Experiment – 5

Round Robin CPU Scheduling

Round Robin scheduling is simple, easy to implement an starvation free. In this algorithm equal portion of time (also known as time quanta) is assigned to each process and process order is handled in circular order without external priority. It is a preemptive process scheduling algorithm as the process are preempted after it spends a time equal to time quanta of system without completing it's execution. It's main focus is on multitasking. It is a real time algorithm and is so widely used across several fields. This algorithm gives the best performance in terms of average response time. First execution ties can easily tackled using arrival time or optional priorities.

ALGORITHM

1. Maintain a circular queue to store new and preempted process for future execution
2. Append new processes at the end of circular queue
3. If CPU is free, process the process at the front of queue of a constant time quanta
4. If the process completes execution befor time quanta ends, remove it from further execution.
5. Else add it to the end of circular queue for future execution of remaining work and load the process at the front of the queue into CPU for processing.

Numerical

Q. Concider the following process, whose arrival time and burst time is given, using Round Robin, with time quanta 3

Process	Arrival Time	Burst Time
P_1	0	4
P_2	1	2
P_3	3	5
P_4	7	8
P_5	12	3

Ans

Process	AT	BT	CT	TAT	WT	RT
P_1	0	4	9	9	5	0
P_2	1	2	5	4	2	2
P_3	3	5	14	11	6	2
P_4	7	8	22	15	7	2
P_5	12	3	17	5	2	2
			Average	8.8	4.4	1.6

Queue: 2 5 1 3 2 3 5

P_2	P_3	P_1	P_4	P_5	P_5	P_4

P_1	P_2	P_3	P_1	P_4	P_3	P_5	P_4

0 3 5 8 9 12 14 17 22

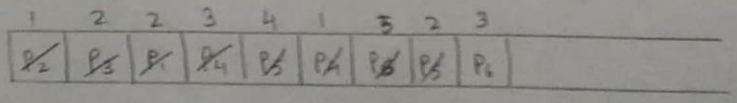
Q2. Concider the following process, whose arrival time & burst time is given, using Round Robin with time quanta 2

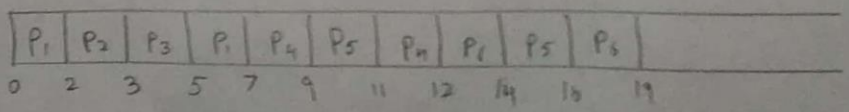
Process	Arrival Time	Burst Time
P_1	0	4
P_2	1	1
P_3	2	2
P_4	6	3
P_5	9	4
P_6	10	5

Ans

Process	AT	BT	CT	TAT	WT	RT
P_1	0	4	7	7	3	0
P_2	1	1	3	2	1	1
P_3	2	2	5	3	1	1
P_4	6	3	12	6	3	1
P_5	9	4	16	7	3	0
P_6	10	5	19	9	4	2
	Average		10.33	5.67	2.5	0.83

Queue





Q3. Concider the following processes, whose arrival time & burst time is given, using Round Robin, with time quanta 1

Process	Arrival Time	Burst time
P_1	0	3
P_2	2	5
P_3	3	2
P_4	5	2

Ans

Process	AT	BT	CT	TAT	WT	RT
P_1	0	3	4	4	1	0
P_2	2	5	12	10	5	0
P_3	3	2	8	5	3	1
P_4	5	2	10	5	3	1
		Average	8.5	6	3	0.5

Queue:

5	1	2	4	7	1	3	1	2
~~P_2~~	~~P_1~~	~~P_3~~	~~P_2~~	~~P_4~~	~~P_3~~	~~P_2~~	~~P_4~~	P_2

P_1	P_2	P_1	P_3	P_2	P_4	P_3	P_2	P_4	P_2
0	2	3	4	5	6	7	8	9	10	12

Code

```
from collections import deque

class Process:
    def __init__(self, idx, AT, BT):
        self.idx = idx
        self.AT = AT
        self.BT = BT
        self.CT = None
        self.firstExecution = None
        self.remaining = BT

    def calc(self):
        self.TAT = self.CT - self.AT
        self.WT = self.TAT - self.BT
        self.RT = self.firstExecution - self.AT

    def __repr__(self) -> str:
        return f"Process({self.idx}): {self.AT}, {self.BT}, {self.CT}, {self.TAT}
, {self.WT}, {self.RT}"
        # return f"Process({self.idx}): {self.AT}, {self.BT}, {self.firstExecutio
n}, {self.remaining}"


n = int(input("Number of processes: "))
r = int(input("Round Robin: "))
arrivalTime = list(map(int, input("Arrival Times: ").split()))
burstTime = list(map(int, input("Burst Times: ").split()))

processes = sorted([Process(x+
1,arrivalTime[x],burstTime[x]) for x in range(len(arrivalTime))],key=
lambda x:x.AT)
waitQueue = deque()
completed = []
arrivalIndex = 0
rotate = 0
cpuTime = processes[0].AT

while arrivalIndex<n or waitQueue:
    # CPU processing
    if waitQueue:
        # First Execution
        if waitQueue[0].firstExecution is None:
            waitQueue[0].firstExecution = cpuTime
```

```
        # preemption
        if waitQueue[0].remaining > r:
            waitQueue[0].remaining -= r
            cpuTime+=r
            rotate=1

        # processing complete
        else:
            cpuTime+=waitQueue[0].remaining
            waitQueue[0].CT = cpuTime
            waitQueue[0].calc()
            completed.append(waitQueue.popleft())

    # new arrivals
    for i in range(arrivalIndex,n):
        process = processes[i]
        if process.AT <= cpuTime:
            waitQueue.append(process)
            arrivalIndex+=1
        else:
            break

    # if processess preempted
    if rotate:
        waitQueue.rotate(-1)
        rotate = 0


print("Process, AT, BT, CT, TAT, WT, RT")
[print(x) for x in sorted(completed,key=lambda x: x.idx)]

print("\nDeekshant Wadhwa- 0129633118")
print("\nAverage:")
print(f"CT: {sum((x.CT for x in completed))/n}")
print(f"TAT: {sum((x.TAT for x in completed))/n}")
print(f"WT: {sum((x.WT for x in completed))/n}")
print(f"RT: {sum((x.RT for x in processes))/n}")
```

Output

```
PS D:\Drive\Sem 6\OS\lab> python -u "d:\Drive\Sem 6\OS\lab\roundRobin.py"
Number of processes: 5
Round Robin: 3
Arrival Times: 0 1 3 7 12
Burst Times: 4 2 5 8 3
Process, AT, BT, CT, TAT, WT, RT
Process(1): 0, 4, 9, 9, 5, 0
Process(2): 1, 2, 5, 4, 2, 2
Process(3): 3, 5, 14, 11, 6, 2
Process(4): 7, 8, 22, 15, 7, 2
Process(5): 12, 3, 17, 5, 2, 2

Deekshant Wadhwa- 0129633118

Average:
CT: 13.4
TAT: 8.8
WT: 4.4
RT: 1.6
PS D:\Drive\Sem 6\OS\lab>
```