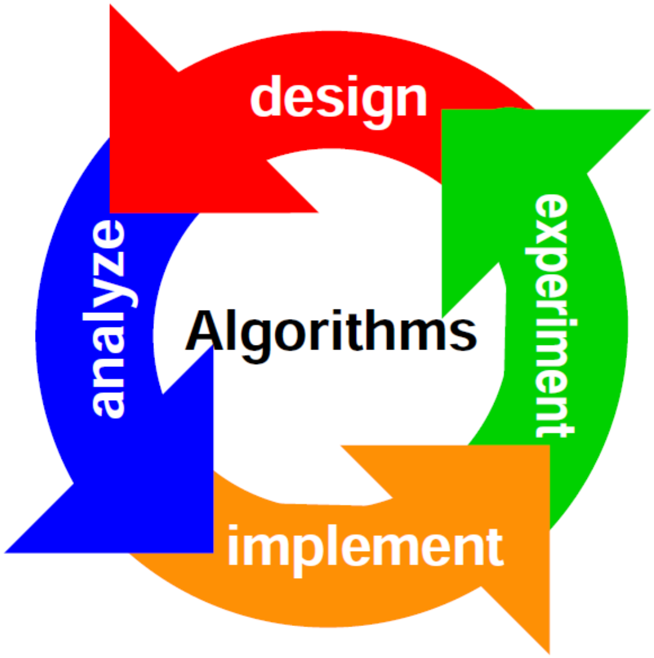
- Different CPU **scheduling algorithms** have different properties
- The choice of a particular algorithm may favor one class of processes over another
- In choosing which algorithm to use, the properties of the various algorithms should be considered

Scheduling Criteria: *response*, *waiting* and *turnaround*

- Criteria for comparing CPU scheduling algorithms may include the following
  - **CPU utilization** – percent of time that the CPU is busy executing a process
  - **Throughput** – number of processes that are completed per time unit
  - **Response time** – amount of time it takes from when a request was submitted until the first response occurs (but not the time it takes to output the entire response)
  - **Waiting time** – the amount of time before a process starts after first entering the ready queue (or the sum of the amount of time a process has spent waiting in the ready queue)
  - **Turnaround time** – amount of time to execute a particular process from the time of submission through the time of completion

Optimization Criteria

- It is desirable to
  - Maximize CPU utilization
  - Maximize throughput
  - Minimize turnaround time
  - Minimize start time
  - Minimize waiting time
  - Minimize response time



- In most cases, we strive to optimize the average measure of each metric
- In other cases, it is more important to optimize the minimum or maximum values rather than the average

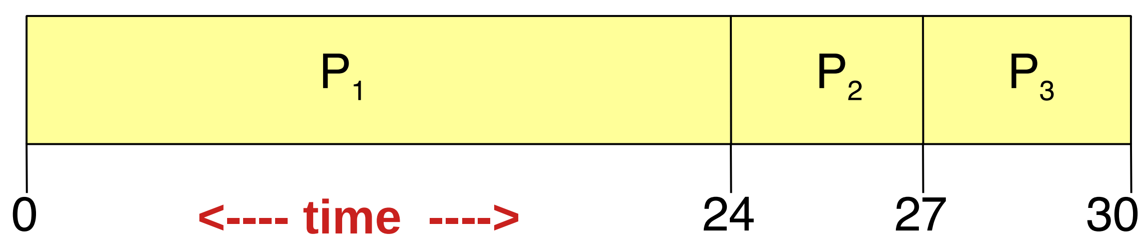
What Makes Scheduling Happen?

- Algorithms *(programming recipes for specific tasks)*
- Scheduling Algorithms:
  - First Come, First Served (FCFS)
  - Shortest Job First (SJF)
  - Priority
  - Round Robin (RR)

Algorithm: First Come, First-Served (1)

- Process: Burst Time
  - P1: 24
  - P2: 3
  - P3: 3
- With FCFS, the process that requests the CPU first is allocated the CPU first
- Case #1: Suppose that the processes arrive in the order: P1 , P2 , P3

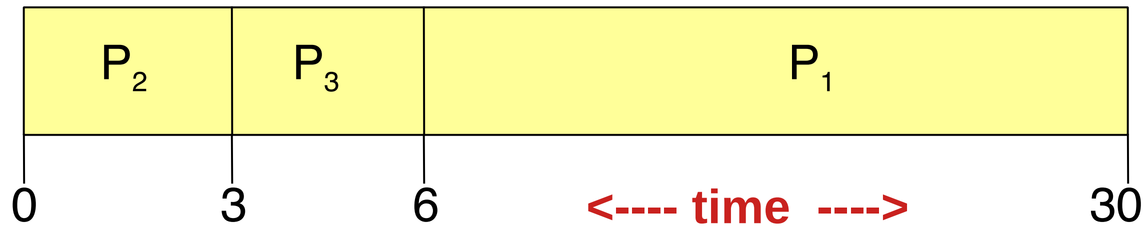
  The Gantt Chart for the schedule is:



- **Waiting time** for P1 = 0; P2 = 24; P3 = 27
- **Average waiting time**: (0 + 24 + 27)/3 = 17
- **Average turn-around time**: (24 + 27 + 30)/3 = 27

Algorithm: First Come, First-Served (2)

- Case #2: Suppose that the processes arrive in the order: P2 , P3 , P1
- The Gantt chart for the schedule is:



- **Waiting time** for P1 = 6; P2 = 0; P3 = 3
- **Average waiting time**: (6 + 0 + 3)/3 = 3 (Much better than Case #1: 17)
- **Average turn-around time**: (3 + 6 + 30)/3 = 13 (case #1: 27)
- Case #1 is an example of the ***convoy effect***;
  - all the other processes wait for one long-running process to finish using the CPU
- This problem results in lower CPU and device utilization; Case #2 shows that higher utilization might be possible if the short processes were allowed to run first
- The FCFS scheduling algorithm is non-preemptive (non-scheduling)
- Once the CPU has been allocated to a process, that process keeps the CPU until it releases it either by terminating or by requesting I/O
- It is a troublesome algorithm for time-sharing systems (*users and devices* are hard to control!)

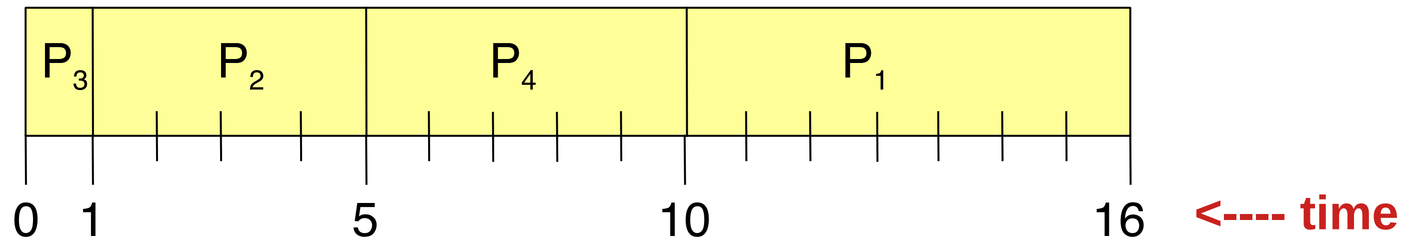
Algorithm: Shortest Job First

- The SJF algorithm associates with each process the length of its next CPU burst
- When the CPU becomes available, it is assigned to the process that has the smallest next CPU burst (in the case of matching bursts, FCFS is used)
- Two schemes:
  - **Nonpreemptive** – once the CPU is given to the process, it cannot be preempted until it completes its CPU burst
  - **Preemptive** – if a new process arrives with a CPU burst length less than the remaining time of the current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)

Algorithm: Shortest Job First Ex1 (simultaneous arrival)

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	0.0	4
P_3	0.0	1
P_4	0.0	5

- SJF (non-preemptive, simultaneous arrival)



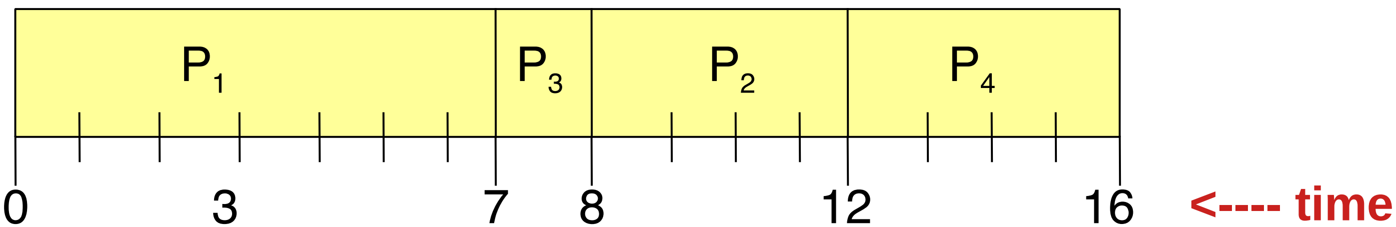
- **Average waiting time** = (0 + 1 + 5 + 10)/4 = 4
- **Average turn-around time** = (1 + 5 + 10 + 16)/4 = 8

Algorithm: SJF Ex2 (non-preemptive, varied arrival times)

- Non-preemptive (no stalling)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive, varied arrival times)



- **Average waiting time**

P1 + P2 + P3 + P4

$= ((0 – 0) + (8 – 2) + (7 – 4) + (12 – 5))/4$

$= (0 + 6 + 3 + 7)/4 = 4$

- **Average turn-around time**:

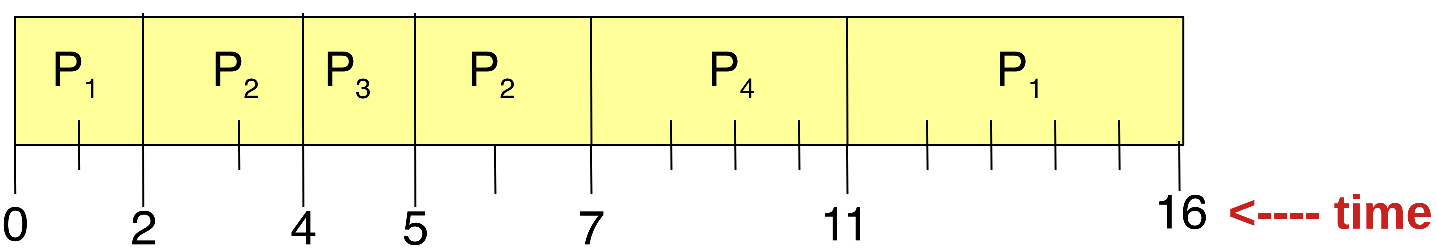
$= ((7 – 0) + (12 – 2) + (8 - 4) + (16 – 5))/4$

$= (7 + 10 + 4 + 11)/4 = 8$

Preemptive SJF
(shortest-remaining time first)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive, varied arrival times)



- **Average waiting time**

  = ([(0 – 0) + (11 - 2)] + [(2 – 2) + (5 – 4)] + (4 - 4) + (7 – 5))/4

  = 9 + 1 + 0 + 2)/4

  = 3

- **Average turn-around time** = (16 + 7 + 5 + 11)/4 = 9.75

Algorithm: Priority (1)

- **The SJF algorithm is a *special case* of the general priority scheduling algorithm**
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
- Priority scheduling can be either *preemptive* or *non-preemptive:*
  - A **preemptive** approach will preempt the CPU if the priority of the newly-arrived process is higher than the priority of the currently running process
  - A **non-preemptive** approach will simply put the new process (with the highest priority) at the head of the ready queue

Algorithm: Priority (2)

- SJF is a priority scheduling algorithm where priority is the predicted next CPU burst time
- The main problem with priority scheduling is *starvation*, that is, low priority processes may never execute
- A solution is *aging*; as time progresses, the priority of a process in the ready queue is increased

Algorithm: Round Robin

- In the round robin algorithm, each process gets a small unit of CPU time (a time quantum), 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-1)q time units.*
- Performance of the round robin algorithm
  - *q* large --> Simply use: the First Come, First Served algorithm
  - *q* small --> *q* must be greater than the context switch time; otherwise, the overhead is too high
- Typically: 80% of the CPU bursts should be shorter than the time quantum
- *Def of time quantum: The period of time for which a process is allowed to run uninterrupted in a pre-emptive multitasking operating system*

Round Robin Ex, Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162 **<---- time**

- Typically, <u>higher</u> average turnaround than SJF, but <u>better</u> *response time*
- **Average waiting time**
  = ([(0 – 0) + (77 - 20) + (121 – 97)] + (20 – 0) + [(37 – 0) + (97 - 57) + (134 – 117)] + [(57 – 0) + (117 – 77)]) / 4
  = (0 + 57 + 24) + 20 + (37 + 40 + 17) + (57 + 40)) / 4
  = (81 + 20 + 94 + 97)/4
  = 292 / 4 = 73
- **Average turn-around time**
- = 134 + 37 + 162 + 121) / 4 = 113.5

Nice Scheduling Demo

- https://ess.cs.tu-dortmund.de/Software/AnimOS/CPU-Scheduling/