Preemptive scheduling assignment

60004200142
Abhishek Tiwari
B3 computer

Aim- CPU scheduling algorithms like Preemptive Priority, Round Robin etc.

Problem Statement-

- Perform comparative assessment of various Scheduling Policies like Priority preemptive and Round Robin.
- Take the input processes, their arrival time, burst time, priority, quantum from user.

Description:

Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first.

Terms used

1)Submit time :The process at which the process is given to CPU

2)Burst time : The amount of time each process takes for execution

3)Response time :The difference between the time when the process starts execution and the submit time.

4)Turnaround time :The difference between the time when the process completes execution and the submit time.

Priority Scheduling

Each process is assigned a priority and executable process with highest priority is allowed to run

CODE:

```
#include<bits/stdc++.h>

using namespace std;

#define ll long long int
```

```
void pro_sort(vector<pair<pair<ll,ll>,pair<ll,ll>>> &p){
	for(int i=0;i<p.size()-1;i++){
		for(int j=0;j<p.size()-i-1;j++){
			if(p[j].second.first>p[j+1].second.first){
				swap(p[j],p[j+1]);
			}
		}
	}
}
int main(){
	ll n,calc=0;
	cout<<"Number of processes: ";
	cin>>n;
	cout<<"Process"<<" "<<"Priority"<<" "<<"AT"<<" "<<"BT"<<endl;
	vector<pair<pair<ll,ll>,pair<ll,ll>>> process,copy;
	for(int i=0;i<n;i++){
		int pro,pri,at,bt;
		cin>>pro>>pri>>at>>bt;
		process.push_back({{pro,pri},{at,bt}});
		calc+=bt;
	}
	copy=process;
	pro_sort(copy);
	pro_sort(process);
	unordered_map<ll,ll> ma;
	int flag=1;
	int t=min(process[1].second.first,process[0].second.first+process[0].second.second);
	process[0].second.second-=(t-process[0].second.first);
	int start=process[0].second.first;
```

```
calc+=start;
vector<pair<ll,ll>> gc;
gc.push_back({process[0].first.first,t});
while(t<calc){
        int maxp=INT_MIN,idx;
        for(int i=0;i<n;i++){
                int pro=process[i].first.first;
                int pri=process[i].first.second;
                int at=process[i].second.first;
                if(ma[pro]!=1 && at<=t){
                        if(pri>maxp){
                                maxp=pri;
                                idx=i;
                        }
                }
        }
        int pro=process[idx].first.first;
        int pri=process[idx].first.second;
        int at=process[idx].second.first;
        int prev=gc[gc.size()-1].second;
        gc.push_back({process[idx].first.first,prev+1});
        process[idx].second.second-=1;
        t+=1;
        if(process[idx].second.second==0){
                ma[process[idx].first.first]=1;
        }
}
unordered_map<ll,ll> m;
vector<ll> ct(n+1),tat(n+1),wt(n+1);
```

```
	for(int i=gc.size()-1;i>=0;i--){
		if(m[gc[i].first]==0){
			ct[gc[i].first]=gc[i].second;
			m[gc[i].first]=1;
		}
	}
	for(int i=0;i<n;i++){
		int pro=process[i].first.first;
		tat[pro]=ct[pro]-process[i].second.first;
		wt[pro]=tat[pro]-copy[i].second.second;
	}
	float avgt=0,avgw=0;
	cout<<"Process\t\t"<<"Priority\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<endl;
	for(int i=0;i<n;i++){
		int pro=copy[i].first.first;

	cout<<copy[i].first.first<<"\t\t"<<copy[i].first.second<<"\t\t"<<copy[i].second.first<<"\t"<<copy[
i].second.second<<"\t"<<ct[pro]<<"\t"<<tat[pro]<<"\t"<<wt[pro]<<endl;
		avgt+=tat[pro];
		avgw+=wt[pro];
	}
	cout<<"Average tat: "<<avgt*1.0/n<<endl;
	cout<<"Average wt: "<<avgw*1.0/n<<endl;
}
```

Standard Input: ○ Interactive Console ◉ Text

```
7
1    2    0 4
2    4    1 2
3    6    2 3
4    10    3 5
5    8    4 1
6    12    5 4
7    9    6 6
```

Compiled Successfully. memory: 3692 time: 0.24 exit code: 0

```
Number of processes: Process Priority AT BT
Process         Priority        AT      BT      CT      TAT     WT
1               2               0       4       25      25      21
2               4               1       2       22      21      19
3               6               2       3       21      19      16
4               10              3       5       12      9       4
5               8               4       1       19      15      14
6               12              5       4       9       4       0
7               9               6       6       18      12      6
Average tat: 15
Average wt: 11.4286
```

Round Robin

- Each process is assigned a time interval called its quantum(time slice)
- If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed

```
int main()
{
	int i,j,pro,time,remain,flag=0,ts;
	struct times a[100];
	float avgwt=0,avgtt=0;
	printf("Round Robin Scheduling Algorithm\n");
	printf("Note -\n1. Arrival Time of at least on process should be 0\n2. CPU should never be idle\n");
	printf("Enter Number Of Processes : ");
	scanf("%d",&pro);
	remain=pro;
	for(i=0;i<pro;i++)
	{
		printf("Enter arrival time and Burst time for Process P%d : ",i);
		scanf("%d%d",&a[i].art,&a[i].but);
		a[i].p = i;
		a[i].rnt = a[i].but;
	}
	sortart(a,pro);
	printf("Enter Time Slice OR Quantum Number : ");
	scanf("%d",&ts);
	printf("\n****************************************\n");
	printf("Gantt Chart\n");
	printf("0");
	for(time=0,i=0;remain!=0;)
	{
		if(a[i].rnt<=ts && a[i].rnt>0)
		{
```

```
            time = time + a[i].rnt;
            printf(" -> [P%d] <- %d",a[i].p,time);
            a[i].rnt=0;
            flag=1;
        }
        else if(a[i].rnt > 0)
        {
            a[i].rnt = a[i].rnt - ts;
            time = time + ts;
            printf(" -> [P%d] <- %d",a[i].p,time);
        }
        if(a[i].rnt==0 && flag==1)
        {
            remain--;
            a[i].tat = time-a[i].art;
            a[i].wtt = time-a[i].art-a[i].but;
            avgwt = avgwt + time-a[i].art-a[i].but;
            avgtt = avgtt + time-a[i].art;
            flag=0;
        }
        if(i==pro-1)
            i=0;
        else if(a[i+1].art <= time)
            i++;
        else
            i=0;
    }
    printf("\n\n");
```

```
	printf("*************************************\n");
	printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n");
	printf("*************************************\n");
	for(i=0;i<pro;i++)
	{
		printf("P%d\t%d\t%d\t%d\t%d\n",a[i].p,a[i].art,a[i].but,a[i].tat,a[i].wtt);
	}
	printf("*************************************\n");
	avgwt = avgwt/pro;
	avgtt = avgtt/pro;
	printf("Average Waiting Time : %.2f\n",avgwt);
	printf("Average Turnaround Time : %.2f\n",avgtt);
	return 0;
}
```

OUTPUT:

```
Round Robin Scheduling Algorithm
Note -
1. Arrival Time of at least on process should be 0
2. CPU should never be idle
Enter Number Of Processes : 6
Enter arrival time and Burst time for Process P0 : 0
4
Enter arrival time and Burst time for Process P1 : 1
5
Enter arrival time and Burst time for Process P2 : 2
2
Enter arrival time and Burst time for Process P3 : 3
1
Enter arrival time and Burst time for Process P4 : 4
6
Enter arrival time and Burst time for Process P5 : 6
3
Enter Time Slice OR Quantum Number : 2
```

```
**************************************
Gantt Chart
0 -> [P0] <- 2 -> [P1] <- 4 -> [P2] <- 6 -> [P3] <- 7 -> [P4] <- 9 -> [P5] <- 11 -> [P0] <- 13 -> [P1] <- 15 -> [P4] <-
 17 -> [P5] <- 18 -> [P1] <- 19 -> [P4] <- 21

**************************************
Pro     ArTi    BuTi    TaTi    WtTi
**************************************
P0      0       4       13      9
P1      1       5       18      13
P2      2       2       4       2
P3      3       1       4       3
P4      4       6       17      11
P5      6       3       12      9
**************************************
Average Waiting Time : 7.83
Average Turnaround Time : 11.33
```