**Priority preemptive**

n = int(input("Enter the number of processes: "))

proc = [] for i in range(n):

proc.append([0] \* 4)

for i in range(n):

arrival\_time = int(input(f"Enter arrival time for process {i+1}: ")) burst\_time = int(input(f"Enter burst time for process {i+1}: ")) priority = int(input(f"Enter priority for process {i+1}: "))

proc[i][0] = arrival\_time proc[i][1] = burst\_time proc[i][2] = priority proc[i][3] = i + 1

proc.sort()

wt = [0] \* n tat = [0] \* n

wavg = 0 tavg = 0

service = 0 # Cumulative burst time for i in range(n):

service += proc[i][1]

tat[i] = service - proc[i][0] wt[i] = tat[i] - proc[i][1] wavg += wt[i] tavg += tat[i]

# Display the process details, start time, complete time, turnaround time, and waiting time print("Process\_no\tStart\_time\tComplete\_time\tTurn\_Around\_Time\tWaiting\_Time")

for i in range(n):

print(proc[i][3], "\t\t", proc[i][0], "\t\t", proc[i][0] + tat[i], "\t\t", tat[i], "\t\t\t", wt[i])

# Display the average waiting time and average turnaround time print("Average waiting time: ", wavg / n) print("Average turnaround time: ", tavg / n)

**output:**

Enter the number of processes: 5

Enter arrival time for process 1: 1

Enter burst time for process 1: 3

Enter priority for process 1: 3

Enter arrival time for process 2: 2

Enter burst time for process 2: 5

Enter priority for process 2: 4

Enter arrival time for process 3: 3

Enter burst time for process 3: 1

Enter priority for process 3: 1

Enter arrival time for process 4: 4

Enter burst time for process 4: 7

Enter priority for process 4: 7

Enter arrival time for process 5: 5

Enter burst time for process 5: 4

Enter priority for process 5: 8

Process\_no Start\_time Complete\_time Turn\_Around\_Time Waiting\_Time

1. 1 3 2 -1
2. 2 8 6 1
3. 3 9 6 5
4. 4 16 12 5
5. 5 20 15 11

Average waiting time: 4.2

Average turnaround time: 8.2

**Priority non preemptive:**

def priority\_non\_preemptive(burst\_times, priorities):

n = len(burst\_times) processes = [] for i in range(n): process = {

'pid': i + 1,

'burst\_time': burst\_times[i],

'priority': priorities[i] # Priority

}

processes.append(process) processes.sort(key=lambda x: (x['priority'], x['pid']))

current\_time = 0 completed\_processes = []

for i in range(n):

process = processes[i]

process['start\_time'] = current\_time process['completion\_time'] = current\_time + process['burst\_time'] process['waiting\_time'] = process['start\_time'] process['turnaround\_time'] = process['completion\_time'] current\_time = process['completion\_time'] completed\_processes.append(process)

total\_waiting\_time = sum(process['waiting\_time'] for process in completed\_processes)

total\_turnaround\_time = sum(process['turnaround\_time'] for process in completed\_processes) avg\_waiting\_time = total\_waiting\_time / n avg\_turnaround\_time = total\_turnaround\_time / n

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}") print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

n = int(input("Enter the number of processes: "))

burst\_times = []

priorities = []

for i in range(n):

burst\_time = int(input(f"Enter the burst time for process {i + 1}: ")) burst\_times.append(burst\_time)

priority = int(input(f"Enter the priority for process {i + 1}: ")) priorities.append(priority) priority\_non\_preemptive(burst\_times, priorities) **output:**

Enter the number of processes: 5

Enter the burst time for process 1: 10

Enter the priority for process 1: 3

Enter the burst time for process 2: 1

Enter the priority for process 2: 1

Enter the burst time for process 3: 2

Enter the priority for process 3: 4

Enter the burst time for process 4: 1

Enter the priority for process 4: 5

Enter the burst time for process 5: 5

2Enter the priority for process 5: 2

Average Waiting Time: 8.20 Average Turnaround Time: 12.00

**SJF PREEPMTIVE :**

def findavgTime(processes, n):

wt = [0] \* n tat = [0] \* n rt = [p[1] for p in processes] complete = 0

t = 0

total\_wt = 0 total\_tat = 0

while complete != n:

short = min((rt[i], i) for i in range(n) if processes[i][2] <= t and rt[i] > 0)[1] rt[short] -= 1 t += 1

if rt[short] == 0: complete += 1

wt[short] = t - processes[short][1] - processes[short][2]

if wt[short] < 0: wt[short] = 0

tat[short] = processes[short][1] + wt[short] total\_wt += wt[short] total\_tat += tat[short]

print("Processes\tBurst Time\tWaiting Time\tTurnaround Time")

for i in range(n):

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

avg\_wt = total\_wt / n avg\_tat = total\_tat / n

print("\nAverage Waiting Time = %.5f" % avg\_wt) print("Average Turnaround Time = %.5f" % avg\_tat)

# Driver code if \_\_name\_\_ == "\_\_main\_\_":

n = int(input("Enter the number of processes: "))

proc = [] for i in range(n):

arrival\_time = int(input("Enter arrival time for process {}: ".format(i + 1))) burst\_time = int(input("Enter burst time for process {}: ".format(i + 1))) proc.append([i + 1, burst\_time, arrival\_time])

findavgTime(proc, n)

**output:**

Enter the number of processes: 4

Enter arrival time for process 1: 0

Enter burst time for process 1: 12

Enter arrival time for process 2: 2

Enter burst time for process 2: 4

Enter arrival time for process 3: 3

Enter burst time for process 3: 6

Enter arrival time for process 4: 8

Enter burst time for process 4: 5

Processes Burst Time Waiting Time Turnaround Time

1. 12 15 27
2. 4 0 4
3. 6 3 9
4. 5 4 9

Average Waiting Time = 5.50000

Average Turnaround Time = 12.25000

**Demonstrate Bankers Algorithm for Deadlock Avoidance.**

n = 5 # Number of processes

m = 3 # Number of resources

alloc = [[0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]]

max = [[7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]]

avail = [3, 3, 5]

f = [0] \* n

ans = []

need = [[max[i][j] - alloc[i][j] for j in range(m)] for i in range(n)]

for \_ in range(n):

for i in range(n):

if f[i] == 0 and all(need[i][j] <= avail[j] for j in range(m)):

ans.append(i)

for y in range(m):

avail[y] += alloc[i][y]

f[i] = 1

print("Following is the SAFE Sequence")

print("P", " -> P".join(map(str, ans)), sep="")

**output:**

P1 -> P3 -> P4 -> P0 -> P2

**Demonstrate Producer and Consumer problem using Semaphores.**

import threading

import time

mutex = threading.Lock()

full = 0

empty = 10

x = 0

def producer():

global mutex, full, empty, x

mutex.acquire()

if full < 10:

full += 1

empty -= 1

x += 1

print("Producer produces item", x)

else:

print("Buffer is full!")

mutex.release()

def consumer():

global mutex, full, empty, x

mutex.acquire()

if full > 0:

full -= 1

empty += 1

print("Consumer consumes item", x)

x -= 1

else:

print("Buffer is empty!")

mutex.release()

# Driver Code

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

while True:

print("\n1. Press 1 for Producer")

print("2. Press 2 for Consumer")

print("3. Press 3 for Exit")

n = int(input("Enter your choice: "))

if n == 1:

producer()

elif n == 2:

consumer()

elif n == 3:

break

else:

**print("Invalid choice!")**

**time.sleep(1)**

**output:**

1. Press 1 for Producer

2. Press 2 for Consumer

3. Press 3 for Exit

Enter your choice: 2

Buffer is empty!

1. Press 1 for Producer

2. Press 2 for Consumer

3. Press 3 for Exit

Enter your choice: 1

Producer produces item 1

1. Press 1 for Producer

2. Press 2 for Consumer

3. Press 3 for Exit

Enter your choice: 1

Producer produces item 2

1. Press 1 for Producer

2. Press 2 for Consumer

3. Press 3 for Exit

Enter your choice: 2

Consumer consumes item 2

1. Press 1 for Producer

2. Press 2 for Consumer

3. Press 3 for Exit

Enter your choice: 3

**Illustrate : fork (), wait (), exec() and exit () system calls**

import os

import multiprocessing

def child\_process():

print("Child Process")

print("Child PID:", os.getpid())

print("Parent PID:", os.getppid())

print("Executing ls command:")

os.execl("/bin/ls", "ls")

def parent\_process():

print("Parent Process")

print("Parent PID:", os.getpid())

print("Forking a child process...")

p = multiprocessing.Process(target=child\_process)

p.start()

p.join()

print("Child process has completed.")

print("Exiting parent process.")

parent\_process()

**output:**

Parent Process

Parent PID: 13764

Forking a child process...

Child process has completed.

Exiting parent process.