Report

(Operating System Assignment)

Course Code: CSE306

Under the guidance of

Vashudha Ma'am (Assistant Professor)

School of Computer Science and Engineering

Student Name: Sanjay Kumar

Student Registration: 11500754

Email Address: kumarsksingh010@gmail.com

GitHub Link:

Code:

```
#include<stdio.h>
#include <stdlib.h>
struct process{
        int priority;
        int burst_time;
        int pid;
        int waiting_time;
        int turnaround_time;
        int remaining_time;
        int arrival_time;
};
void getInput();
void calcWaitingTime(struct process *q,int);
void calcTurnAroundTime(struct process *q,int);
void printQueue(struct process *q,int size);
void RoundRobin();
void PrioSorting();
void FCFS();
void printQueueI(struct process);
void printQueue(struct process *,int);
```

```
int q1_n=0,q2_n=0,q3_n=0,n=0;
struct process *q1,*q2,*q3;
int time_quantum = 4;
void getInput(){
	printf("\n Total Number of Process:\t");
	scanf("%d",&n);
	q1 = (struct process *)malloc(n*sizeof(struct process));
	q2 = (struct process *)malloc(n*sizeof(struct process));
	q3 = (struct process *)malloc(n*sizeof(struct process));
	for(int i=0;i<n;i++){
		struct process p;
		printf("\n\t\tProcess
%d\n==========================================\n\n",i+1);
		p.arrival_time = (rand())%(n+1);
		printf("PId:\t");
		scanf("%d",&p.pid);
		printf("Priority (1-9):\t");
		scanf("%d",&p.priority);
		printf("\nBurst Time: %d\t",p.burst_time);
		scanf("%d",&p.burst_time);
		p.remaining_time = p.burst_time;
		if(p.priority>0 && p.priority<=3){
			q1[q1_n++]  = p;
		}else if(p.priority>3 && p.priority<=6){
			q2[q2_n++] = p;
```

```
                }else{
                        q3[q3_n++] = p;
                }
        }
}
void printQueue(struct process *q,int size){
        calcWaitingTime(q,size);
        calcTurnAroundTime(q,size);
        printf("\nPId\t\tPriority\t\tBurst Time\t\tWaiting Time\t\tTurnAround Time\t\tArrival");
        printf("\n=========================================================
======================\n");
        for(int i=0;i<size;i++){
                printQueueI(q[i]);
        }
        printf("\n\n");
}
void printQueueI(struct process p){
        printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t%d",p.pid,p.priority,p.burst_time,p.waiting_ti
me,p.turnaround_time,p.arrival_time);
}
void calcWaitingTime(struct process *q,int size){
        q[0].waiting_time = 0;
        for(int i=1;i<size;i++){
                q[i].waiting_time = q[i-1].waiting_time + q[i-1].burst_time;
        }
}
```

```
void calcTurnAroundTime(struct process *q,int size){
	q[0].waiting_time = 0;
	for(int i=0;i<size;i++){
		q[i].turnaround_time = q[i].waiting_time + q[i].burst_time;
	}
}
void RoundRobinAlgo(struct process *q,int size){
	int time=0,i=0,remain=size,flag=0,wait_time=0,tat_time=0,total_times=0;
	for(time=0,i=0;remain!=0;){
		struct process p = q[i];
		if(p.remaining_time<=time_quantum && p.remaining_time>0){
			time += p.remaining_time;
			p.remaining_time = 0;
			flag = 1;
		}else if(p.remaining_time>time_quantum){
			p.remaining_time -= time_quantum;
			time += time_quantum;
		}
		if(p.remaining_time==0 && flag==1){
			remain--;

	printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",p.pid,p.priority,p.burst_time,p.waiting_time,p.
turnaround_time);
			wait_time += time -p.arrival_time - p.burst_time;
			tat_time += time -p.arrival_time;
			flag = 0;
```

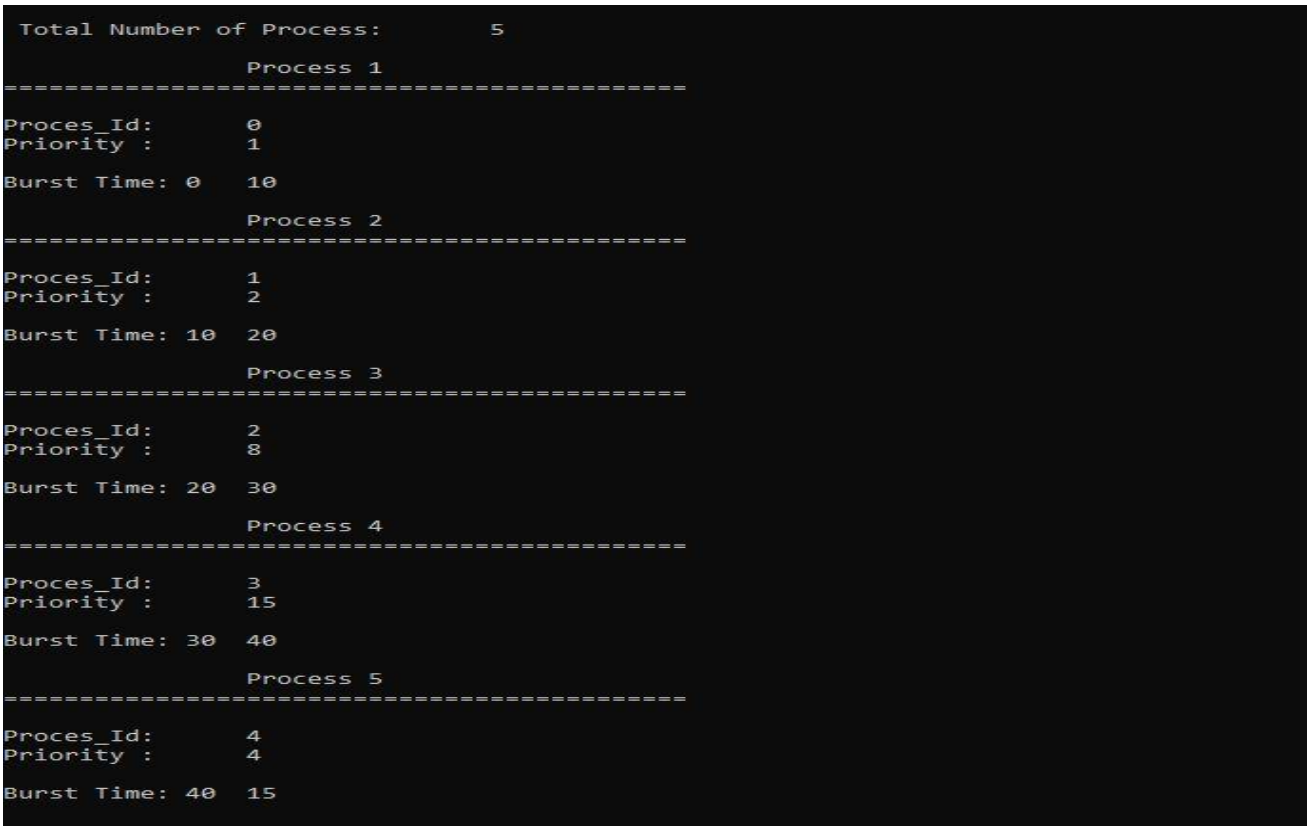
```
                }


                if(i==remain-1){
                        i=0;
                }else if(q[i+1].arrival_time<time){
                        i++;
                }else{
                        i=0;
                }


                q[i] = p;
        }
        printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
        printf("Avg Turnaround Time = %f\n",tat_time*1.0/n);
}
void RoundRobin(){
        printf("\n\n==========================================================
==================");
        printf("\n\t\tRound Robin\t");
        printf("\n==========================================================
==================\n\n");


        printf("\nPId\t\tPriority\t\tBurst Time\t\tWaiting Time\t\tTurnAround Time");
        printf("\n==========================================================
======================\n");
        calcWaitingTime(q3,q3_n);
```

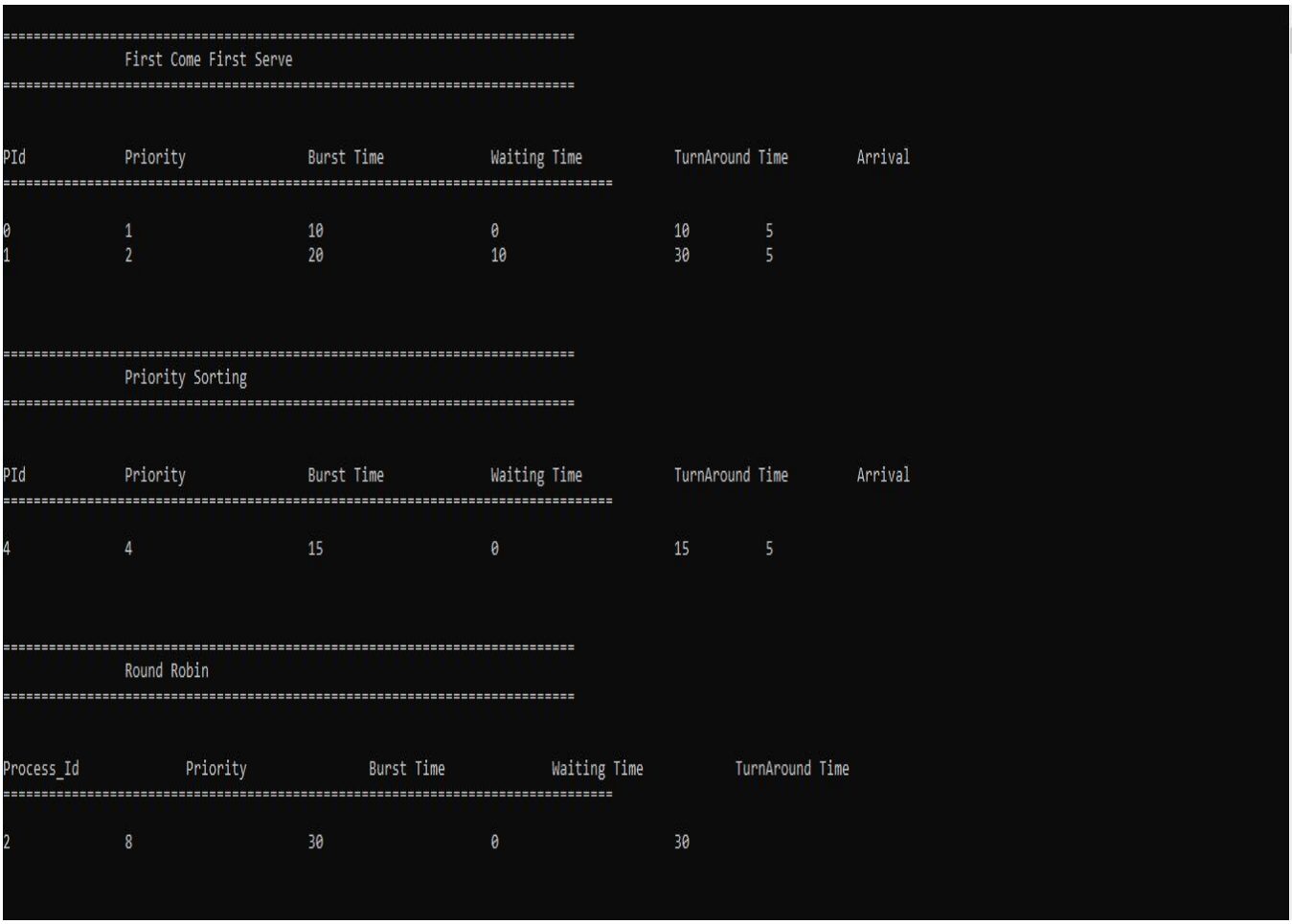
```
	calcTurnAroundTime(q3,q3_n);


	RoundRobinAlgo(q3,q3_n);
}
void PrioSortingAlgorithm(struct process *q,int size){
	for(int i=0;i<size;i++){
		for(int j=0;j<size;j++){
			if(q[j].priority>q[i].priority){
				struct process t = q[i];
				q[i] = q[j];
				q[j] = t;
			}
		}
	}
}
void PrioSorting(){
	printf("\n\n==========================================================
====================");
	printf("\n\t\tPriority Sorting\t");
	printf("\n==========================================================
==================\n\n");
	PrioSortingAlgorithm(q2,q2_n);
	printQueue(q2,q2_n);
}
void FCFSAlgorithm(struct process *q,int size){
	for(int i=0;i<size;i++){
```

```
                for(int j=0;j<size;j++){
                        if(q[j].arrival_time>q[i].arrival_time){
                                struct process t = q[i];
                                q[i] = q[j];
                                q[j] = t;
                        }
                }
        }
}
void FCFS(){
        printf("\n\n=========================================================
====================");
        printf("\n\t\tFirst Come First Serve\t");
        printf("\n=========================================================
==================\n\n");
        FCFSAlgorithm(q1,q1_n);
        printQueue(q1,q1_n);
}
int main(){
        getInput();
        int i=1;
        while(n>0){
                switch(i){
                        case 3:
                                RoundRobin();
                                break;
```

```
                case 2:
                        PrioSorting();
                        break;
                case 1:
                        FCFS();
                        break;
        }
        i++;
        sleep(10);
    }
    printf("\n\n");
}
```

Output:





Problem:

Multilevel queue scheduling: The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm. For example: separate ques might be used for foreground and background processes.

Description:

It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs.

A common division is a foreground (Interactive) process and background (batch) processes. These two classes have different scheduling needs. For this kind of situation Multilevel Queue scheduling is used. Now, let us see how it works. Ready queue is divided into separate queue for each class of processes, For example, let us take three different types of process system

processes, Interactive processes and Batch processes. All three process have their own queue. Now, look at the below figure. All three different type of processes have their own queue.

Each queue have its own Scheduling Algorithm. For example, queue 1 and queue 2 uses Round Robin while queue 3 can use FCFS to schedule there processes.

Scheduling among the queues: what will happen if all the queues have some processes? Which process should get the cpu. To determine this scheduling among the queue is necessary.

There are two ways to do so:-

Fixed priority pre-emptive scheduling method: - Each queue has absolute priority over queue 1> queue 2> queue 3. According to this algorithm no process in the batch queue (queue 3) can run unless queue 1 and 2 are empty. If any batch process (queue 3) is running and any system (queue 1) or interactive process is pre-empted.

Time Slicing – In this method each queue gets certain portion of CPU time and can use it to schedule its own processes. For instance, queue 1 takes 50 percent of CPU time.

Algorithm:

Queue	Priority Range	Algorithm
1	1-3	First Come First Serve
2	4-6	Priority Scheduling
3	>6	Round Robin

Code Snippet:

```
q1 = (struct process *)malloc(n*sizeof(struct process));

q2 = (struct process *)malloc(n*sizeof(struct process));

q3 = (struct process *)malloc(n*sizeof(struct process));
```

Here, we will learn the name malloc are library functions that allocate memory dynamically. It means that memory is allocated during runtime(execution of the program) from heap segment.

GitHub Revision:

I have made 13 revision of code on GitHub.

GitHub Link: