**SVKM’S NMIMS**

**MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT& ENGINEERING**

**(Campus Name)**

Academic Year: 2020-2021

# **Practical 4 – Program to Demonstrate the Round Robin Algorithm**

|  |  |  |
| --- | --- | --- |
| Name Arjun Mehta | Roll\_No K036 | SAP-ID 70102300018 |
|  |  |  |

Dear all,

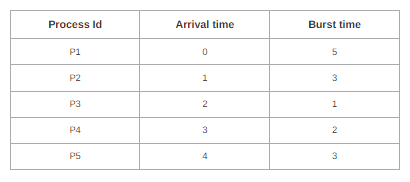
Kindly complete the following task with your name in output file also attach the C program with the file.

Find the Turnaround time and Average Turnaround time.

Find the Waiting time and Average Waiting time.

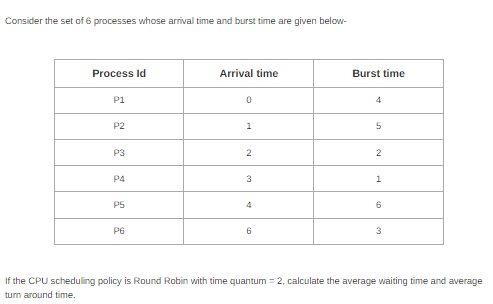
1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.   
   Waiting Time = Turn Around Time – Burst Time

Problem – 1: Consider the set of 5 processes whose arrival time and burst time are given below-



If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time

Problem 2:



Code:

from collections import deque

def round\_robin(processes, time\_quantum):

processes = sorted(processes, key=lambda x: x['arrival\_time'])

queue = deque()

current\_time = 0

next\_process = 0

total\_tat = 0

total\_wt = 0

# Initialize remaining\_time for all processes

for p in processes:

p['remaining\_time'] = p['burst\_time']

p['completion\_time'] = None

# Add initial processes to the queue

while next\_process < len(processes) and processes[next\_process]['arrival\_time'] <= current\_time:

queue.append(processes[next\_process])

next\_process += 1

while queue:

current\_process = queue.popleft()

execute\_time = min(time\_quantum, current\_process['remaining\_time'])

current\_time += execute\_time

current\_process['remaining\_time'] -= execute\_time

# Add newly arrived processes to the queue

while next\_process < len(processes) and processes[next\_process]['arrival\_time'] <= current\_time:

queue.append(processes[next\_process])

next\_process += 1

if current\_process['remaining\_time'] == 0:

current\_process['completion\_time'] = current\_time

tat = current\_process['completion\_time'] - current\_process['arrival\_time']

wt = tat - current\_process['burst\_time']

total\_tat += tat

total\_wt += wt

else:

queue.append(current\_process)

avg\_tat = total\_tat / len(processes)

avg\_wt = total\_wt / len(processes)

return avg\_tat, avg\_wt

# Problem 1 data

processes = [

{'pid': 'P1', 'arrival\_time': 0, 'burst\_time': 5},

{'pid': 'P2', 'arrival\_time': 1, 'burst\_time': 3},

{'pid': 'P3', 'arrival\_time': 2, 'burst\_time': 1},

{'pid': 'P4', 'arrival\_time': 3, 'burst\_time': 2},

{'pid': 'P5', 'arrival\_time': 4, 'burst\_time': 3},

]

time\_quantum = 2

avg\_tat, avg\_wt = round\_robin(processes, time\_quantum)

# Print results

print("Process | Completion Time | Turnaround Time | Waiting Time")

for p in processes:

tat = p['completion\_time'] - p['arrival\_time']

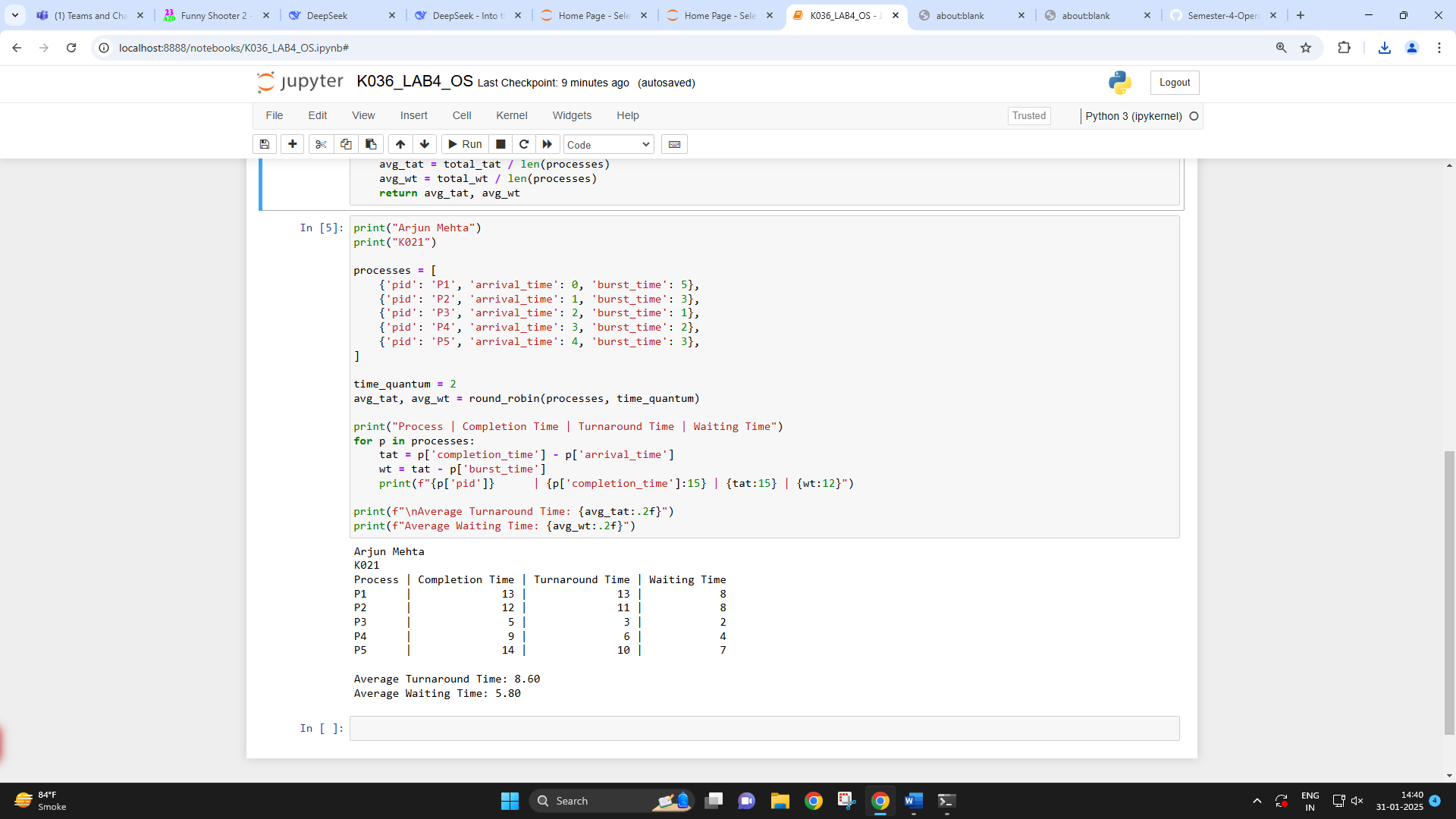
wt = tat - p['burst\_time']

print(f"{p['pid']} | {p['completion\_time']:15} | {tat:15} | {wt:12}")

print(f"\nAverage Turnaround Time: {avg\_tat:.2f}")

print(f"Average Waiting Time: {avg\_wt:.2f}")

OUTPUT:



**Conclusion: -**

Successfully learnt and implemented round robin.

The round-robin algorithm has several advantages over other scheduling algorithms. First, it is easy to implement and understand, as it only requires a simple queue and a timer. Second, it is fair and equitable, as it gives each task an equal share of the CPU time and prevents starvation.