REPORT – 3

Dr. Manish Kumar Bajpai

PDPM IIITDM JABALPUR

Ankit Kesarwani (2015032)

Vivek Shukla (2015281)

MODIFICATION IN PROPOSED PRIORITY SCHEDULING ALGORITHM

Steps involve in Proposed Priority Scheduling Algorithm: -

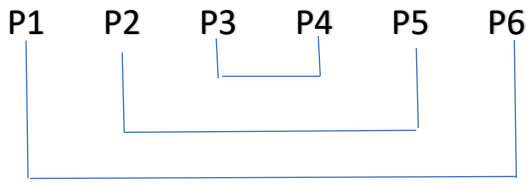
Step 1: - The task is sorted according to their priorities. Lower numbers are used to represent higher priorities. The process with the highest priority is allocated first and those with the same priorities are scheduled by FCFS policy.

Step 2: - Allocate all sorted process to CPU for a predetermined time slice.

Time Slice = (Burst time of shortest process + Burst time of largest process) / 2

Step 3: - After all the process have been executed once using the given time slice, swap the process priorities symmetrically with other process as shown below:

Suppose there are 6 processes say P1, P2, P3, P4, P5, P6 where priority of P1 is 1; P2 is 2; P3 is 3; P4 is 4; P5 is 5; P6 is 6, so swapping takes place symmetrically as follows:



SWAP

Additional Steps involve in Modification of Proposed Priority Scheduling Algorithm: -

Step 4: - After all the process have been executed once again calculate the new Time slice by using the same formula, and execute the process at maximum of new time slice and so on. i.e., No swapping of process takes place.

Suppose there are 6 processes say P1, P2, P3, P4, P5, P6 where priority of P1 is 1; P2 is 2; P3 is 3; P4 is 4; P5 is 5; P6 is 6, so once a cycle is completed again calculate the time slice of remaining processes and then execute them accordingly:

IMPLEMENTATION

```
#include<bits/stdc++.h>
using namespace std;
struct process{
	int pid;	// process id
	int priority;		// process priority
	double bt; // process burst time
	double wt;// process waiting time
	double rt;  // process remaining time
	double tat;// process turn around time
};
bool compare(struct process p1,struct process p2){
	return p1.priority>p2.priority?0:1;			// sorting according to priority
}
int main(){
	int n;
	cout<<"enter the number of processes"<<endl;
	cin>>n;
	struct process p[n];
```

```
	cout<<"enter burst time and priority of each process respectively"<<endl;
	for(int i=0;i<n;i++){
		p[i].pid=i+1;
		cin>>p[i].bt>>p[i].priority;		// taking input process burst time and priority
		p[i].rt=p[i].bt;
	}
	sort(p,p+n,compare);				// sorting
	bool flag=false;
	queue<int> q;
	double time=0;
	double bt_min=INT_MAX,bt_max=INT_MIN;
	double time_slice;
	while(flag==false){				// all process have not finished
		flag=true;
		bt_min=INT_MAX;
		bt_max=INT_MIN;
		for(int i=0;i<n;i++){
			if(p[i].rt>0){
				if(p[i].rt<bt_min){
					bt_min=p[i].rt;
```

```
                }                       // finding min and max
remaining time
                if(p[i].rt>bt_max){
                        bt_max=p[i].rt;
                }
            }
            else{
                continue;
            }

        }
        time_slice=(bt_max+bt_min)/2;          //  taking
time_slice as mean of max and min remaining time
        for(int i=0;i<n;i++){
            if(p[i].rt>0){
                if(time_slice>=p[i].rt){
                    time=time+p[i].rt;
                    p[i].rt=0;

                    p[i].tat=time;
                    p[i].wt=time-p[i].bt;
                    q.push(p[i].pid);
                }
```

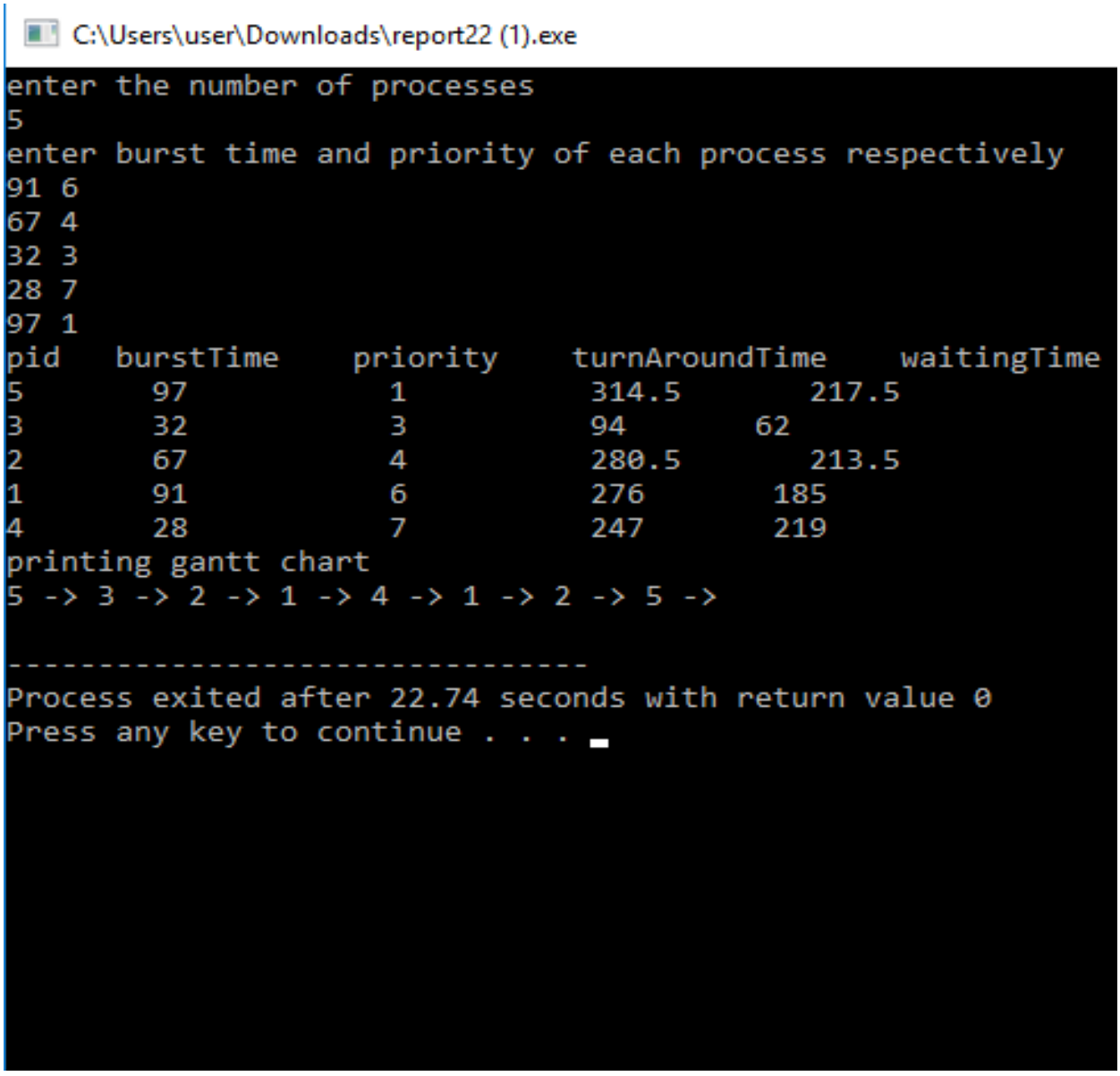
```
                else{
                        p[i].rt=p[i].rt-time_slice;
                        time=time+time_slice;
                        q.push(p[i].pid);
                }
        }
}
int pid,tat,bt,priority,wt,rt;

for(int i=0,j=n-1;i<n/2;i++,j--){
        pid=p[i].pid;
        tat=p[i].tat;
        bt=p[i].bt;
        priority=p[i].priority;
        wt=p[i].wt;
        rt=p[i].rt;
        p[i].pid=p[j].pid;              // swapping process
array
        p[i].tat=p[j].tat;
        p[i].bt=p[j].bt;
        p[i].priority=p[j].priority;
        p[i].wt=p[j].wt;
```

```
                p[i].rt=p[j].rt;
                p[j].pid=pid;
                p[j].tat=tat;
                p[j].bt=bt;
                p[j].priority=priority;
                p[j].wt=wt;
                p[j].rt=rt;

            }
            for(int i=0;i<n;i++){
                if(p[i].rt>0){                          // checking all process have been finished or not
                    flag=false;
                }
            }
        }

        // printing result

        cout<<"pid  burstTime  priority  turnAroundTime  waitingTime "<<endl;
        for(int i=0;i<n;i++){
```

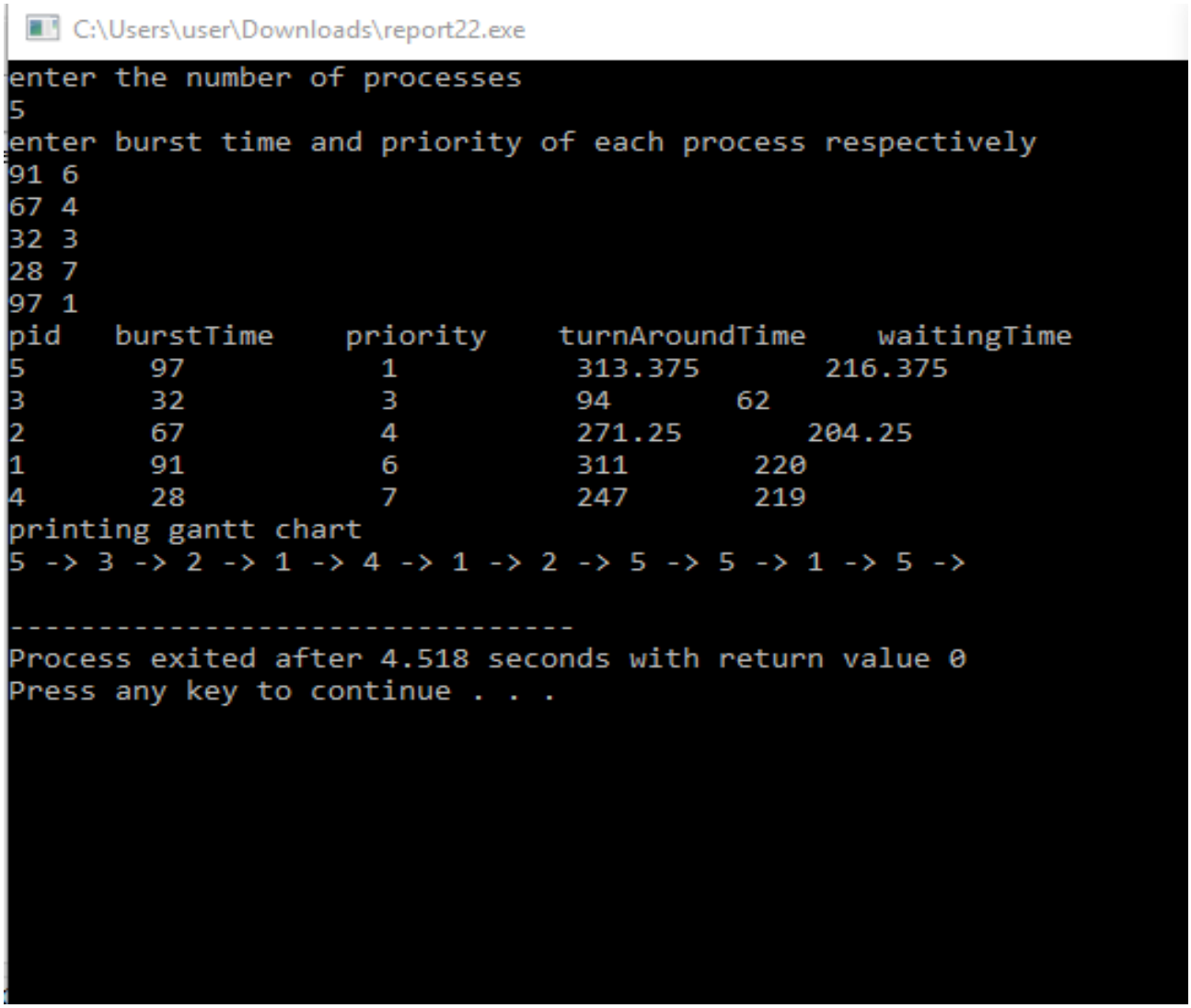
```
                cout<<p[i].pid<<"      "<<p[i].bt<<"
"<<p[i].priority<<"         "<<p[i].tat<<"      "<<p[i].wt<<endl;
        }
        int sz=q.size();
        cout<<"printing gantt chart"<<endl;
        for(int i=0;i<sz;i++){
                cout<<q.front()<<" -> ";
                q.pop();
        }
        cout<<endl;

        return 0;
}
```

OUTPUT SNAPSHOT FOR SAMPLE CASES

Snapshot for Proposed Priority Scheduling Algorithm: -



Snapshot for Proposed Priority Scheduling Algorithm after modification: -


C:\Users\user\Downloads\report22.exe
enter the number of processes
5
enter burst time and priority of each process respectively
91 6
67 4
32 3
28 7
97 1
pid burstTime priority turnAroundTime waitingTime
5 97 1 313.375 216.375
3 32 3 94 62
2 67 4 271.25 204.25
1 91 6 311 220
4 28 7 247 219
printing gantt chart
5 -> 3 -> 2 -> 1 -> 4 -> 1 -> 2 -> 5 -> 5 -> 1 -> 5 ->

Process exited after 4.518 seconds with return value 0
Press any key to continue . . .

CASE STUDY

Process No.	Priority	Arrival Time	Burst Time
P1	6	0	91
P2	4	0	67
P3	3	0	32
P4	7	0	28
P5	1	0	97

According to Proposed Priority Scheduling Algorithm:

Calculation of Time Slice:

Time Slice = (Burst time of shortest priority process + Burst time of largest priority process)/2

Time slice = (97 + 28) / 2 = 62.5

Gantt Chart: -

P5	P3	P2	P1	P4	P1	P2	P5

0 62.5 94.5 157 219.5 247.5 276 280.5 315

Process No.	Priority	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
P1	6	0	91	276	276	185	157
P2	4	0	67	280.5	280.5	213.5	94.5
P3	3	0	32	94.5	94.5	62.5	62.5
P4	7	0	28	247.5	247.5	219.5	219.5
P5	1	0	97	315	315	218	0

- Average Completion Time = 242.7
- Average Turn Around Time = 242.7
- Average Waiting Time = 179.7
- Average Response Time = 106.7

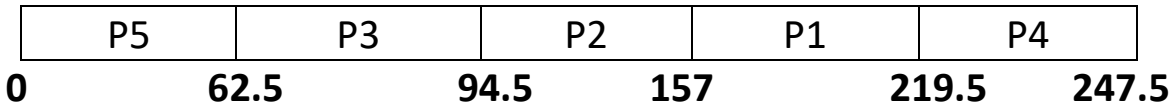
After Modification in Proposed Priority Scheduling Algorithm:

Calculation of Time Slice:

Time Slice = (Burst time of shortest priority process + Burst time of largest priority process)/2

Time slice = (97 + 28) / 2 = 62.5

Gantt Chart: -



Process Left for execution: -

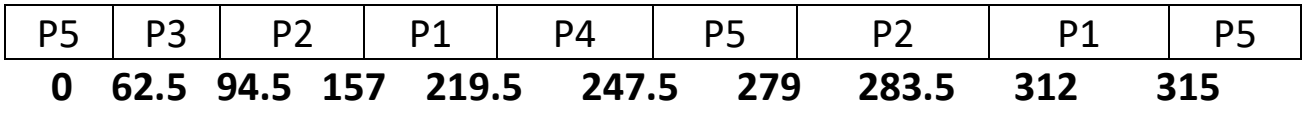
P1 for 28.5 with priority 6,

P2 for 4.5 with priority 4,

P5 for 34.5 with priority 1.

Now, Calculate new Time Slice: -

New Time Slice = (34.5 + 28.5) / 2 = 31.5



Process No.	Priority	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
P1	6	0	91	312	312	221	157
P2	4	0	67	283.5	283.5	216.5	94.5
P3	3	0	32	94.5	94.5	62.5	62.5
P4	7	0	28	247.5	247.5	219.5	219.5
P5	1	0	97	315	315	218	0

- Average Completion Time = 238.7
- Average Turn Around Time = 238.7
- Average Waiting Time = 187.5

- Average Response Time = 106.7

Analysis:

Algorithms	Average Turn Around Time	Average Waiting Time	Average Response Time	Context Switch
Proposed Priority Scheduling	242.7	179.7	106.7	7
After Modification in Proposed Priority Scheduling	238.7	187.5	106.7	8

Conclusion: -

- Response time for individual process is same.
- Lowest priority processes are prevented from starvation.
- Waiting time for Lowest priority process is minimized as compared to proposed priority scheduling algorithm.