**Process Scheduling**

Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process based on a particular strategy. Throughout its lifetime, a process moves between various [scheduling queues](https://www.geeksforgeeks.org/types-of-scheduling-queues/), such as the ready queue, waiting queue, or devices queue.

**CPU scheduling**:

CPU scheduling is a process used by the operating system to decide which task or process gets to use the CPU at a particular time. This is important because a CPU can only handle one task at a time, but there are usually many tasks that need to be processed. The following are different purposes of a CPU scheduling time.

* Maximize the CPU utilization
* Minimize the response and waiting time of the process.

**Terminologies Used in CPU Scheduling:**

* **Arrival Time:** The time at which the process arrives in the ready queue.
* **Completion Time:** The time at which the process completes its execution.
* **Burst Time:** Time required by a process for CPU execution.
* **Turn Around Time:** Time Difference between completion time and arrival time.
* Turn Around Time = Completion Time – Arrival Time
* **Waiting Time:** Time Difference between turn around time and burst time.
* Waiting Time = Turn Around Time – Burst Time

**A] First Come First Serve:**

First Come, First Serve (FCFS) is one of the simplest types of CPU scheduling algorithms. It is exactly what it sounds like: processes are attended to in the order in which they arrive in the ready queue, much like customers lining up at a grocery store. FCFS Scheduling is a non-preemptive algorithm, meaning once a process starts running, it cannot be stopped until it voluntarily relinquishes the CPU, typically when it terminates or performs I/O. This method schedules processes in the order they arrive, without considering priority or other factors.

**Code:**

def findWaitingTime(processes, n, bt, wt):

wt[0] = 0

for i in range(1, n ):

wt[i] = bt[i - 1] + wt[i - 1]

def findTurnAroundTime(processes, n, bt, wt, tat):

for i in range(n):

tat[i] = bt[i] + wt[i]

def findavgTime( processes, n, bt):

wt = [0] \* n

tat = [0] \* n

total\_wt = 0

total\_tat = 0

findWaitingTime(processes, n, bt, wt)

findTurnAroundTime(processes, n, bt, wt, tat)

print( "Processes Burst time " + " Waiting time " + " Turn around time")

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" " + str(i + 1) + "\t\t" + str(bt[i]) + "\t " + str(wt[i]) + "\t\t " + str(tat[i]))

print( "Average waiting time = "+str(total\_wt / n))

print("Average turn around time = "+str(total\_tat / n))

if \_\_name\_\_ =="\_\_main\_\_":

processes = [ 1, 2, 3]

n = len(processes)

burst\_time = [10, 5, 8]

findavgTime(processes, n, burst\_time)

**Output:**

Processes Burst time Waiting time Turn around time

1 10 0 10

2 5 10 15

3 8 15 23

Average waiting time = 8.333333333333334

Average turn around time = 16.0

**B] Shortest Job First:**

Shortest Job First (SJF) or Shortest Job Next (SJN) is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed.

**Code:**

def main():

# Taking the number of processes

n = int(input("Enter number of process: "))

# Matrix for storing Process Id, Burst Time, Average Waiting Time & Average Turn Around Time.

A = [[0 for j in range(4)] for i in range(100)]

total, avg\_wt, avg\_tat = 0, 0, 0

print("Enter Burst Time:")

for i in range(n): # User Input Burst Time and alloting Process Id.

A[i][1] = int(input(f"P{i+1}: "))

A[i][0] = i + 1

for i in range(n): # Sorting process according to their Burst Time.

index = i

for j in range(i + 1, n):

if A[j][1] < A[index][1]:

index = j

temp = A[i][1]

A[i][1] = A[index][1]

A[index][1] = temp

temp = A[i][0]

A[i][0] = A[index][0]

A[index][0] = temp

A[0][2] = 0 # Calculation of Waiting Times

for i in range(1, n):

A[i][2] = 0

for j in range(i):

A[i][2] += A[j][1]

total += A[i][2]

avg\_wt = total / n

total = 0

# Calculation of Turn Around Time and printing the data.

print("P BT WT TAT")

for i in range(n):

A[i][3] = A[i][1] + A[i][2]

total += A[i][3]

print(f"P{A[i][0]} {A[i][1]} {A[i][2]} {A[i][3]}")

avg\_tat = total / n

print(f"Average Waiting Time= {avg\_wt}")

print(f"Average Turnaround Time= {avg\_tat}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

Enter number of process: 5

Enter Burst Time:

P1: 5

P2: 8

P3: 6

P4: 9

P5: 0

P BT WT TAT

P5 0 0 0

P1 5 0 5

P3 6 5 11

P2 8 11 19

P4 9 19 28

Average Waiting Time= 7.0

Average Turnaround Time= 12.6

**C] Round Robin:**

Round Robin Scheduling is a method used by operating systems to manage the execution time of multiple processes that are competing for CPU attention. It is called "round robin" because the system rotates through all the processes, allocating each of them a fixed time slice or "quantum", regardless of their priority. The primary goal of this scheduling method is to ensure that all processes are given an equal opportunity to execute, promoting fairness among tasks.

**Code:**

def findWaitingTime(processes, n, bt, wt, quantum):

rem\_bt = [0] \* n

for i in range(n):

rem\_bt[i] = bt[i]

t = 0

while(1):

done = True

for i in range(n):

if (rem\_bt[i] > 0):

done = False

if (rem\_bt[i] > quantum):

t += quantum

rem\_bt[i] -= quantum

else:

t = t + rem\_bt[i]

wt[i] = t - bt[i]

rem\_bt[i] = 0

if (done == True):

break

def findTurnAroundTime(processes, n, bt, wt, tat):

for i in range(n):

tat[i] = bt[i] + wt[i]

def findavgTime(processes, n, bt, quantum):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, bt, wt, quantum)

findTurnAroundTime(processes, n, bt, wt, tat)

print("Processes Burst Time Waiting", "Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", i + 1, "\t\t", bt[i], "\t\t", wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f " % (total\_wt / n))

print("Average turn around time = %.5f " % (total\_tat / n))

if \_\_name\_\_ == "\_\_main\_\_":

proc = [1, 2, 3]

n = 3

burst\_time = [10, 5, 8]

quantum = 2

findavgTime(proc, n, burst\_time, quantum)

**Output:**

Processes Burst Time Waiting Time Turn-Around Time

1 10 13 23

2 5 10 15

3 8 13 21

Average waiting time = 12.00000

Average turn around time = 19.66667

**D] Priority Scheduling:**

Priority scheduling is one of the most common scheduling algorithms used by the operating system to schedule processes based on their priority. Each process is assigned a priority. The process with the highest priority is to be executed first and so on. Processes with the same priority are executed on a first-come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement. Also, priority can be decided on the ratio of average I/O to average CPU burst time.

**Code:**

def findWaitingTime(processes, n, wt):

wt[0] = 0

for i in range(1, n):

wt[i] = processes[i - 1][1] + wt[i - 1]

def findTurnAroundTime(processes, n, wt, tat):

for i in range(n):

tat[i] = processes[i][1] + wt[i]

def findavgTime(processes, n):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, wt)

findTurnAroundTime(processes, n, wt, tat)

print("\nProcesses Burst Time Waiting", "Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", processes[i][0], "\t\t", processes[i][1], "\t\t", wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f " % (total\_wt / n))

print("Average turn around time = ", total\_tat / n)

def priorityScheduling(proc, n):

proc = sorted(proc, key=lambda proc: proc[2], reverse=True)

print("Order in which processes gets executed")

for i in proc:

print(i[0], end=" ")

findavgTime(proc, n)

if \_\_name\_\_ == "\_\_main\_\_":

proc = [[1, 10, 1], [2, 5, 0], [3, 8, 1]]

n = 3

priorityScheduling(proc, n)

**Output:**

Order in which processes gets executed

1 3 2

Processes Burst Time Waiting Time Turn-Around Time

1 10 0 10

3 8 10 18

2 5 18 23

Average waiting time = 9.33333

Average turn around time = 17.0