OPERATING SYSTEMS

Topic-Process Management

Process

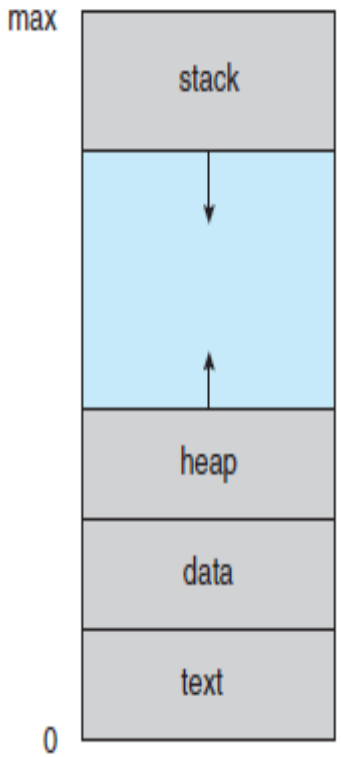
- A process is ***an Operating System abstraction for a running program***
- A process is a ***program in execution***
- A Process is ***associated with an address space***
- A Thread is *a running program without its own address spaCe*

Process

A process consists of the following:

- Text Section: contains the *program code* and also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.
- Static Data Section: contains global variables.
- Stack: contains temporary data (such as function parameters, return addresses, and local variables)
- Heap: it is memory that is dynamically allocated during process run time (don't confuse with data structure Min-Heap and Max-Heap)

Process in memory



Program versus Process

A program by itself is not a process.

- A program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file).
- A process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

A program becomes a process when its executable file is loaded into memory

Process States

As a process executes, it changes state. The state of a process is defined by the current activity of that process.

A process may be in one of the following states:

New: The process is being created.

A process is said to be in new state when a program present in the secondary memory is initiated for execution

Ready: The process is waiting to be assigned to a processor.

A process moves from new state to ready state after it is loaded into the main memory and is ready for execution. In ready state, the process waits for its execution by the processor. In multiprogramming environment, many processes may be present in the ready state.

Process State (cont.)

Running: Instructions are being executed.

A process moves from ready state to run state after it is assigned the CPU for execution

Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).

A process moves from run state to block or wait state if it requires an I/O operation or some blocked resource during its execution. After the I/O operation gets completed or resource becomes available, the process moves to the ready state

Terminated: The process has finished execution.

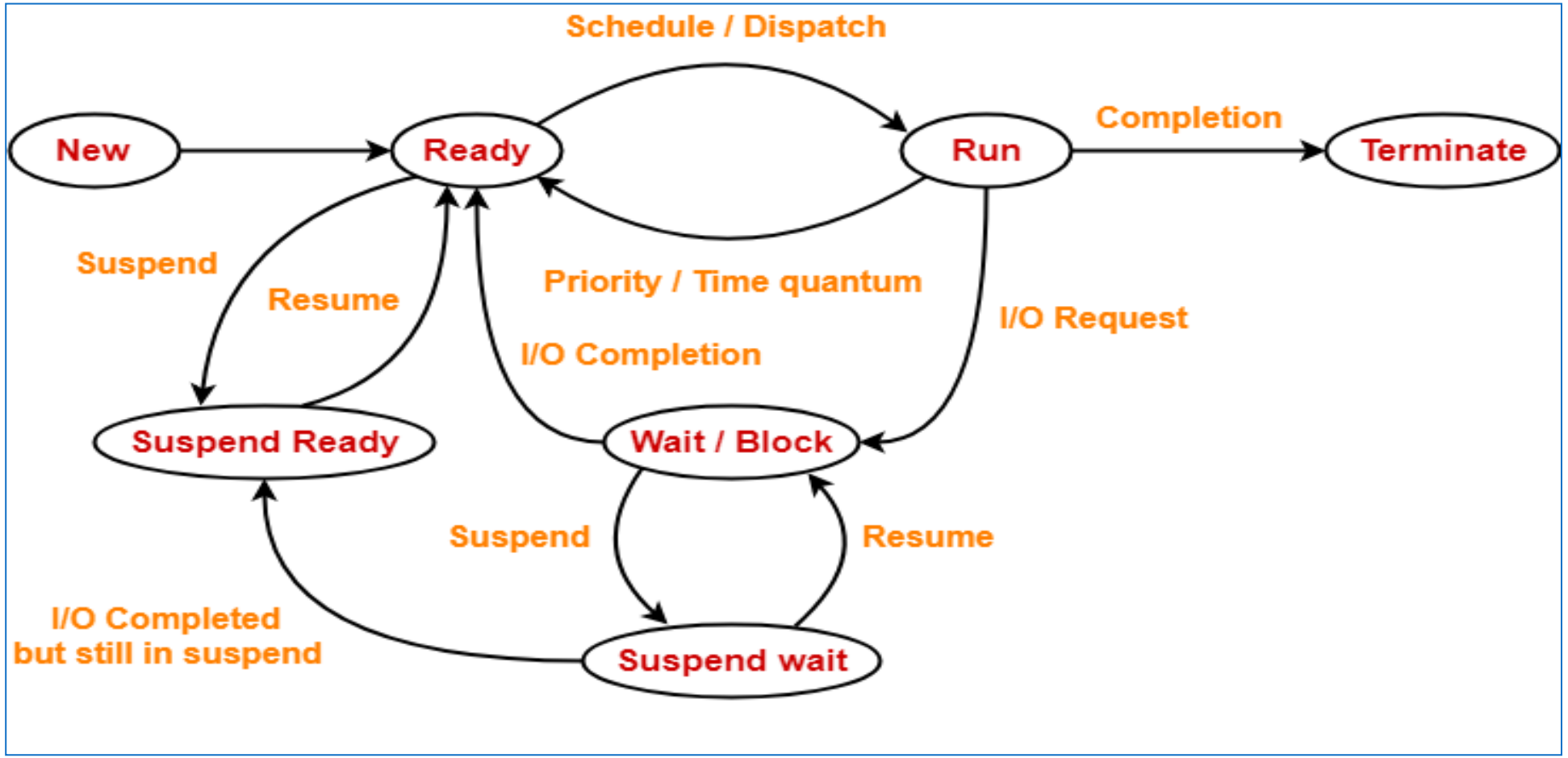
A process moves from run state to terminate state after its execution is completed. After entering the terminate state, context (PCB) of the process is deleted by the operating system.

Process States (cont.)

Suspend Ready: A process moves from ready state to suspend ready state if a process with higher priority has to be executed but the main memory is full. Moving a process with lower priority from ready state to suspend ready state creates a room for higher priority process in the ready state. The process remains in the suspend ready state until the main memory becomes available. When main memory becomes available, the process is brought back to the ready state.

Suspend Wait: A process moves from wait state to suspend wait state if a process with higher priority has to be executed but the main memory is full. Moving a process with lower priority from wait state to suspend wait state creates a room for higher priority process in the ready state. After the resource becomes available, the process is moved to the suspend ready state. After main memory becomes available, the process is moved to the ready state.

Process State Transition Diagram



Process Control Block

The OS groups all information that it need about a particular process into a data structure called a *Process Descriptor* or *Process Control Block* (PCB).

- These information in PCB is required by the CPU while executing the process and is also called as **context** of the process.
- Each process is identified by its own *Process Control Block* (PCB)
- Whenever a process is created, the OS creates its corresponding PCB
- When the process terminates, its PCB is released.
- A process become known to OS and thus eligible to compete for system resources, only when it has an active PCB associated with it

The Information Stored in PCB

Process Id: Process Id is a unique Id that identifies each process of the system uniquely and is assigned to each process during its creation.

Program Counter: Program counter specifies the address of the next instruction to be executed. Before starting the execution, program counter is initialized with the address of the first instruction of the process. After executing an instruction, value of program counter is automatically incremented to point to the next instruction.

Process State: Each process goes through different states during its lifetime and Process state specifies the current state of the process.

Priority: Priority specifies how urgent is to execute the process. Process with the highest priority is allocated the CPU first among all the processes.

The Information Stored in PCB (cont.)

General Purpose Registers: General purpose registers are used to hold the data of process generated during its execution. Each process has its own set of registers which are maintained by its PCB.

File management information: Each process requires some files which must be present in the main memory during its execution. PCB maintains a list of files used by the process during its execution with access rights.

I/O status: Allocated devices and pending requests.

Process Scheduling

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.

To meet these objectives, *the process scheduler selects an available process (possibly from a set of several available processes) for execution on the CPU*.

For a single-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled

Scheduling refers to the set policies and mechanism built in to operating system that govern the order in which work to be done by a computer system will be completed

Scheduler

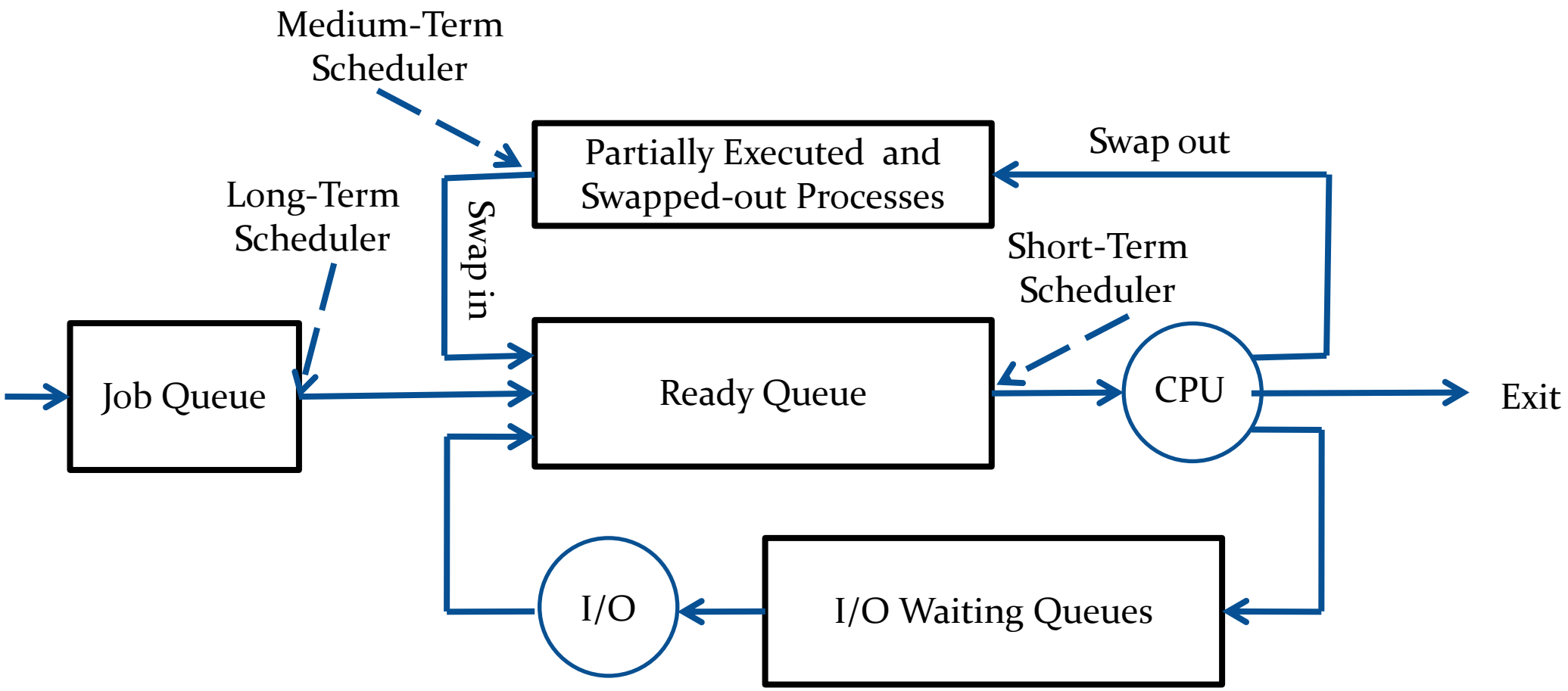
Scheduler: It is an operating system module that selects next job to be admitted in to the system and next process to run.

Objective of the Scheduler: To optimize system performance in accordance with the criteria deemed to be most important by the OS designer

Types of Schedulers : There are three types of schedulers

- Long-term Scheduler
- Medium-term scheduler
- Short-term scheduler

Types of Scheduler



Scheduling Queues

Job Queue: As processes enter the system, they are put into a job queue, which consists of all processes in the system.

Ready Queue: The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.

- This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.

Device Queue: The process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a device queue.

- Each device has its own device queue

Queues, States and PCB

- The OS maintains a set of queues that represent the state of processes in the system
  - e.g., ready queue: all runnable processes
  - e.g., wait queue: processes blocked on some condition
- Process migrates among various queues throughout its life-time
- As a process changes state, its PCB is unlinked from one queue, and linked onto another
- The scheduler must select these processes in some fashion for scheduling purpose The selection process is carried out by the appropriate scheduler

Long-term scheduler

Long-term scheduler is also known as **Job Scheduler**. In a batch system, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device, where they are kept for later execution.

- Batch jobs contains all necessary data and commands for their execution and also contains programmer or system-assigned estimate of the resource requirement.

The objective of the **long-term scheduler is to** selects a good balanced mix of I/O bound and CPU bound processes from the secondary memory.

- An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
- A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.

Long-term scheduler (cont.)

If all processes are I/O bound, the ready queue will almost always be empty

If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused

For the best performance will thus have a proper combination of CPU-bound and I/O-bound processes.

The long-term scheduler executes much less frequently and controls the degree of multiprogramming (the number of processes in memory)

Medium-term scheduler

- A running process may become suspended by making an I/O request or by issuing a system call or by creating a new sub-process and wait for its completion. The suspended process cannot make progress towards completion until the suspension condition is removed
- The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove them from memory to make room for other processes . Saving the image of suspended process in secondary memory is called swapping and the process is said to be swapped out or rolled-out.
- The medium-term scheduler is in-charge of handling the swapped-out processes. It has do little while a process remains suspended, however once the suspension condition is removed, the medium term scheduler attempts to allocate the required amount of main memory and make it ready.

Short-term Scheduler

- Short-term scheduler is also known as **CPU Scheduler**.
- It decides which process to execute next from the ready queue.
- After short-term scheduler decides the process, **Dispatcher** assigns the decided process to the CPU for execution.
- The short-term scheduler must select a process from the ready queue quite frequently. A process may execute only for a few milliseconds before waiting for I/O request

Scheduling Criteria

- **CPU Utilization**
  - ↑ Keep the CPU and other resources as busy as possible
- **Throughput**
  - ↑ Number of processes that complete their execution per unit time.
- **Turnaround Time**
  - ↓ Amount of time to execute a particular process from its submission time to completion time.
- **Waiting Time**
  - ↓ Amount of time a process has been waiting in the ready queue.
- **Response Time** (in a time-sharing environment)
  - ↓ Amount of time it takes from when a request was submitted until the first response is produced (Not output).

Optimization Criteria

- To optimize Overall System Performance
  - ✓ Maximize CPU Utilization
  - ✓ Maximize Throughput

- To optimize Individual Processes' Performance
  - ✓ Minimize Turnaround time
  - ✓ Minimize Waiting time
  - ✓ Minimize Response time

First Come First Serve (FCFS) Scheduling

Policy: The process which arrives first in the ready queue is firstly assigned the CPU. In case of a tie, process with smaller process id is executed first.

- FCFS is always non-preemptive in nature - once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU
- Gantt Charts is used to visualize schedules.

Performance Metric:

- Evaluated on the parameters of *Average Waiting Time* in queue and the *Average Turnaround Time.*

Implementation

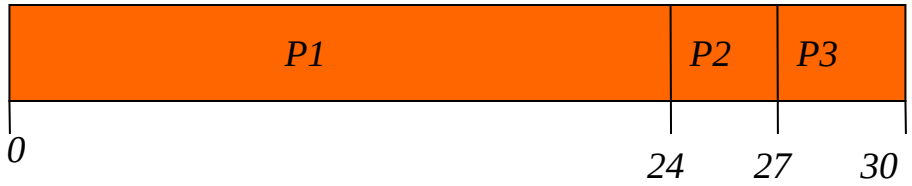
- using FIFO queue and it is easy to understand and to implement.
- When a process enters the ready queue incoming process is added to the tail of the queue, its PCB is linked to the rear of the queue.
- When CPU becomes free , it is allocated to the Process at the front of the queue and the running process is removed from the ready queue

Performance in terms of average waiting time of the processes

Example Problem-1

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for FCFS Scheduling



Turnaround Time = Exit Time – Arrival Time

Waiting Time = Turn Around Time – Burst Time

Suppose the arrival order for the processes is P1, P2, P3 and at same time

Turnaround Time

P1=24-0=24

P2=27-0=27

P3=30-0=30

Average Turnaround time

(24+27+30)/3 = 27

Waiting time

P1 = 24-24=0;

P2 = 27-3=24;

P3 = 30-3=27;

Average waiting time

(0+24+27)/3 = 17

Performance in terms of average waiting time of the processes (cont.)

Example Problem-2

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for FCFS Scheduling

Turnaround Time = Exit Time – Arrival Time
Waiting Time = Turn Around Time – Burst Time

Suppose the arrival order for the processes is P3, P2, P1 and at same time

Turnaround Time

P1=30-0=30

P2=6-0=6

P3=3-0=3

Average Turnaround time

(30+6+3)/3 = 13 (best)

Waiting time

P1 = 30-24=6;

P2 = 6-3=3;

P3 = 3-3=0;

Average waiting time

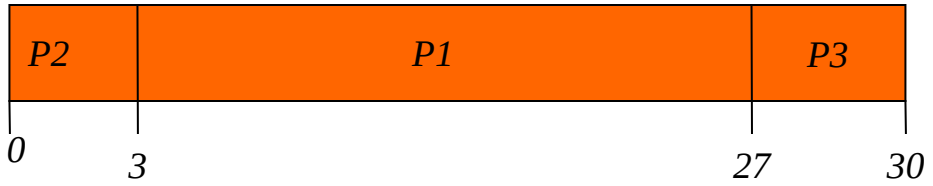
(6+3+o)/3 = 3 (best)

Performance in terms of average waiting time of the processes(cont.)

Example Problem-3

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for FCFS Scheduling



Turnaround Time = Exit Time – Arrival Time
Waiting Time = Turn Around Time – Burst Time

Suppose the arrival order for the processes is P3, P2, P1 and at same time

Turnaround Time

P1=27-0=27

P2=3-0=3

P3=30-0=30

Average Turnaround time

(27+3+30)/3 = 20 (better)

Waiting time

P1 = 27-24=3;

P2 = 3-3=0;

P3 = 30-3=27;

Average waiting time

(3+0+27)/3 = 10 (better)

Performance in terms of CPU utilization

Assume that there is one CPU-bound process and many I/O-bound processes

- When the CPU-bound process gets the CPU and hold it, the I/O-bound process will finish their I/O and wait for CPU in the ready queue. I/O devices are idle
- When the CPU-bound process will wait for I/O, the I/O-bound processes which have shortest CPU bursts execute quickly and move i/o device queues. CPU sits idle
- When one big process holds the CPU, the other processes have wait in the ready queue until CPU becomes free. This is called convey effect.
- This results in lower CPU and I/O device utilization.

Advantage & Disadvantage of FCFS scheduling

Advantages

- ✓ It is simple and easy to understand.
- ✓ It can be easily implemented using queue data structure.
- ✓ It does not lead to starvation.

Disadvantage

- ✖ It does not consider the priority or burst time of the processes.
- ✖ It suffers from *convoy effect*. (Convoy Effect: *Short processes behind a long process, e.g. 1 CPU bound process, many I/O bound processes.*)
- ✖ FCFS scheduling is not suitable for time-sharing system where it is must for each process get a share of the CPU at regular interval.

Shortest-Job-First (SJF) Scheduling

Policy: Associates with each process the length of its next CPU burst time. When the CPU becomes available, it will be assigned with the process that has the shortest next CPU burst time. When more than one process have same length, FCFS is used to break the tie.

SJF Scheduling algorithm can be either preemptive or non-preemptive and this choice arise when a new process arrive at ready queue while a current process is executing.

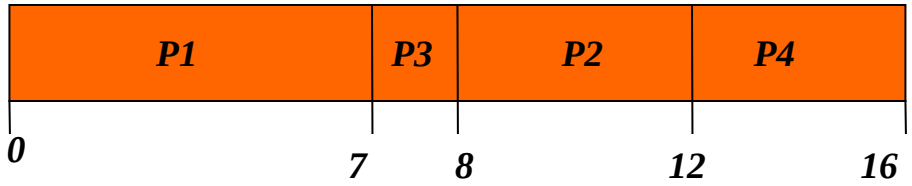
- **Non-preemptive SJF:** Once CPU is given to the process it cannot be preempted until it completes its current CPU burst.
- **Preemptive SJF:** If a new CPU process arrives with CPU burst length less than remaining CPU burst time of current executing process then preempt currently executing process and allocate the new process with shortest CPU burst to CPU. This is also called Shortest-Remaining-Time-First (SRTF)

Problem on Non-Preemptive SJF Scheduling

Consider the table

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for Non-preemptive SJF Scheduling



Turnaround Time = Exit Time – Arrival Time
Waiting Time = Turn Around Time – Burst Time

Turnaround Time

P1=7-0=7

P2=12-2=10

P3=8-4=4

P4=16-5=11

Average Turnaround time

(7+10+4+11)/4 = 32/4=8

Waiting time

P1 = 7-7=0;

P2 = 10-4=6;

P3 = 4-1=3;

P4=11-4=7

Average waiting time

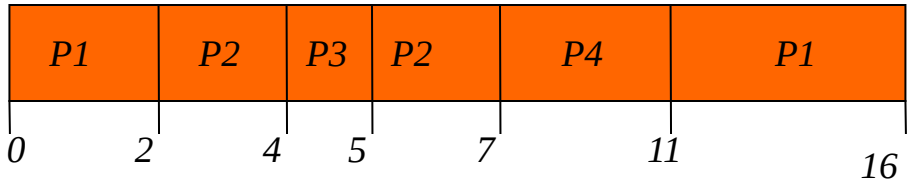
(0+6+3+7)/4 = 16/4=4

Problem on Preemptive SJF Scheduling (SRTF)

Consider the table

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for Non-preemptive SJF Scheduling



Turnaround Time = Exit Time – Arrival Time
Waiting Time = Turn Around Time – Burst Time

Turnaround Time

P1=16-0=16

P2=7-2=5

P3=5-4=1

P4=11-5=6

Average Turnaround time

(16+5+1+6)/4 = 28/4=7 (optimal)

Waiting time

P1 = 16-7=9;

P2 = 5-4=1;

P3 = 1-1=0;

P4=6-4=2

Average waiting time

(9+1+0+2)/4 = 12/4=3 (optimal)

Advantage & Disadvantage of SRTF scheduling

Advantages

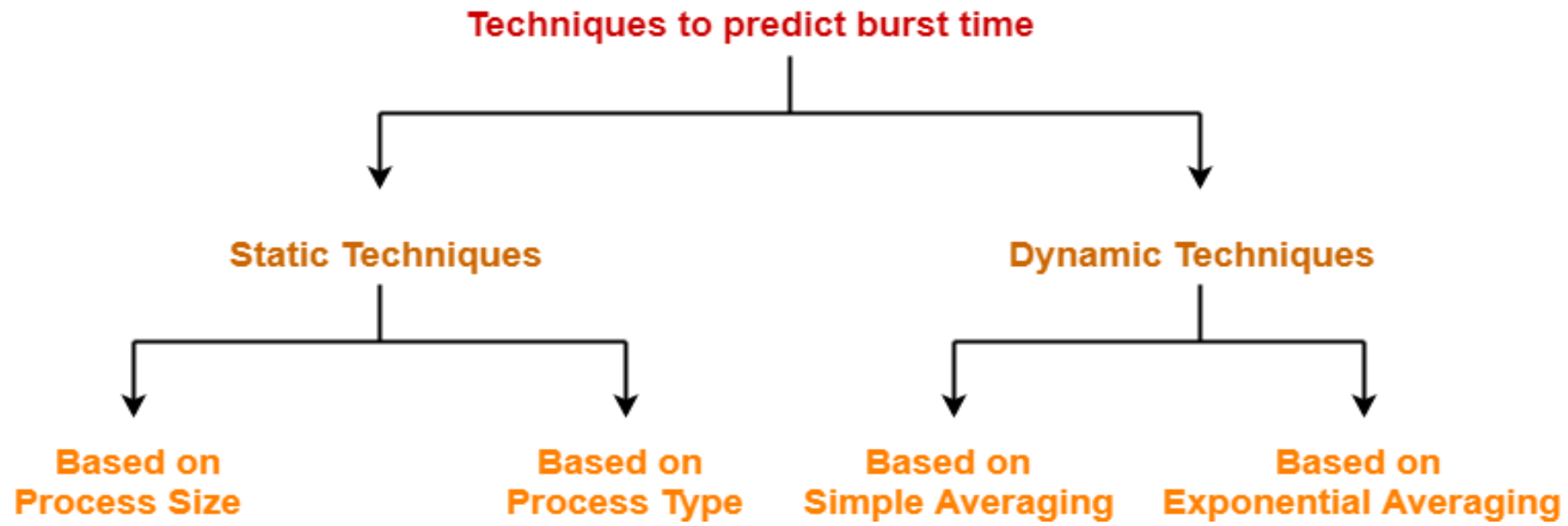
- ✓ SRTF scheduling guarantees the minimum average waiting time for a given set of processes and is optimal . The minimum average wait time is achieved by decreasing the waiting time for short processes and increasing the waiting time for long processes.
- ✓ It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages

- × It can not be implemented practically since burst time of the processes can not be known in advance. One can only estimate the length of next CPU burst
  - ❖ Halting problem: cannot (in general) determine if a program will run forever on some input or complete in finite time
- × It leads to starvation for processes with larger burst time.
- × Priorities can not be set for the processes.
- × Processes with larger burst time have poor response time.

Estimating Length of Next CPU Burst

The real difficulties in implementing SJF Scheduling is determining the length of the next CPU burst and can not be known in advance. The only possibility is to predict burst time.

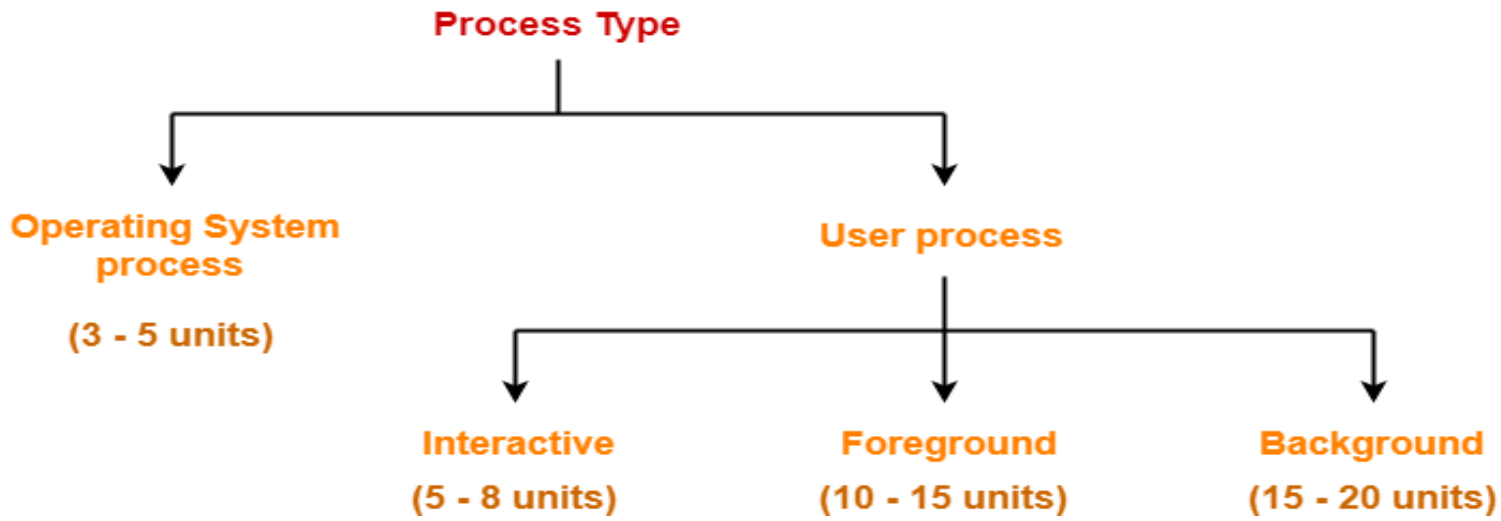


Static Techniques

These techniques are based either on process size 0r on process type and predicted before the process execution.

Process Size Based Technique: Burst time of the already executed process of similar size is taken as the burst time for the process to be executed. This predicted burst time may not always be right.

Process Type Based Technique: The burst time is assumed for different kinds of processes



Dynamic Techniques

These based either on simple averaging 0r on exponential averaging and predicted during the process execution.

Simple Averaging Technique

The next burst time for the process is estimated as the average of all the burst time that are executed till now.

$$t_{n+1} = \frac{1}{n} \sum_{i=1}^{n} t_i$$

Exponential Averaging Technique

Estimate of the next CPU burst time by using an exponentially-weighted moving average over previous ones

$$\forall \tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$$

τ_{n+1} → *predicted value for the next CPU burst*

t_n → *actual length of n^{th} CPU burst*

α → *weight and is called smoothening factor* $(0 \le \alpha \le 1)$

Extreme cases

When $\alpha = 0 \Rightarrow \tau_{n+1} = \tau_n$; Only past history counts and recent history does not count

When $\alpha = 1 \Rightarrow \tau_{n+1} = t_n$; Only the Recent history (i.e, actual last CPU burst)counts.

Average case

When $\alpha = 0.5 \Rightarrow \tau_{n+1} = \frac{1}{2} t_n + \frac{1}{2} \tau_n$

Expanding the exponential averaging formula, we get

$$\tau_{n+1} = \alpha\ t_n + (1-\alpha)\ \alpha\ t_{n-1} + \ldots + (1-\alpha)^j \alpha\ t_{n-j} + \ldots + (1-\alpha)^{n-1} \alpha\ t_1 + (1-\alpha)^{n-1} \tau_1$$

- ***Each successive term has less weight than its predecessor.***

Problem on Exponential Averaging Formula

Calculate the predicted burst time using exponential averaging for the fifth CPU burst if the predicted burst time for the first is 10 units and actual burst time of the first four are 4, 8, 6 and 7 units respectively and the value for smoothening factor α is 0.5.

Solution-1

Given : $\tau_1=10$, $t_1=4$, $t_2=8$, $t_3=6$, $t_4=7$ $\alpha = 0.5$

Formula Used: $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$

$\tau_2 = \alpha\, t_1 + (1 - \alpha)\, \tau_1 = ½ \times 4 + (1-½) \times 10 = 7$

$\tau_3 = \alpha\, t_2 + (1 - \alpha)\, \tau_2 = ½ \times 8 + (1-½) \times 7 = 7.5$

$\tau_4 = \alpha\, t_3 + (1 - \alpha)\, \tau_3 = ½ \times 6 + (1-½) \times 7.5 = 6.75$

$\tau_5 = \alpha\, t_4 + (1 - \alpha)\, \tau_4 = ½ \times 7 + (1-½) \times 6.75 = \underline{6.875}$

Solution-2

Given $\tau_1=10,\quad t_1=4,\quad t_2=8,\quad t_3=6,\quad t_4=7\quad \alpha = 0.5$

$\forall\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$: Formula to be expanded

$\tau_5 = \alpha\, t_4 + (1 - \alpha)\, \tau_4$

$=\alpha t_4 +(1- \alpha)\, (\alpha\, t_3 + (1 - \alpha)\, \tau_3)$: substituting $\tau_4 = \alpha\, t_3 + (1 - \alpha)\, \tau_3$

$= \alpha t_4 +(1- \alpha)\, \alpha\, t_3 + (1 - \alpha)^2\, \tau_3$

$= \alpha t_4 +(1- \alpha)\, \alpha\, t_3 + (1 - \alpha)^2\, (\alpha\, t_2 + (1 - \alpha)\, \tau_2)$: substituting $\tau_3 = \alpha\, t_2 + (1 - \alpha)\, \tau_2$

$= \alpha t_4 +(1- \alpha)\, \alpha\, t_3 + (1 - \alpha)^2\, \alpha\, t_2 + (1 - \alpha)^3\, \tau_2$

$=\alpha t_4 +(1- \alpha)\, \alpha\, t_3 + (1 - \alpha)^2\, \alpha\, t_2 + (1 - \alpha)^3\, (\alpha\, t_1 + (1 - \alpha)\, \tau_1)$: substituting $\tau_2 = \alpha\, t_1 + (1 - \alpha)\, \tau_1$

Formula (expanded) used: $\tau_5 =\alpha t_4 +(1- \alpha)\, \alpha\, t_3 + (1 - \alpha)^2\, \alpha\, t_2 + (1 - \alpha)^3\, \alpha\, t_1 + (1 - \alpha)^3\, \tau_1$

$$=\frac{1}{2}\times 7+\left(1-\frac{1}{2}\right)\times\frac{1}{2}\times 6+\left(1-\frac{1}{2}\right)^2\times\frac{1}{2}\times 8+\left(1-\frac{1}{2}\right)^3\times\frac{1}{2}\times 4+\left(1-\frac{1}{2}\right)^3\times 10$$

$= 3.5+1.5+1+0.25+0.625 = \underline{6.875}$

Implementation of SJF Algorithm

- Practically, the algorithm can not be implemented but theoretically it can be implemented.
- Among all the available processes, the process with smallest burst time has to be selected. Min heap is a suitable data structure where root element contains the process with least burst time.
- In min heap, each process will be added and deleted exactly once.
- Adding an element takes log(n) time and deleting an element takes log(n) time.
- Thus, for n processes, time complexity = n x 2log(n) = nlog(n)

Priority Scheduling Algorithm

Policy: A priority value (integer) is associated with each process. When the CPU becomes available, it will be assigned with the process with highest priority. The equal priority processes are scheduled in FCFS order.

- Priority Scheduling can be either preemptive or non-preemptive
- Preemptive priority algorithm preempts the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.
- Priority can be defined either internally by the OS by using some measurable quantities such as *Aging, %CPU time used in last predefined time period, memory requirement* to compute the priority or externally set by some criteria that are external to OS.

Priority Scheduling (cont.)

SJF scheduling algorithm is the special case of a priority scheduling algorithm where the priority is the predicted value of next CPU burst time.

- General convention lower priority number represent higher priority. For example priority 1 is higher priority than priority 2
- But sometimes higher priority number represent higher priority only when it is explicitly stated in the problems. For example priority 2 is higher priority than priority 1
- The waiting time for the process having the highest priority will always be zero in preemptive mode.
- The waiting time for the process having the highest priority may not be zero in non-preemptive mode

Starvation Problem and Aging Technique

Starvation or indefinite blocking : A process that is ready to run but could not get CPU for long period of time can be considered as blocked. A priority scheduling algorithm can leave some low priority processes waiting indefinitely for the CPU and will result in one of the two possible ends:

- Eventually such processes will run after a long wait Or
- Left unfinished in case of system crash or shutdow

- Both SJF and Priority scheduling algorithms suffer from starvation problem

Aging Technique (A remedy/solution for starvation problem) : The problem of indefinite blockage of low-priority processes can be solved by using aging technique. The priority of the low priority processes will be incremented after specified time-interval (as time progresses the priority of the process will increase).

Round Robin (RR) Scheduling Algorithm

Policy: A small unit of time is called time-quantum or time-slice. The time quantum is usually 10-100 milliseconds . .In RR scheduling, the CPU scheduler goes around ready queue, allocating the CPU to each process for a time interval up to one time quantum.

Implementation: Here the ready queue is treated as a circular queue. New processes are added to the rear of the queue and the CPU scheduler picks up the process at the front of the queue and set a timer to interrupt after one time-quantum.

- The processes which may have the CPU-burst time less than one time-quantum will itself release the CPU.
- The processes which may have the CPU-burst time larger than one time-quantum will be interrupted, context switch is executed and sent to the rear of the ready queue

Round Robin (RR) Scheduling Algorithm (cont.)

Hence RR scheduling algorithm is similar to FCFS , but with pre-emption of running processes after one time quantum. (no process runs more than one time-quantum)

For *n* processes with Time Quantum = *q*

- Each process gets 1/n CPU time in chunks of at most *q* at a time.
- it will appear that user feel running the process with speed of 1/n of real processor
- No process waits more than (n-1)q time units.

Performance of RR scheduling

If Time slice *q* too large then results in FIFO (FCFS) behavior

If Time slice *q* too small - Overhead of context switch is too expensive.

- Suppose Process CPU-Burst time is 10 milliseconds
  - If q =12 then o additional context switch
  - If q = 6 then 1 additional context switch
  - If q =1 then 9 additional context switch
- So it is preferable to ser the time quantum such that the context switch time is approximately 10% of the time quantum.

Turnaround time also depends on the size of the time quantum.

- Heuristic - 70-80% of CPU-burst should be within the time slice

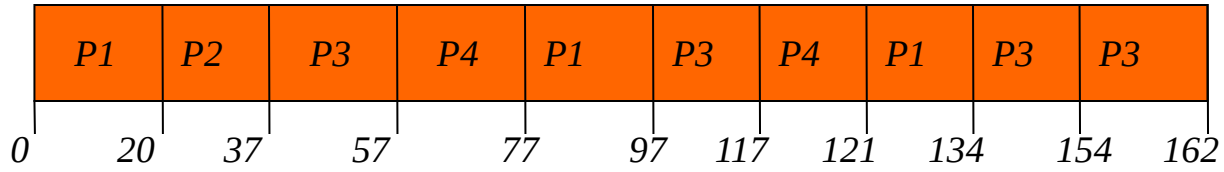
Problem on Round Robin scheduling Algorithm

Consider the table

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

Time Quantum = 20

Gantt Chart for RR Scheduling



Turnaround Time = Exit Time – Arrival Time
Waiting Time = Turn Around Time – Burst Time

Turnaround Time

P1=134-0=30

P2=37-0=37

P3=162-0=162

P4=121-0=121

Average: (30+37+162+121)/4= 50/4=87.5

Waiting time

P1 = 134-53=81;

P2 = 37-17=20;

P3 = 162-68=94;

P4=121-24=97

Average: (81+20+94+97)/4 = 292/4=73

- **Typically, higher average turnaround time than SRTF (0+17+41+94 = 38), but better response time**

Context switching

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch**.

- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is pure overhead, because the system does no useful work while switching.
- Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers).
- A typical speed is a few milliseconds.