Experiment no 7

Name: Shawn Louis SE COMPS Roll No: 31

Aim : Write a program to implement CPU scheduling algorithm - Round robin algorithm

Theory :
Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.
• It is simple, easy to implement, and starvation-free as all processes get fair share ofCPU.
• One of the most commonly used technique in CPU scheduling as a core.
• It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
• The disadvantage of it is more overhead of context switching.

Round Robin Example:

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2

0 **10**

P1 waiting time : 4

P2 waiting time: 6

P3 waiting time: 6

The average waiting time(AWT) : (4+6+6)/3=5.33

Algorithm :
Steps to find waiting times of all processes:
1- Create an array rem_bt[] to keep track of
remaining burst time of processes. This array is
initially a
copy of bt[] (burst times array)
2- Create another array wt[] to storewaiting
times of processes. Initialize this array as 0.
3- Initialize time : t = 0
4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is
not done yet.
a- If rem_bt[i] > quantum
(i) t = t + quantum
(ii) bt_rem[i] -= quantum;
c- Else // Last cycle for this process
(i) t = t + bt_rem[i];
(ii) wt[i] = t - bt[i]
(ii) bt_rem[i] = 0; // This process is over

Output :
Code :

```
import operator
class process:
	def init (self,no,bt,order,at,start=-1,end=-1,done=False,tat=-1,wt=-1):
		self.no=no
		self.bt=bt
		self.order=order
		self.at=at
		self.start=start
		self.end=end
		self.done=done
		self.tat=tat
		self.wt=wt
		self.tempbt=bt

l=[]
time=0
no=3
q=1
def cinput():
	global l,no,q;
	no=int(input('enter no of process'))
	q=int(input('time quantum'))
	for i in range(no):
		print('enter process no ,bt,order,at')
		tl=list(map(int,input().split()))
		l.append(process(tl[0],tl[1],tl[2],tl[3]))


def display():
	for i in range(no):
		print(l[i].no,l[i].bt,l[i].order,l[i].at)


cinput()

l.sort(key=operator.attrgetter('at'))
tk=[]
bursttime=[]
for i in range(no):
	tk.append(l[i].at)
```

```
        bursttime.append(l[i].bt)
    bts=sum(bursttime)
    time=min(tk)
    display()
    print()
    while (time<bts):
        for a in l:
            if(a.at<=time and a.done==False):
                temptime=time+q
                a.start=time

                while time<temptime and a.done==False:
                    time+=1
                    a.bt-=1
                    if(a.bt==0):
                        a.end=time
                        a.done=True



    avgt=0
    avgw=0
    for i in range(no):
        l[i].tat=l[i].end-l[i].at
        l[i].wt=l[i].tat-l[i].tempbt
        avgt+=l[i].tat
        avgw+=l[i].wt
        print('P{} CT={}s. TAT={}s. WT={}s.'.format(l[i].no,l[i].end,l[i].tat,l[i].wt))
print("Average TAT={}s. Average WaitTime={}s.".format(avgt/no,avgw/no))
```

Output:

```
enter no of process3
time quantum1
enter process no ,bt,order,at
1 3 1 0
enter process no ,bt,order,at
2 4 2 0
enter process no ,bt,order,at
3 3 3 0
1 3 1 0
2 4 2 0
3 3 3 0

P1 CT=7s. TAT=7s.  WT=4s.
P2 CT=10s. TAT=10s.  WT=6s.
P3 CT=9s. TAT=9s.  WT=6s.
Average TAT=8.666666666666666s. Average WaitTime=5.333333333333333s.
```

Conclusion:

In this experiment we have implemented the round robin method and output for the same has been recorded. The key factor about this algorithm is that is avoids starvation and this factor was studied and justified.