# Progress Report on

## SIMULATION BASED ASSIGNMENT

……..


School Of Computer Science And Engineering

**Subject: Operating System (CSE-316)**

**Submitted to: Dr. Baljit Singh Saini**

**Submitted by: M.Jai Surya Teja**
**B-60**
**K18KK**
**Reg. No: 11803022**

**Email address: suryateja2552@gmail.com**
**GitHub Link: https://github.com/suryateja2552/OS_CODE**

## Problem Statement:

Design a scheduling program to implements a Queue with two levels:
Level 1 : Fixed priority preemptive Scheduling

Level 2 : Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2.

All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.
2. Priority of Queue 2 has lower priority than in Queue 1.

## Code:

```
#include<iostream>
#include<queue>
#include<algorithm>
using namespace std;

struct Process_Data
{
	int Num;
	int Pid;			// denotes the Id of the process
	int arr_time;			// denotes the arrival time of the process
for execution
	int bur_time;			// denotes burst time of the process
	int Priority;			// denotes Priority of the process
```

```
	int end_time;				// time of complete execution of the process
Finish Time
	int rem_time;				// denotes remaining Time for complete
Execution of the process
	int idle_time;				// denotes for how much time time the
process is in waiting state
	int start_time;				// denotes the starting Time of the process
	int Res_time;				// denotes response time

};

struct Process_Data cur;
typedef struct Process_Data temp ;


//sorting in the ascending order of process ids
bool idsort(const temp& x , const temp& y)
{
	return x.Pid < y.Pid;
}


// Sorting on the basis of arrival time if that match then on Priority if their Priority
also  matche then on the basis of Process Id

bool arrivalsort( const temp& x ,const temp& y)
{
	if(x.arr_time < y.arr_time)
		return true;
	else if(x.arr_time > y.arr_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 temp& x ,const temp& y)
{
	return x.Num < y.Num;
}


//Sorting on the basis of Priority if they are same then on the basis of PID
struct comp
{
	bool operator()(const temp& x ,const temp& 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;

        }

};



int main()
{
        int i;
        vector< temp > input;
        vector<temp> input_copy;
        temp t1;
        int level1_q_process = 0;                       // for 1st level queue process
        int level2_q_process = 0;                       // for 2nd level queue process
        int arr_time;
        int bur_time;
        int Pid;
        int Priority;
        int n;
        int clock;
        int total_exection_time = 0;
        cout<<"enter the total number of processes : ";
        cin>>n;
        for( i= 0; i< n; i++ )
        {
                cout<<"\nprocess "<<(i+1)<<endl;
                cout<<"\nenter process id : ";
                cin>>Pid;
                cout<<"enter arrival time : ";
                cin>>arr_time;
                cout<<"enter burst time : ";
                cin>>bur_time;
                cout<<"enter priority of the process : ";
                cin>>Priority;
                t1.Num = i+1;
                t1.arr_time = arr_time;
                t1.bur_time = bur_time;
                t1.rem_time = bur_time;
                t1.Pid = Pid;
                t1.Priority = Priority;
                input.push_back(t1);
                cout<<"\n";
        }
        input_copy = input;
        sort( input.begin(), input.end(), arrivalsort );
    total_exection_time = total_exection_time + input[0].arr_time;
    for( i= 0 ;i< n; i++ )
    {
        if( total_exection_time >= input[i].arr_time )
        {
                total_exection_time = total_exection_time +input[i].bur_time;
        }
        else
        {
                int diff = (input[i].arr_time - total_exection_time);
```

```
            total_exection_time = total_exection_time + diff + bur_time;

        }
    }

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

        priority_queue < temp ,vector<Process_Data> ,comp> level1_q; //Priority Queue 1st
level queue

        queue< temp > level2_q;                                      //Round Robin Queue
2nd level queue
        int cpu_state = 0;                                           //idle if 0 then Idle
if 1 the Busy
        int quantum = 2 ;                                            //Time Quantum
        cur.Pid = -2;
        cur.Priority = 999999;

        for ( clock = 0; clock< total_exection_time; clock++ )
        {

                for( int j = 0; j< n ; j++ )
                {
                        if(clock == input[j].arr_time)
                        {
                                level1_q.push(input[j]);
                        }
                }


                if(cpu_state == 0)                      //If CPU is idle
                {
                        if(!level1_q.empty())
                        {
                                cur = level1_q.top();
                                cpu_state = 1;
                                level1_q_process = 1;
                                level1_q.pop();
                                quantum = 2;
                        }
                        else if(!level2_q.empty())
                        {
                                cur = level2_q.front();
                                cpu_state = 1;
                                level2_q_process = 1;
                                level2_q.pop();
                                quantum = 2;
                        }
                }
                else if(cpu_state == 1)                                 //If cpu has any
process i.e.,cpu not idle
                {
                        if(level1_q_process == 1 && (!level1_q.empty()))
                        {
```

```
                        if(level1_q.top().Priority < cur.Priority )     //If new
process has high priority
                        {
                                level2_q.push(cur);                          //push cur
in 2nd level queue
                                cur = level1_q.top();
                                level1_q.pop();
                                quantum = 2;
                        }
                }
                else if(level2_q_process == 1 && (!level1_q.empty()))    //If
process is from 2nd level queue and new process come  in 1st level queue
                {
                        level2_q.push(cur);
                        cur = level1_q.top();
                        level1_q.pop();
                        level2_q_process = 0;
                        level1_q_process = 1;
                        quantum = 2 ;
                }


        }


        if(cur.Pid != -2)                          // Process Execution
        {
                cur.rem_time--;
                quantum--;
                Ghant[clock] = cur.Pid;
                if(cur.rem_time == 0)                 //If process Finish
                {
                        cpu_state = 0 ;
                        quantum = 2 ;
                        cur.Pid = -2;
                        cur.Priority = 999999;
                        level2_q_process = 0;
                        level1_q_process = 0;
                }
                else if(quantum == 0 )                //If time Qunatum of a current
running process Finish
                {
                        level2_q.push(cur);
                        cur.Pid = -2;
                        cur.Priority = 999999;
                        level2_q_process = 0;
                        level1_q_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].end_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].start_time=k;
				break;
			}
		}
	}

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

	for(int i=0;i<n;i++)
	{
		input[i].Res_time=input[i].start_time-input[i].arr_time;
		input[i].idle_time=(input[i].end_time-input[i].arr_time)-input[i].bur_time;

	}

	for(int i=0;i<n;i++)
	{
		cout<<"\n\n\nprocess "<<(i+1);
		cout<<"\nprocess id is : "<<input[i].Pid<<endl;
		cout<<"response time of the process is : "<<input[i].Res_time<<endl;
		cout<<"finish time of the process is : "<<input[i].end_time<<endl;
		cout<<"idle time of the process : "<<input[i].idle_time<<endl;

	}
	return 0;
}
```

**Fixed-priority preemptive scheduling:** It 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.

**Round-robin:** It is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time slices (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free.

## Constraints:

1. Number of processes and their descriptions can't be altered during the execution of the code. They should be fixed initially.
2. Time should be entered as integer values of seconds.

Code Execution and Test Cases:

Sample test case input:

```
enter the total number of processes : 5

process 1

enter process id : 1
enter arrival time : 0
enter burst time : 14
enter priority of the process : 2


process 2

enter process id : 2
enter arrival time : 7
enter burst time : 8
enter priority of the process : 1


process 3

enter process id : 3
enter arrival time : 3
enter burst time : 10
enter priority of the process : 0


process 4

enter process id : 4
enter arrival time : 5
enter burst time : 7
enter priority of the process : 2


process 5

enter process id : 5
enter arrival time : 1
enter burst time : 5
enter priority of the process : 3
```

Sample test case output:

```
process 1
process id is : 1
response time of the process is : 0
finish time of the process is : 44
idle time of the process : 30



process 2
process id is : 2
response time of the process is : 0
finish time of the process is : 36
idle time of the process : 21



process 3
process id is : 3
response time of the process is : 0
finish time of the process is : 40
idle time of the process : 27



process 4
process id is : 4
response time of the process is : 0
finish time of the process is : 34
idle time of the process : 22



process 5
process id is : 5
response time of the process is : 1
finish time of the process is : 23
idle time of the process : 17
```