**ASSIGNMENT 4**

**AIM:** To study the scheduling algorithm i.e FCFS,SJF,Round Robin.

**THEORY:**

*Scheduling:*A typical process involves both I/O time and CPU time. In a uni programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Scheduling algorithms are:

1)First come first serve(FCFS)

2)Shortest job first (SJF)

3)Round Robin

*First come first serve(FCFS)*:In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

* First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
* This is used in Batch Systems.
* It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.

*Shortest job first:*Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

* This is the best approach to minimize waiting time.
* This is used in [Batch Systems](https://www.studytonight.com/operating-system/types-of-os).
* It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
* To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
* This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

*Round robin:*

A fixed time is allotted to each process, called **quantum**, for execution.

* Once a process is executed for given time period that process is preemptied and other process executes for given time period.
* Context switching is used to save states of preemptied processes.

**CODE:**

#include<stdlib.h>

#include<stdio.h>

#include<stdbool.h>

typedef struct cpu //structure for node

{

int pid,at,bt;

struct cpu \*next;

bool executed;

int et,tat,wt;

}node;

node \*head=NULL,\*temp,\*current;

//Function to add processes

void add()

{

x:printf("Enter the Process ID, Arrival time, and Burst Time of the process.\n");

temp=malloc(sizeof(node)); //allocation of memory

scanf("%d%d%d" ,&temp->pid,&temp->at,&temp->bt);

temp->executed=false;

temp->next=NULL;

if(head==NULL)

{

head=temp;

head->next=NULL;

}

else

{

current=head;

if(current->pid==temp->pid) //checking if there are repeat process ids

{

printf("Process with the given PID already exists\n");

goto x;

}

while(current->next!=NULL)

{

if(current->pid==temp->pid)

{

printf("Process with the given PID already exists\n");

goto x;

}

current=current->next;

}

if(current->pid==temp->pid)

{

printf("Process with the given PID already exists\n");

goto x;

}

current->next=temp;

current=current->next;

current->next=NULL;

}

}

//Printing the information of process block

void info()

{

printf("PID Arrival Time Burst Time\n");

temp=head;

do{

printf("%d\t\t%d\t\t%d\n",temp->pid,temp->at,temp->bt);

temp=temp->next;

}while(temp!=NULL);

}

//reseting everything

void refresh()

{

current=head;

if(current!=NULL)

{

do

{

current->executed=false; //Setting execution status to false.

current->et=0;

current->tat=0;

current->wt=0;

current=current->next;

} while (current!=NULL);

}

current=head;

}

//calculating average

void calculateavg()

{

int avgturn=0,avgwait=0,n=0;

current=head;

while(current!=NULL)

{

avgturn+=current->tat;

avgwait+=current->wt;

n++;

current=current->next;

}

printf("Average turn-around time = %f\n",(float)avgturn/n);

printf("Average wait time = %f\n",(float)avgwait/n);

}

//Function to implement fcfs

void FCFS()

{

int min,duration=0;

current=head;

if(current==NULL) //Checking if tthere is process or not empty

printf("Process block is empty\n");

else

{

refresh();

min=head->at;

p:current=head;

//giving slots to the unexecured processes based on their at time

while(current!=NULL)

{

if(min>current->at && current->executed==false)

min=current->at;

current=current->next;

}

current=head;

//Printing processes with minimum at time

while(current!=NULL)

{

if(min==current->at)

{

current->executed=true;

current->et=duration+current->bt;

duration+=current->bt;

current->tat=duration-current->at;

current->wt=current->tat-current->bt;

if(current->wt<0)

current->wt=0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,current->et,current->tat,current->wt);

}

current=current->next;

}

current=head;

while(current!=NULL)

{

if(current->executed==false)

{

min=current->at;

goto p;

}

current=current->next;

}

calculateavg();

}

}

//Function to implement sjf

void SJF()

{

int min,minbt,duration=0,pidref=0,minarr;

current=head;

if(current==NULL)

printf("Process block is empty\n");

else

{

refresh(); //if process is not empty

min=current->at;

minbt=32456;

pidref=head->pid;

while(current!=NULL)

{

if((current->at<=min && minbt>current->bt) || current->at<min)

{

min=current->at;

minbt=current->bt;

pidref=current->pid;

}

current=current->next;

}

current=head;

while(current!=NULL)

{

if(pidref==current->pid)

{

current->executed=true;

current->et=duration+current->bt;

duration+=current->bt;

current->tat=duration-current->at;

current->wt=current->tat-current->bt;

if(current->wt<0)

current->wt=0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,current->et,current->tat,current->wt);

break;

}

current=current->next;

}

goto l;

m:current=head;

minarr=1000;

while(current!=NULL)

{

if(min>=current->bt && current->executed==false && current->at<=duration)

{

if(current->bt==min)

{

if(current->at>minarr)

break;

}

min=current->bt;

pidref=current->pid;

minarr=current->at;

}

current=current->next;

}

current=head;

while(current!=NULL)

{

if(current->executed==false && pidref==current->pid)

{

current->executed=true;

current->et=duration+current->bt;

duration+=current->bt;

current->tat=duration-current->at;

current->wt=current->tat-current->bt;

if(current->wt<0)

current->wt=0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,current->et,current->tat,current->wt);

}

current=current->next;

}

l:current=head;

while(current!=NULL)

{

if(current->executed==false)

{

min=30000;

pidref=current->pid;

goto m;

}

current=current->next;

}

calculateavg();

}

}

//Function to implement rr

void RR()

{

if(head==NULL)

printf("Process block is empty\n");

else

{

refresh(); //if block not empty

int executiontime=head->bt,attime=head->at,duration=0;

while(current!=NULL)

{

if(executiontime>current->bt)

executiontime=current->bt;

if(attime>current->at)

attime=current->at;

current=current->next;

}

printf("Execution time alloted is %d.\n" ,executiontime);

printf("PID\tArrival\_time\tBurst\_time\tExecuted\_for\tTurn-around\tWaiting-time\n");

while(current!=NULL)

{

if(current->at==attime)

{

if(executiontime>current->bt)

{

current->executed=true;

duration+=current->bt;

current->tat=duration-current->at;

current->wt=current->tat-current->bt;

if(current->wt<0)

current->wt=0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,current->bt,current->tat,current->wt);

current->et+=current->bt;

}

else

{

duration+=executiontime;

current->et+=executiontime;

printf("%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,executiontime);

}

int s1,s2,s3,s4;

bool check;

s1=current->at;

s2=current->bt;

s3=current->et;

s4=current->pid;

check=current->executed;

current->at=head->at;

current->bt=head->bt;

current->executed=head->executed;

current->et=head->et;

current->pid=head->pid;

head->at=s1;

head->bt=s2;

head->et=s3;

head->pid=s4;

head->executed=check;

break;

}

current=current->next;

}

current=head;

if(current->et>0)

current=current->next;

z:while (current!=NULL)

{

if(current->at<=duration && current->executed==false)

{

if(executiontime>=(current->bt)-(current->et))

{

current->executed=true;

duration+=current->bt-current->et;

current->tat=duration-current->at;

current->wt=current->tat-current->bt;

if(current->wt<0)

current->wt=0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,(current->bt-current->et),current->tat,current->wt);

current->et+=current->bt;

}

else

{

duration+=executiontime;

current->et+=executiontime;

printf("%d\t\t%d\t\t%d\t\t%d\n",current->pid,current->at,current->bt,executiontime);

}

}

current=current->next;

}

current=head;

while(current!=NULL)

{

if(current->executed==false)

{

current=head;

goto z;

}

current=current->next;

}

calculateavg();

}

}

int main()

{

int choice;

p: printf("Enter\n1: Add information.\n2. Display all processes.\n3. FCFS\n4. Non-Preemptive SJF\n5. Round Robin\n6. Exit.\n");

scanf("%d" ,&choice);

switch(choice)

{

case 1:

add();

printf("Process added\n");

break;

case 2:

printf("The list is:\n\n");

info();

break;

case 3:

printf("PID\tArrival time\tBurst time\tExecution finished:\tTurn-around\tWaiting-time\n");

FCFS();

break;

case 4:

printf("PID\tArrival time\tBurst time\tExecution finished:\tTurn-arount\tWaiting-time\n");

SJF();

break;

case 5:

RR();

break;

case 6:

printf("Exiting\n");

exit(0);

break;

default:

printf("Enter correct choice\n");

}

choice=0;

printf("Do you want to continue?If yes,enter 1\n");

scanf("%d",&choice);

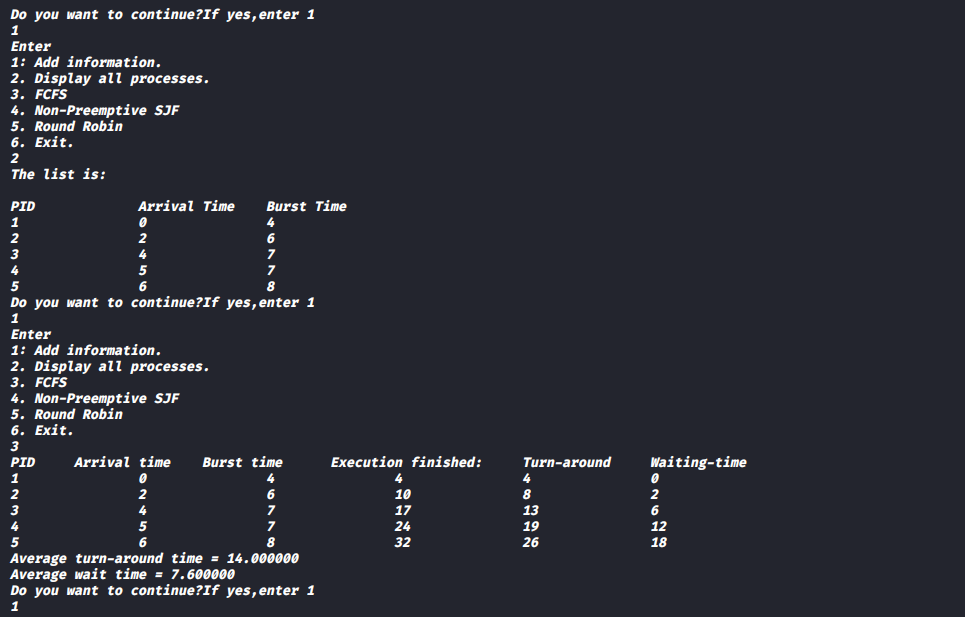
if(choice==1)

goto p;

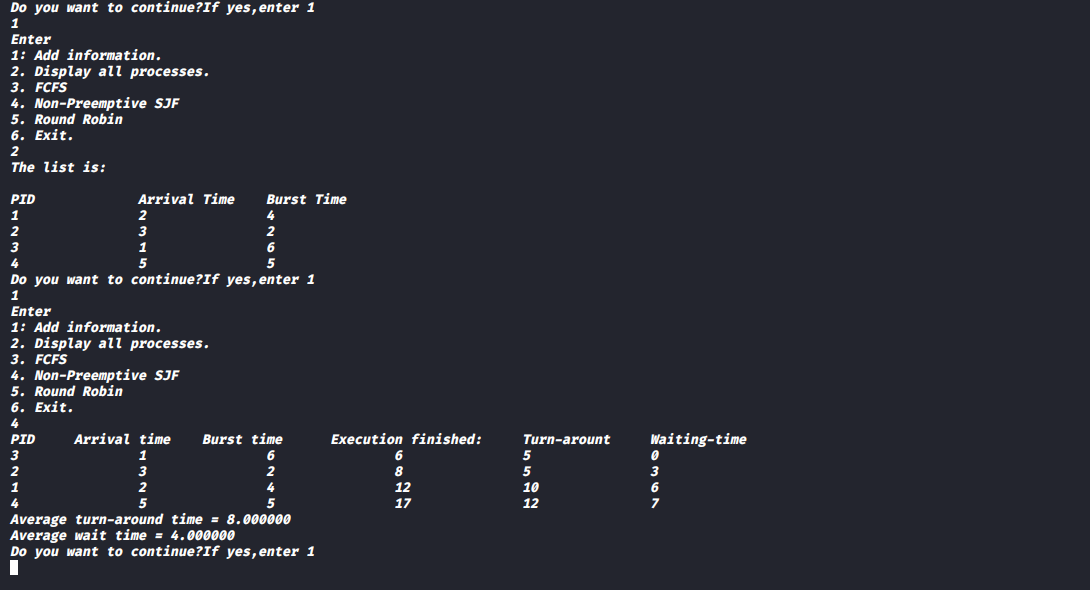
return 0;

}

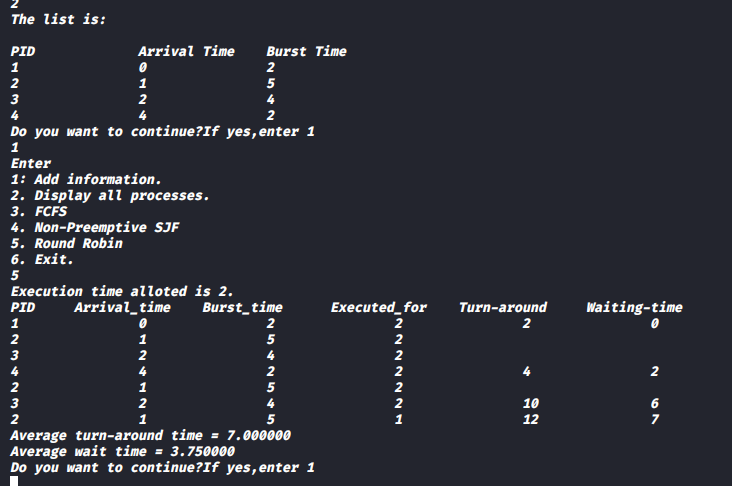
**FCFS :**



**SJF:**

****

**RR:**

****

**Conclusion:**

In this assignment we learn how the processes are scheduled in the operating system using different scheduling mechanisms:1)FCFS(First Come First Serve)

2)SJF(Shortest Job First) 3)RR(Round Robin) and also the working of these mechanisms and their attributes.