Student Name: SATYA SAI RAM DASWANTH APPANA.

Student ID : 11802428

Roll no: 07

Email Address : daswanthappana@gmail.com.

Git Hub: https://github.com/Daswanth/Os-project

```
#include
<bits/stdc++.h>
                    using namespace std;

                    struct Process_Data
                    {
                            int Num;
                            int Pid;  //Process Id
                            int A_time; //Process Arrival Time
                            int B_time; //Process Bruest Time
                            int Priority; //Process Priority
                            int F_time; //Process Finish Time
                            int R_time; //Process Remaining  Time During Execution
                            int W_time; //Waiting Time
                            int S_time; //Process start Time
                            int Res_time;

                    };

                    struct Process_Data current;
                    typedef struct Process_Data P_d ;

                    bool idsort(const P_d& x , const P_d& y)
                    {
                            return x.Pid < y.Pid;
                    }
                    /** Sorting on the base of arrival time if that match then on Priority
                    of Priority also  match than on the base of Process Id**/
                    bool arrivalsort( const P_d& x ,const P_d& y)
                    {
                            if(x.A_time < y.A_time)
                                    return true;
                            else if(x.A_time > y.A_time)
                                    return false;
```

```
        if(x.Priority < y.Priority)
                return true;
        else if(x.Priority > y.Priority)
                return false;
        if(x.Pid < y.Pid)
                return true;

        return false;
}


bool Numsort( const P_d& x ,const P_d& y)
{
        return x.Num < y.Num;
}
/*Sorting on the base of Priority if that same then on the base of
PID*/
struct comPare
{
        bool operator()(const P_d& x ,const P_d& y)
        {
                if( x.Priority > y.Priority )
                        return true;
                else if( x.Priority < y.Priority )
                        return false;
                if( x.Pid > y.Pid )
                        return true;

                return false;

        }

};

/**To check the Input **/
void my_check(vector<P_d> mv)
{
        for(unsigned int i= 0; i < mv.size() ;i++)
        {
                cout<<" Pid :"<<mv[i].Pid<<" _time :
"<<mv[i].A_time<<" B_time : "<<mv[i].B_time<<" Priority :
"<<mv[i].Priority<<endl;
        }

}
```

```
int main()
{
	int i;
	vector< P_d > input;
	vector<P_d> input_copy;
	P_d temp;
	int pq_process = 0; // for PQ process
	int rq_process = 0; // for RQ process
	int A_time;
	int B_time;
	int Pid;
	int Priority;
	int n;
	int clock;
	int total_exection_time = 0;
	cin>>n;
	for( i= 0; i< n; i++ )
	{
		cin>>Pid>>A_time>>B_time>>Priority;
		temp.Num = i+1;
		temp.A_time = A_time;
		temp.B_time = B_time;
		temp.R_time = B_time;
		temp.Pid = Pid;
		temp.Priority = Priority;
		input.push_back(temp);
	}
	input_copy = input;
	sort( input.begin(), input.end(), arrivalsort );
  //cout<<"arrivalsort : "<<endl;
  //my_check( input ); // To check the sort unomment it
  total_exection_time = total_exection_time + input[0].A_time;
  for( i= 0 ;i< n; i++ )
  {
	if( total_exection_time >= input[i].A_time )
	{
		total_exection_time = total_exection_time
+input[i].B_time;
	}
	else
	{
		int diff = (input[i].A_time - total_exection_time);
		total_exection_time = total_exection_time + diff +
B_time;

	}
```

```
    }

        int Ghant[total_exection_time]={0}; //Ghant Chart
        for( i= 0; i< total_exection_time; i++ )
        {
                Ghant[i]=-1;
        }
        //cout<<"total_exection_time : "<<total_exection_time<<endl;

        priority_queue < P_d ,vector<Process_Data> ,comPare> pq;
//Priority Queue PQ

        queue< P_d > rq; //Round Robin Queue RQ
        int cpu_state = 0; //idle if 0 then Idle if 1 the Busy
        int quantum = 4 ; //Time Quantum
        current.Pid = -2;
        current.Priority = 999999;

        for ( clock = 0; clock< total_exection_time; clock++ )
        {
                /**Insert the process with same Arrival time in Priority
Queue**/
                for( int j = 0; j< n ; j++ )
                {
                        if(clock == input[j].A_time)
                        {
                                pq.push(input[j]);
                        }
                }


                if(cpu_state == 0) //If CPU idle
                {
                        if(!pq.empty())
                        {
                                current = pq.top();
                                cpu_state = 1;
                                pq_process = 1;
                                pq.pop();
                                quantum = 4;
                        }
                        else if(!rq.empty())
                        {
                                current = rq.front();
                                cpu_state = 1;
                                rq_process = 1;
```

```
                        rq.pop();
                        quantum = 4;
                }
        }
        else if(cpu_state == 1) //If cpu has any procss
        {
                if(pq_process == 1 && (!pq.empty()))
                {
                        if(pq.top().Priority < current.Priority )
//If new process has high priority
                        {
                                rq.push(current); //push current
in RQ
                                current = pq.top();
                                pq.pop();
                                quantum = 4;
                        }
                }
                else if(rq_process == 1 && (!pq.empty())) //If
process is from RQ and new process come  in PQ
                {
                        rq.push(current);
                        current = pq.top();
                        pq.pop();
                        rq_process = 0;
                        pq_process = 1;
                        quantum = 4 ;
                }


        }


        if(current.Pid != -2) // Process Execution
        {
                current.R_time--;
                quantum--;
                Ghant[clock] = current.Pid;
                if(current.R_time == 0) //If process Finish
                {
                        cpu_state = 0 ;
                        quantum = 4 ;
                        current.Pid = -2;
                        current.Priority = 999999;
                        rq_process = 0;
                        pq_process = 0;
```

```
                }
                else if(quantum == 0 ) //If time Qunatum of a current running process Finish
                {
                        rq.push(current);
                        current.Pid = -2;
                        current.Priority = 999999;
                        rq_process = 0;
                        pq_process = 0;
                        cpu_state=0;

                }

        }

}


sort( input.begin(), input.end(), idsort );

for(int i=0;i<n;i++)
{
        for(int k=total_exection_time;k>=0;k--)
        {
                if(Ghant[k]==i+1)
                {
                        input[i].F_time=k+1;
                        break;

                }
        }
}
for(int i=0;i<n;i++)
{
        for(int k=0;k<total_exection_time;k++)
        {

                if(Ghant[k]==i+1)
                {
                        input[i].S_time=k;
                        break;
                }
        }
}

sort( input.begin(), input.end(), Numsort );
```

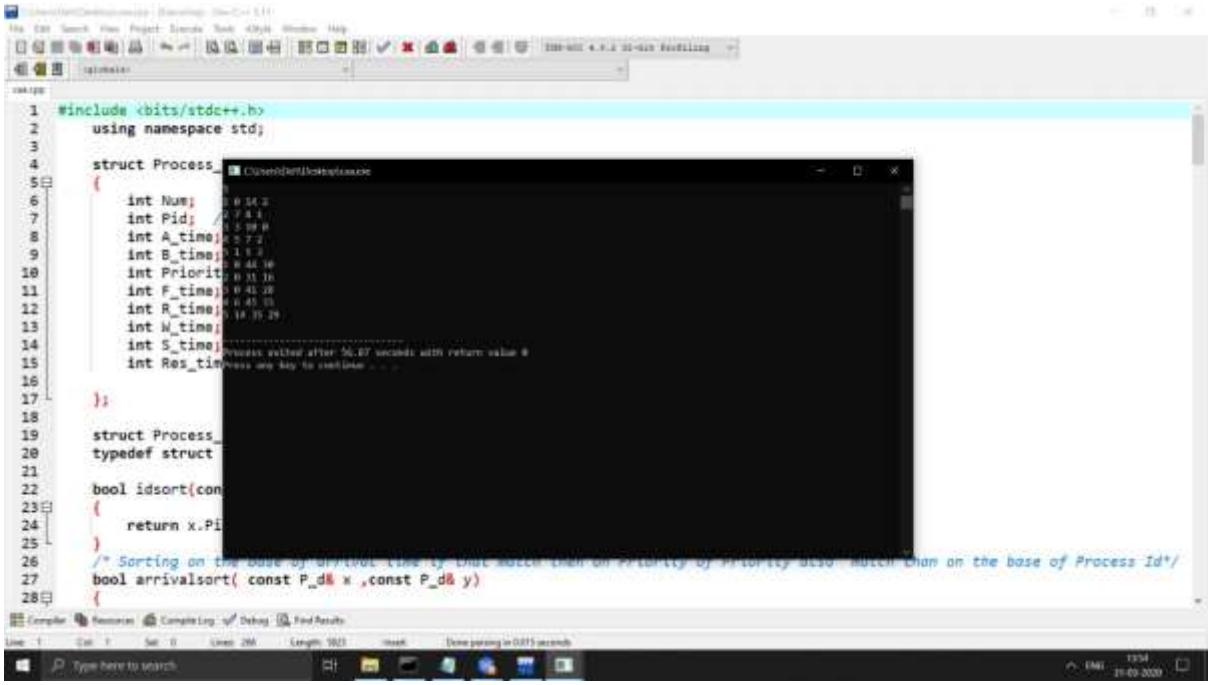
```
        for(int i=0;i<n;i++)
        {
                input[i].Res_time=input[i].S_time-input[i].A_time;
                input[i].W_time=(input[i].F_time-input[i].A_time)-
input[i].B_time;

        }

        for(int i=0;i<n;i++)
        {
                cout<<input[i].Pid<<" "<<input[i].Res_time<<"
"<<input[i].F_time<<" "<<input[i].W_time<<endl;

        }
        return 0;
}
```

OUTPUT:



ROUND ROBIN SCHEDULING :

A time quantum is associated to all processes. Time quantum is Maximum amount of time for which process can run once it is scheduled .And finally the Round Robin scheduling is always Preemptive.

Round Robin scheduling 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 of CPU.It is pre-emptive as processes are assigned CPU only for a fixed slice of time at most.

The main disadvantage of this Round Robin Scheduling of it is more overhead of context switching.

The main usage and nothing but the advantages of this RoundRobin scheduling Was there is a fairness since every process gets equal share of a CPU.The newly created process is added to end of ready queue.This process generally employs time sharing giving each job a time slot or quantum while performing a RoundRobin Scheduling a time quantum is allotted to different jobs.Each process get a chance to reschedule after a particular quantum time in this Scheduling.

In this RoundRobin scheduling the major disadvantages are the there is a larger waiting time and response time.there is low throughput. There is a Context Switches. Gantt chart seems to come too big. Time consuming scheduling for small quantum’s.

FIXED PRIORITY PREEMPTIVE SCHEDULING :

Fixed-priority preemptive scheduling is a scheduling system commonly used in real-time systems. With fixed priority preemptive scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to execute.

In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.
Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.