Name :- Amit.Y.Zope
Roll No. :- 223067
Gr No. :- 21810714
Division :- C
Batch :- C3
Subject :- Operating System

Assignment No 4

Aim: To show the demonstration of Scheduling Algorithms: 1) FCFS 2) SJF 3) Round Robin.

Objective: To study the Scheduling Algorithms: 1) FCFS 2) SJF 3) Round Robin.

Theory:

What is Process Scheduling ?

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

The Operating System maintains the following important process scheduling queues −

1. **Job queue** − This queue keeps all the processes in the system.

2. **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

3. **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system.

What is Preemptive Scheduling ?

Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining.

Algorithms based on preemptive scheduling are: **Round Robin (RR),Shortest Remaining Time First (SRTF), Priority (preemptive).**

What is Non-Preemptive Scheduling ?

Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.
Algorithms based on non-preemptive scheduling are: **Shortest Job First (SJF basically non preemptive) and Priority (non preemptive).**

Scheduling Algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithm −

1. **First-Come, First-Served (FCFS) Scheduling* .**
2. **Shortest-Job-First (SJF) Scheduling* .**
3. **Priority Scheduling.**
4. **Shortest Remaining Time.**

5. **Round Robin(RR) Scheduling* .**
6. **Multiple-Level Queues Scheduling.**

These algorithms are either non-preemptive or preemptive.

1. First Come First Serve (FCFS).

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

2. Shortest –Job-First (SJF).

- This is also known as shortest job first, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

3. Round Robin Scheduling.

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

NOTE : The above given example is executed with following code.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
int np;

struct node{
   int process_no,arrival_time,burst_time,burst_time2,CP_time,WT_time,TA_time;
   int status,status2,checkar;
   struct node* next;
}s;

void sort_common(struct node* temp,struct node* temp1){
                        int temp2;
                     temp2 = temp->arrival_time;
                              temp->arrival_time = temp1->arrival_time;
                              temp1->arrival_time = temp2;
                              temp2 = temp->burst_time;
                              temp->burst_time = temp1->burst_time;
                              temp1->burst_time = temp2;
                              temp2 = temp->process_no;

                              temp->process_no = temp1->process_no;

                              temp1->process_no = temp2;

}
void display(struct node* first){
     struct node* temp = first;
     printf("\n\n");
```

```
	printf("\t\tProcessNumber  \tArrivalTime \tBurstTime\n");
	while(temp !=NULL){
	 printf("\t\t%-16d %-15d %-17d\n",temp->process_no,temp->arrival_time,temp-
>burst_time);
	 temp = temp->next;
	}
}
struct node* create(int pr,int at,int bt,struct node* first){

		struct node* newnode =(struct node*)malloc(sizeof(struct node));
		newnode->process_no=pr;
		newnode->arrival_time=at;
		newnode->burst_time=bt;
		newnode->burst_time2=bt;
		newnode->checkar = 0;
		newnode->status = 0;
		newnode->status2 = 0;
		newnode->next=NULL;

		 if(first == NULL){
			first =newnode;
		 }
			else{
			struct node* temp = first;
			while(temp->next!=NULL){
				temp = temp->next;
			}
			temp->next=newnode;
		}

	return first;
}
void SA_FCFS(struct node* first){
	struct node *temp = first,*temp1,*temp3;
	int temp2,c_time;
	float avgwt=0.0,tat=0.0;
	while(temp->next!=NULL){
		temp1=temp->next;
		while(temp1!=NULL){
			if(temp->arrival_time>temp1->arrival_time){
				sort_common(temp,temp1);
			}
```

```
                temp1=temp1->next;
            }
            temp=temp->next;

        }
        temp3= first;
        c_time=temp3->arrival_time;
        printf("\n\nProcessNumber \tArrivalTime\tBurstTime\tCompletionTime
\tWaitingTime  \tTurnaroundTime \n");
        while(temp3!=NULL){
            temp3->CP_time = c_time + temp3->burst_time;
            c_time = temp3->CP_time;
            temp3->TA_time = temp3->CP_time - temp3->arrival_time;
            temp3->WT_time = temp3->TA_time - temp3->burst_time;
            printf("%-16d %-15d %-15d %-15d %-17d %-14d\n",temp3-
>process_no,temp3->arrival_time,temp3->burst_time,c_time,temp3-
>WT_time,temp3->TA_time);
            temp3 = temp3->next;
        }
        printf("\n");
        printf("\n");

        temp = first;



        printf("\n|-------------------------------------------------------------------------------|\n\n");


                printf("\t   SCHEDULING ORDER OF PROCESSES       : ");

                while(temp!=NULL){
                    printf("P(%d)->  ",temp->process_no);
                    temp = temp->next;
                }
                printf("\n")
                temp3 = first;
                while(temp3!=NULL){
                    avgwt = avgwt + temp3->WT_time;
                    tat = tat + temp3->TA_time;
                    temp3 = temp3->next;
            }
```

```
                printf("\n\t   AVERAGE WAITING TIME                    :
        %f\n\n",avgwt/np);
                printf("\t   TURNAROUND TIME IS                      : %f\n",tat/np);
        printf("\n|-------------------------------------------------------------------------------|\n");

}
        void SA_SJF(struct node* first){
        struct node *temp = first,*temp1;
        int temp2,c_time;
        float avgwt=0.0,tat=0.0;
        while(temp->next!=NULL){
                temp1=temp->next;
                while(temp1!=NULL){
                        if(temp->arrival_time>temp1->arrival_time){
                                sort_common(temp,temp1);
                        }
                        temp1=temp1->next;
                }
                temp=temp->next;
        }
        temp=first->next;
        struct node *temp3,*temp7=first->next,*traverse=first;
        int time = first->burst_time,min,min2,min_pos;
         int i =0;
        while(temp->next!=NULL){
                temp3=temp;
                while(temp3->next!=NULL){
                temp7 = temp3;
                        while(temp7){
                        if(temp7->arrival_time <= time){

                                temp7->status = 1;

                        }else{
                                temp7->status =0;
                        }
                        temp7 = temp7->next;
                  }
                        struct node *temp5 = temp3->next;
                        min = temp3->burst_time;
                        int flag=0,flag1=0,flag2=0;
                        while(temp5){
```

```
                        if(temp3->status == 1 && temp5->status == 1){
                                if(min > temp5->burst_time  ){
                                        min = temp5->burst_time;
                                        min_pos = temp5->process_no;
                                        flag = 1;
                                }
                                if(min == temp5->burst_time){
                                     if(temp5->arrival_time < temp3->arrival_time){
                                                flag = 2;
                                                min_pos = temp5->process_no;
                                     }
                                }
                                if(temp5->next == NULL && temp3->next->next == NULL
&& temp5->burst_time > temp3->burst_time){
                                        min = temp3->burst_time;
                                        min_pos = temp3->process_no;
                                        flag1 = 1;
                          }
                        }
                        temp5 = temp5->next;
                }
                  if(flag1==0 &&flag==0 && min == temp3->burst_time && temp3-
>status2 ==0 ){
                                        time = time +min;
                                        temp3->status2 = 1;
                                        if( i == np-1){
                                          break;
                                        }
                                }

                    if(flag == 2 && min == temp3->burst_time  && temp3->status2 ==0){
                            struct node* temp9=first;
                    while(temp9){
                            if(temp9->process_no == min_pos){
                                    break;
                            }
                            temp9 = temp9->next;
                    }
                    sort_common(temp3,temp9);
                            temp3->status2 = 1;
                            time = time+min;
                    }
```

```
                    if(flag == 1 && temp3->status2 == 0){
                     struct node* temp6=first;
                     while(temp6){
                            if(temp6->process_no == min_pos){
                                    break;
                            }
                            temp6 = temp6->next;
                     }
                     sort_common(temp3,temp6);

                               temp3->status2 = 1;
                               time = time+min
                    }
                    if(flag1 == 1 && temp3->status2 == 0){
                     struct node* iter=first;
                     while(iter){
                            if(iter->process_no == min_pos){
                                    break;
                            }
                            iter = iter->next;
                     }
                     sort_common(temp3,iter);
                            temp3->status2 =1;
                            time = time+min;
                    }
                    temp3 = temp3->next;

            }
                     display(first);

            i++;
            if(i==np-2
                     break;
            }else{
                     temp = temp-> next;
            }
     }
     temp3 = first;
      c_time=temp3->arrival_time;
      printf("\n\nProcessNumber \tArrivalTime\tBurstTime\tCompletionTime
\tWaitingTime  \tTurnaroundTime \n");
```

```
        while(temp3!=NULL){
                temp3->CP_time = c_time + temp3->burst_time;
                c_time = temp3->CP_time;
                temp3->TA_time = temp3->CP_time - temp3->arrival_time;
                temp3->WT_time = temp3->TA_time - temp3->burst_time;
                printf("%-16d %-15d %-15d %-15d %-17d %-14d\n",temp3-
>process_no,temp3->arrival->time,temp3->burst_time,c_time,temp3-
>WT_time,temp3->TA_time);
                temp3 = temp3->next;
        }
        printf("\n");
        printf("\n");
        temp = first;
        printf("\n|-------------------------------------------------------------------------------|\n\n")
                        printf("\t   SCHEDULING ORDER OF PROCESSES      : ");
                        while(temp!=NULL){
                                printf("P(%d)->  ",temp->process_no);
                                temp = temp->next;
                        }
                        printf("\n");
                        temp3 = first;
                        while(temp3!=NULL){
                                avgwt = avgwt + temp3->WT_time;
                                tat = tat + temp3->TA_time;
                                temp3 = temp3->next;
                        }

                        printf("\n\t   AVERAGE WAITING TIME                          :
%f\n\n",avgwt/np);
                        printf("\t   TURNAROUND TIME IS                        : %f\n",tat/np);
        printf("\n|-------------------------------------------------------------------------------|\n");
}
        void SA_RRobin(struct node* first){

                        struct node *temp = first,*temp1;
          int temp2,time_quantum;
          float avgwt=0.0,tat=0.0;
                 while(temp->next!=NULL){
                        temp1=temp->next;
                        while(temp1!=NULL){
                                if(temp->arrival_time>temp1->arrival_time){
                                        temp2 = temp->arrival_time;
```

```
                        temp->arrival_time = temp1->arrival_time;
                        temp1->arrival_time = temp2;

                        temp2 = temp->burst_time;
                        temp->burst_time = temp1->burst_time;
                        temp1->burst_time = temp2;

                        temp2 = temp->burst_time2;
                        temp->burst_time2 = temp1->burst_time2;
                        temp1->burst_time2 = temp2;


                        temp2 = temp->process_no;
                        temp->process_no = temp1->process_no;
                        temp1->process_no = temp2;

                }
                temp1=temp1->next;
        }
        temp=temp->next;
  }
display(first);

  printf("\n\n\tENTER THE TIME QUANTUM : ");
  scanf("%d",&time_quantum);
  printf("\n");
  printf("\n\tProcess Occurrrences :   ");
  int remin_pro = np,i,check = 0,total_time,k=0;
  int wt[np],j=0,ta[np],c_time[np];
  temp = first;

  for(i=0,total_time=0;temp!=NULL && remin_pro!=0;){

        if(temp->burst_time2<=time_quantum && temp->burst_time2>0){
                total_time += temp->burst_time2;
                printf("P(%d)->",temp->process_no);
                temp->burst_time2=0;
                check = 1;
        }
        else if(temp->burst_time2>0){

                temp->burst_time2 -= time_quantum;
```

```
                printf("->P(%d)->",temp->process_no);
                total_time += time_quantum;
            }
            if(temp->burst_time2 == 0 && check == 1){
                remin_pro--;

                temp->TA_time = total_time - temp->arrival_time;
                temp->WT_time = total_time - temp->arrival_time - temp-
>burst_time;

                k = temp->process_no-1;
                wt[k] = total_time - temp->arrival_time - temp->burst_time;
                ta[k] = total_time - temp->arrival_time;
                c_time[k] = total_time;


                avgwt += temp->WT_time;
                tat += temp->TA_time;
                j++;
                check = 0;

            }

            if(i==np-1){
                i=0;
                temp = first;
            }
            else if((temp->next)->arrival_time <= total_time){
                i++;
                temp = temp->next;
            }else
            {
                i=0;
                temp= first;
            }


        }


    temp = first;
    k = 0;
```

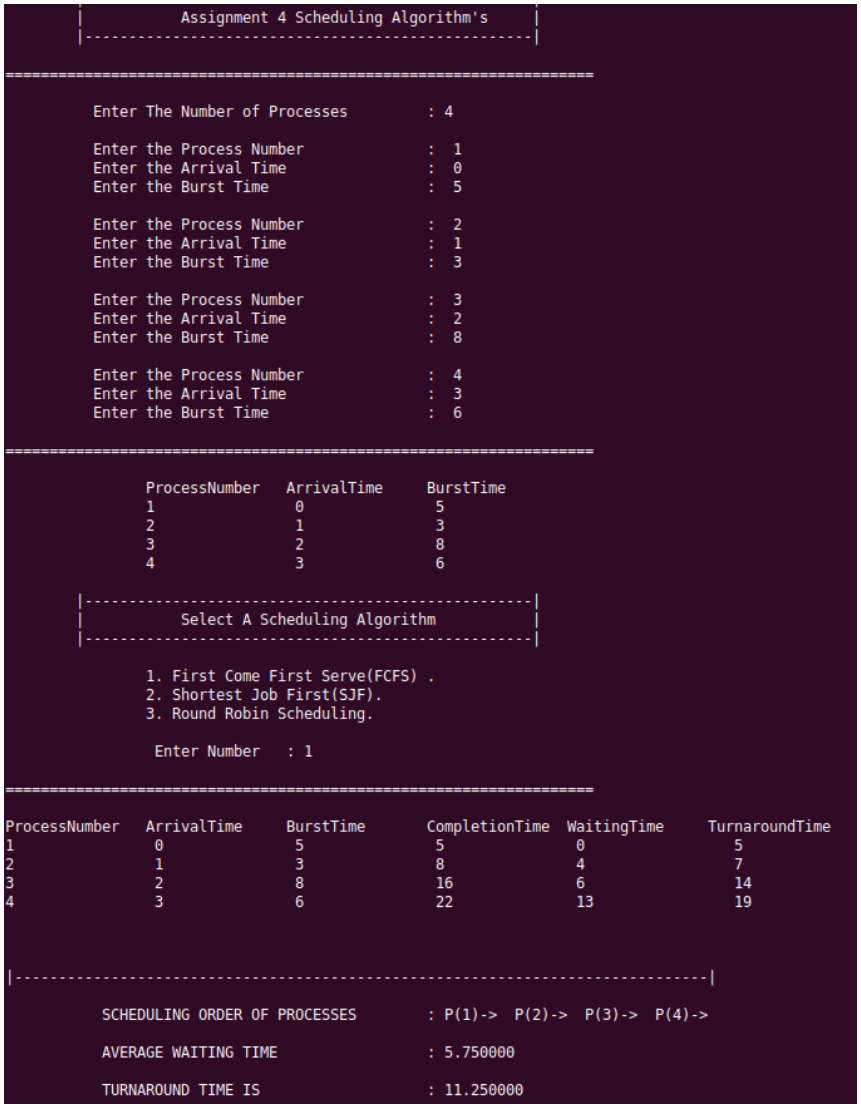
```
    printf("\n\nProcessNumber \tArrivalTime\tBurstTime\tCompletionTime
\tWaitingTime  \tTurnaroundTime \n");
    while(temp!=NULL){
    printf("%-16d %-15d %-15d %-15d %-17d %-14d\n",temp->process_no,temp-
>arrival_time,temp->burst_time,c_time[k],wt[k],ta[k]);
            k++;
            temp = temp->next;

    }

                printf("\n|-----------------------------------------------------------------------------
-|\n\n");
                                        printf("\n\t   AVERAGE WAITING TIME
    : %f\n\n",avgwt/np);
                                        printf("\t   AVG TURNAROUND TIME IS              :
%f\n",tat/np);
                printf("\n|-----------------------------------------------------------------------------
-|\n\n");
}
int main(){
        struct node* first=NULL;
        int d,dec,count;

printf("\n=============================================================
=====\n");
        printf("\n\t|-----------------------------------------------|\n");
        printf("\t|         Assignment 4 Scheduling Algorithm's    |\n");
        printf("\t|-----------------------------------------------|\n\n");
printf("=============================================================
===\n");

         printf("\n\t  Enter The Number of Processes      : ");
        scanf("%d",&np);
        count = np;
       do{
        printf("\n\t  Enter the Process Number            :  ");
        scanf("%d",&s.process_no);
        printf("\t  Enter the Arrival Time              :  ");
        scanf("%d",&s.arrival_time);
        printf("\t  Enter the Burst Time                :  ");
        scanf("%d",&s.burst_time);
```

```
        first = create(s.process_no,s.arrival_time,s.burst_time,first);




        count--;



        }while(count>0);


  printf("\n=================================================================
  =====");

        display(first);
        printf("\n\t|-------------------------------------------------|\n");
        printf("\t|          Select A Scheduling Algorithm          |\n");
        printf("\t|-------------------------------------------------|\n\n");
        printf("\t\t1. First Come First Serve(FCFS) . \n");
        printf("\t\t2. Shortest Job First(SJF). \n");
        printf("\t\t3. Round Robin Scheduling. \n\n");
        printf("\t\t Enter Number      : ");
        scanf("%d",&dec);

  printf("\n=================================================================
  =====");

switch(dec)
        {
                case 1:
                        SA_FCFS(first);
                        break;
                case 2:
                        SA_SJF(first);
                        break;
                case 3:
                        SA_RRobin(first);
                        break;
        }
  return 0;
```

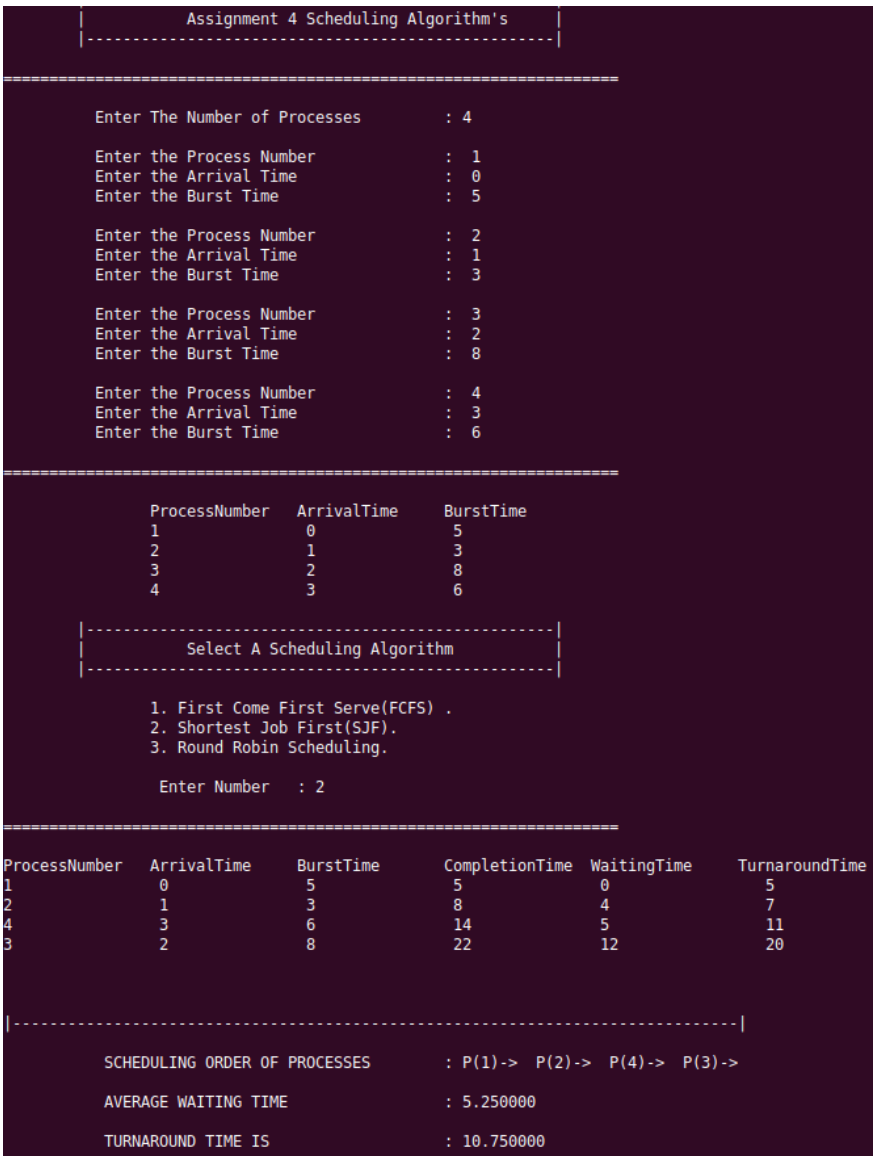
}

OUTPUT :

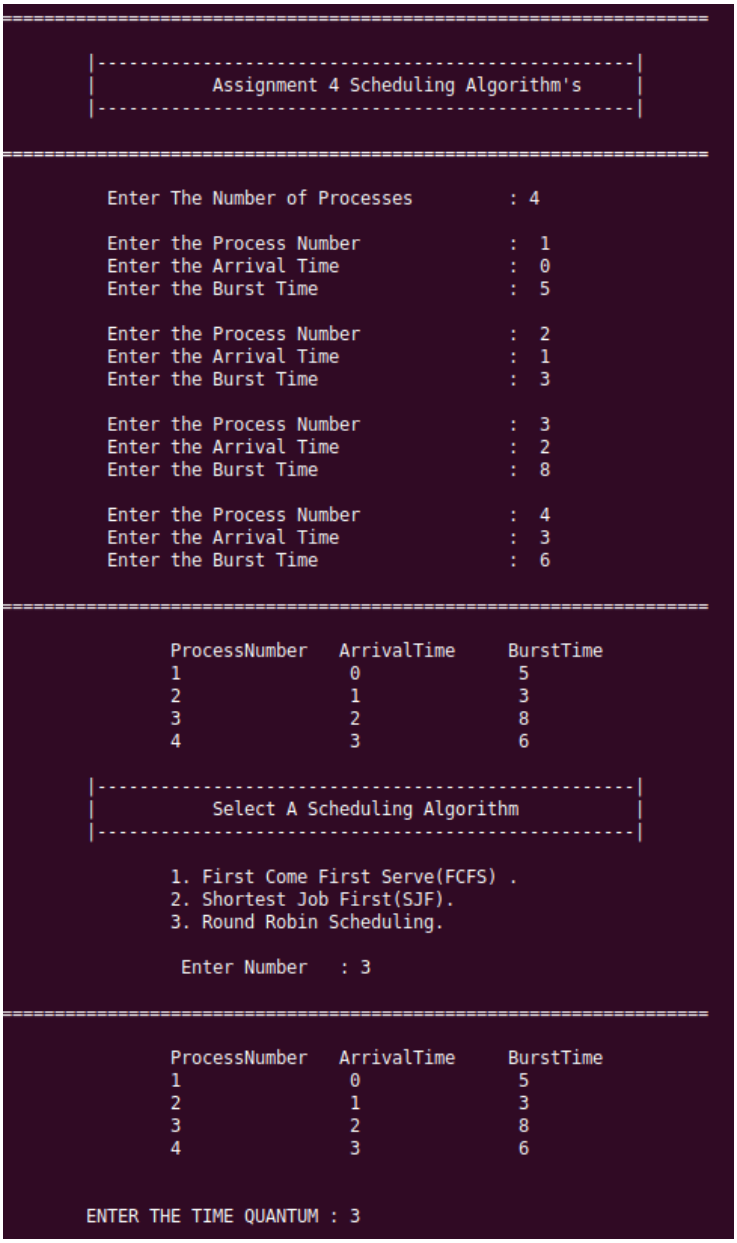
FOR FCFS :



FOR SJF:


Assignment 4 Scheduling Algorithm's
Enter The Number of Processes : 4
Enter the Process Number : 1
Enter the Arrival Time : 0
Enter the Burst Time : 5
Enter the Process Number : 2
Enter the Arrival Time : 1
Enter the Burst Time : 3
Enter the Process Number : 3
Enter the Arrival Time : 2
Enter the Burst Time : 8
Enter the Process Number : 4
Enter the Arrival Time : 3
Enter the Burst Time : 6
ProcessNumber ArrivalTime BurstTime
1 0 5
2 1 3
3 2 8
4 3 6
Select A Scheduling Algorithm
1. First Come First Serve(FCFS) .
2. Shortest Job First(SJF).
3. Round Robin Scheduling.
Enter Number : 2
ProcessNumber ArrivalTime BurstTime CompletionTime WaitingTime TurnaroundTime
1 0 5 5 0 5
2 1 3 8 4 7
4 3 6 14 5 11
3 2 8 22 12 20
SCHEDULING ORDER OF PROCESSES : P(1)-> P(2)-> P(4)-> P(3)->
AVERAGE WAITING TIME : 5.250000
TURNAROUND TIME IS : 10.750000

FOR ROUND ROBIN :



```
        Process Occurrrences :    ->P(1)->P(2)->->P(3)->->P(4)->P(1)->->P(3)->P(4)->P(3)->

ProcessNumber   ArrivalTime     BurstTime       CompletionTime  WaitingTime     TurnaroundTime
1                0              5               14              9                14
2                1              3               6               2                5
3                2              8               22              12               20
4                3              6               20              11               17

|-------------------------------------------------------------------------------|


           AVERAGE WAITING TIME                : 8.500000

           AVG TURNAROUND TIME IS               : 14.000000

|-------------------------------------------------------------------------------|
```