**SIMULATION BASED ASSIGNMENT ASSESMENT**

**BACHELOR OF TECHNOLOGY**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**Name: Kajal Singh**

**Email: kajalsinghkr47@gmail.com**

**Registration number: 11804306**

**Roll No: 06**

**Section: K18BY**

****

**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

**Problem:**

An auto driver charges Rs 10 per customer irrespective of the distance the customer has to travel within the city. All customers are picked up from common point where the auto driver has to return after dropping customer to destination. Considering N customers request for auto service mentioning the distance(measured equivalent to time units)each of them wants to travel,write a computer program that will help the auto driver earn maximum amount of money by travelling least possible distance (equivalent to time units) and that average time for which customers wait for their pickup by auto driver is also as least as possible.

**Explanation:**

This program uses the concept of Shortest Job First(SJF) algorithm. SJF is a scheduling algorithm to each process the length pf its next CPU burst time. CPU is then given to the process with the minimal CPU burst from the waiting queue.

Completion time: Time at which process completes in execution.

Turnaround time: Time difference between completion time and arrival time

Turnaround time = completion time – arrival time

Waiting time: Time difference between turn around time and burst time

Waiting time = turnaround time – burst time.

**Algorithm:**

1. Traverse until all process gets completely executed.
2. Find process with minimum remaining time at every single time lap.
3. Reduce its time by 1.
4. Traverse Check if its remaining time becomes 0.
5. Increment the counter of process completion.
6. Completion time of current process = current\_time +1;
7. Calculate waiting time for each completed process.

Waiting time[i] = completion time – arrival\_time – burst time

1. Increment time lap by one.
2. Find turnaround time

Turnaround time = waiting\_time + burst\_time

**Constraints:**

1.The time taken by a process must be known by CPU.

2.Longer process will have more waiting time,eventually they will suffer starvation.

3. It does not always minimize average turn around time .

**CODE:**

**#include<stdio.h>**

**#include<unistd.h>**

**#include<sys/wait.h>**

**#include<stdlib.h>**

**#define NEW 0**

**#define READY 1**

**#define WAITING 2**

**#define RUNNING 3**

**#define EXIT 4**

**int moneyearned=0;**

**int ongoingtime=0,time\_till\_last\_ct=0,EXIT\_STATE\_ALL=0,CT\_ALLOWED=1,n,i,j;**

**struct Process\_structure**

**{**

**int pid;**

**int state;**

**int timeleft\_to\_exec;**

**int at;**

**int wt,tat,ct,exect;**

**struct Process\_structure \*prev;**

**struct Process\_structure \*next;**

**} \*proc\_arr;**

**int chpp;**

**struct Queue**

**{**

**struct Process\_structure \*front ,\*rear;**

**}\*RQueue;**

**void enqueue(struct Process\_structure \*proc)**

**{**

**if(RQueue->front==NULL)**

**{**

**RQueue->front=proc;**

**RQueue->rear=proc;**

**proc->next=NULL;**

**}**

**else**

**{**

**if(proc->timeleft\_to\_exec<RQueue->front->timeleft\_to\_exec)**

**{**

**proc->next=RQueue->front;**

**RQueue->front->prev=proc;**

**RQueue->front=proc;**

**}**

**else if(proc->timeleft\_to\_exec==RQueue->front->timeleft\_to\_exec)**

**{**

**proc->next=RQueue->front->next;**

**proc->prev=RQueue->front;**

**RQueue->front->next=proc;**

**if(proc->next!=NULL)**

**{**

**proc->next->prev=proc;**

**}**

**}**

**else if(proc->timeleft\_to\_exec>RQueue->rear->timeleft\_to\_exec)**

**{**

**proc->next=NULL;**

**RQueue->rear->next=proc;**

**proc->prev=RQueue->rear;**

**RQueue->rear=proc;**

**}**

**else**

**{**

**struct Process\_structure \*start=RQueue->front->next;**

**while(start->timeleft\_to\_exec<proc->timeleft\_to\_exec)**

**{**

**start=start->next;**

**}**

**if(start!=NULL&& proc->timeleft\_to\_exec==start->timeleft\_to\_exec)**

**{**

**proc->next=start->next;**

**start->next=proc;**

**proc->prev=start;**

**}**

**else**

**{**

**(start->prev)->next=proc;**

**proc->next=start;**

**proc->prev=start->prev;**

**start->prev=proc;**

**}**

**}**

**}**

**}**

**struct Process\_structure \* deQueue()**

**{**

**if(RQueue->front==NULL)**

**{**

**return NULL;**

**}**

**struct Process\_structure \* temp=RQueue->front;**

**RQueue->front=RQueue->front->next;**

**temp->next=NULL;**

**if(RQueue->front==NULL)**

**{**

**RQueue->rear=NULL;**

**}**

**return temp;**

**}**

**void CQ()**

**{**

**int cnt=0;**

**for(i=0;i<n;i++)**

**{**

**if(proc\_arr[i].state == NEW && ongoingtime>=proc\_arr[i].at)**

**{**

**enqueue(&proc\_arr[i]);**

**proc\_arr[i].state=READY;**

**}**

**if(proc\_arr[i].state==EXIT)**

**{**

**cnt++;**

**}**

**}**

**if(cnt==n)**

**{**

**EXIT\_STATE\_ALL=1;**

**}**

**}**

**int main()**

**{**

**RQueue =(struct Queue\*) malloc(sizeof(struct Queue));**

**printf(" Please enter No of Passangers =");**

**scanf("%d",&n);**

**proc\_arr=(struct Process\_structure \*)malloc(sizeof(struct Process\_structure)\*n);**

**for(i=0;i<n;i++)**

**{**

**printf("\n\n Enter Process Id For %d Passanger =",(i+1));**

**scanf("%d",&(proc\_arr[i].pid));**

**printf("\n Enter arrival time For %d Passanger =",(i+1));**

**scanf("%d",&(proc\_arr[i].at));**

**printf("\n Enter Distance to travel time For %d Passanger=",(i+1));**

**scanf("%d",&(proc\_arr[i].timeleft\_to\_exec));**

**proc\_arr[i].timeleft\_to\_exec\*=2;**

**proc\_arr[i].exect=proc\_arr[i].timeleft\_to\_exec;**

**proc\_arr[i].state=NEW;**

**}**

**struct Process\_structure key;**

**for (i = 1; i < n; i++) {**

**key = proc\_arr[i];**

**j = i - 1;**

**while (j >= 0 && proc\_arr[j].at > key.at) {**

**proc\_arr[j + 1] = proc\_arr[j];**

**j = j - 1;**

**}**

**proc\_arr[j + 1] = key;**

**}**

**struct Process\_structure \*pr;**

**printf("Gannt Chart\n====================================================================================================================\n");**

**while(1)**

**{**

**CQ();**

**if(EXIT\_STATE\_ALL==1)**

**{**

**break;**

**}**

**if(RQueue->front!=NULL && CT\_ALLOWED==1)**

**{**

**time\_till\_last\_ct=1;**

**pr=deQueue();**

**moneyearned+=10;**

**printf(" %d| Passanger Id:%d [Drop+Return]|",ongoingtime,pr->pid);**

**pr->state=RUNNING;**

**pr->timeleft\_to\_exec--;**

**ongoingtime++;**

**if(time\_till\_last\_ct==pr->exect)**

**{**

**CT\_ALLOWED=1;**

**pr->state=EXIT;**

**pr->ct=ongoingtime;**

**pr->tat=ongoingtime-pr->at;**

**pr->wt=pr->tat-pr->exect;**

**}**

**else**

**{**

**CT\_ALLOWED=0;**

**}**

**}**

**else if(CT\_ALLOWED==0&&pr!=NULL && pr->state==RUNNING )**

**{**

**if(pr->timeleft\_to\_exec==0)**

**{**

**CT\_ALLOWED=1;**

**pr->state=EXIT;**

**pr->ct=ongoingtime;**

**pr->tat=ongoingtime-(pr->at);**

**pr->wt=(pr->tat)-(pr->exect);**

**continue;**

**}**

**time\_till\_last\_ct++;**

**pr->timeleft\_to\_exec--;**

**ongoingtime++;**

**if(pr->timeleft\_to\_exec==0)**

**{**

**CT\_ALLOWED=1;**

**pr->state=EXIT;**

**pr->ct=ongoingtime;**

**pr->tat=ongoingtime-(pr->at);**

**pr->wt=(pr->tat)-(pr->exect);**

**}**

**else**

**{**

**CT\_ALLOWED=0;**

**}**

**}**

**else**

**{**

**printf(" %d| IDLE |",ongoingtime);**

**ongoingtime++;**

**}**

**}**

**printf("%d|\n",ongoingtime);**

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

**int sumwt=0,sumtat=0;**

**for(i=0;i<n;i++)**

**{**

**printf("\n\nprocess pid=%d\tct=%d\ttat=%d\twt=%d",proc\_arr[i].pid,proc\_arr[i].ct,proc\_arr[i].tat,proc\_arr[i].wt);**

**sumwt+=proc\_arr[i].wt;**

**sumtat+=proc\_arr[i].tat;**

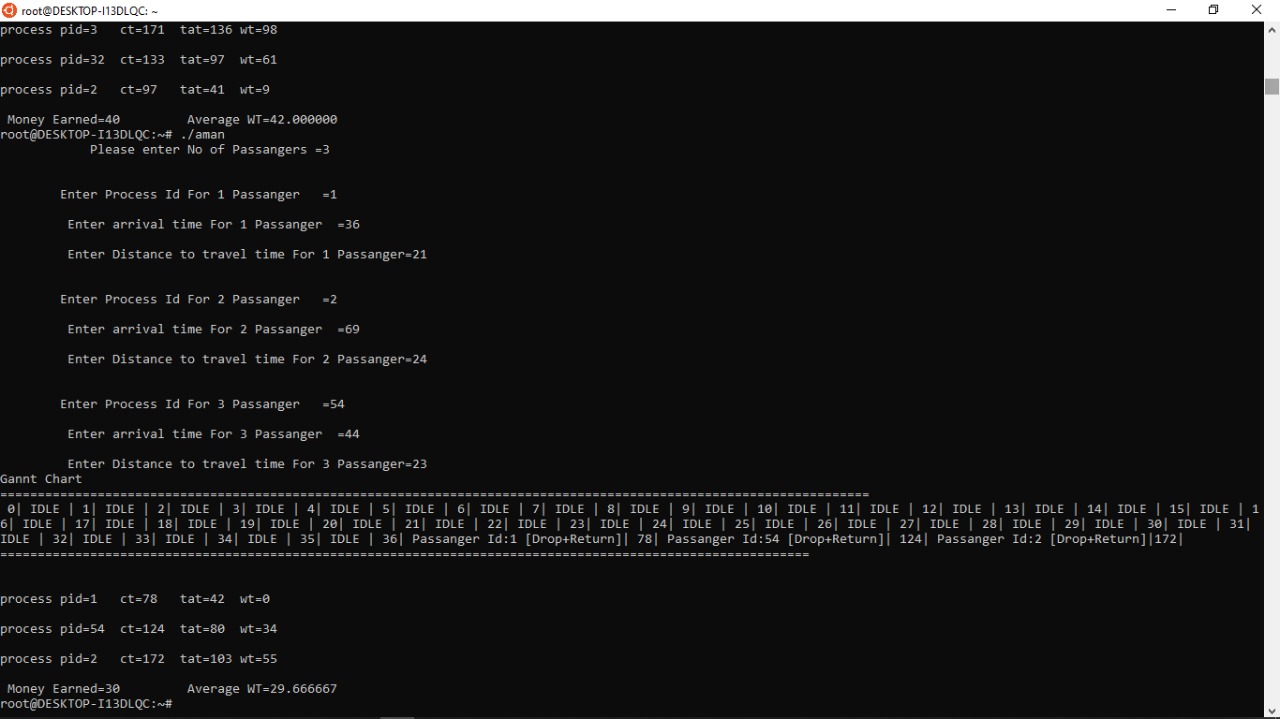
**}**

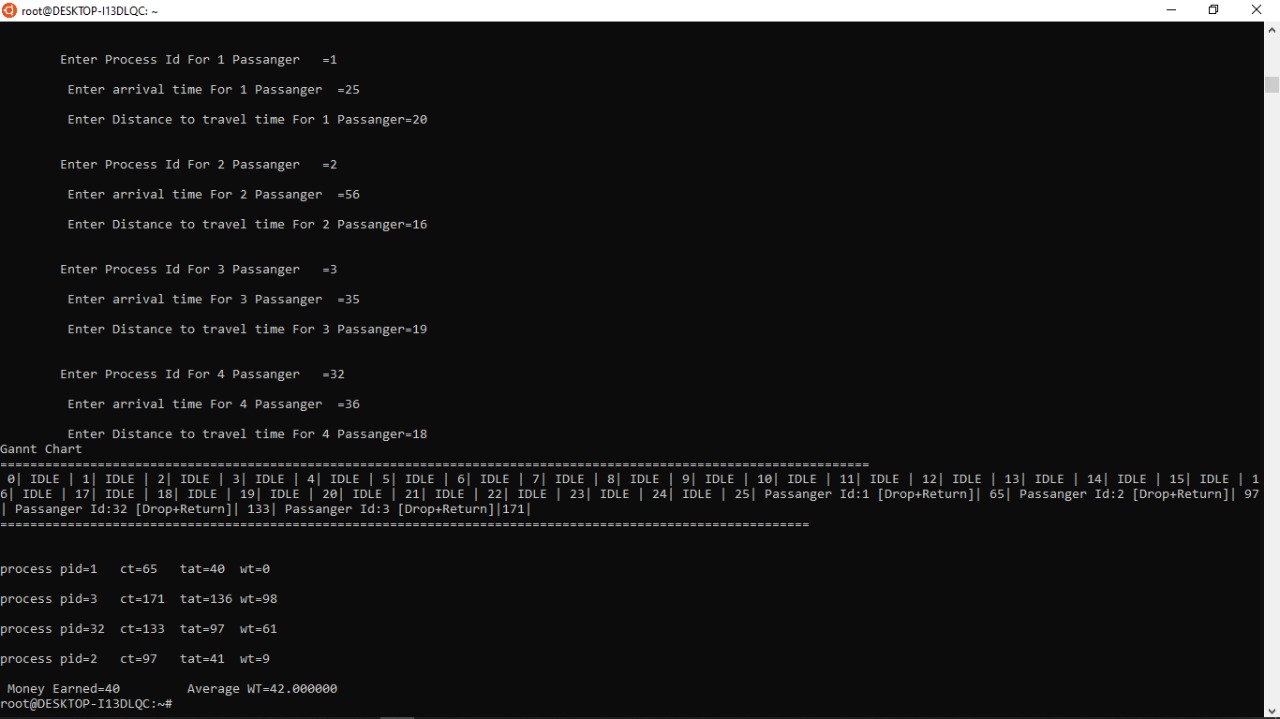
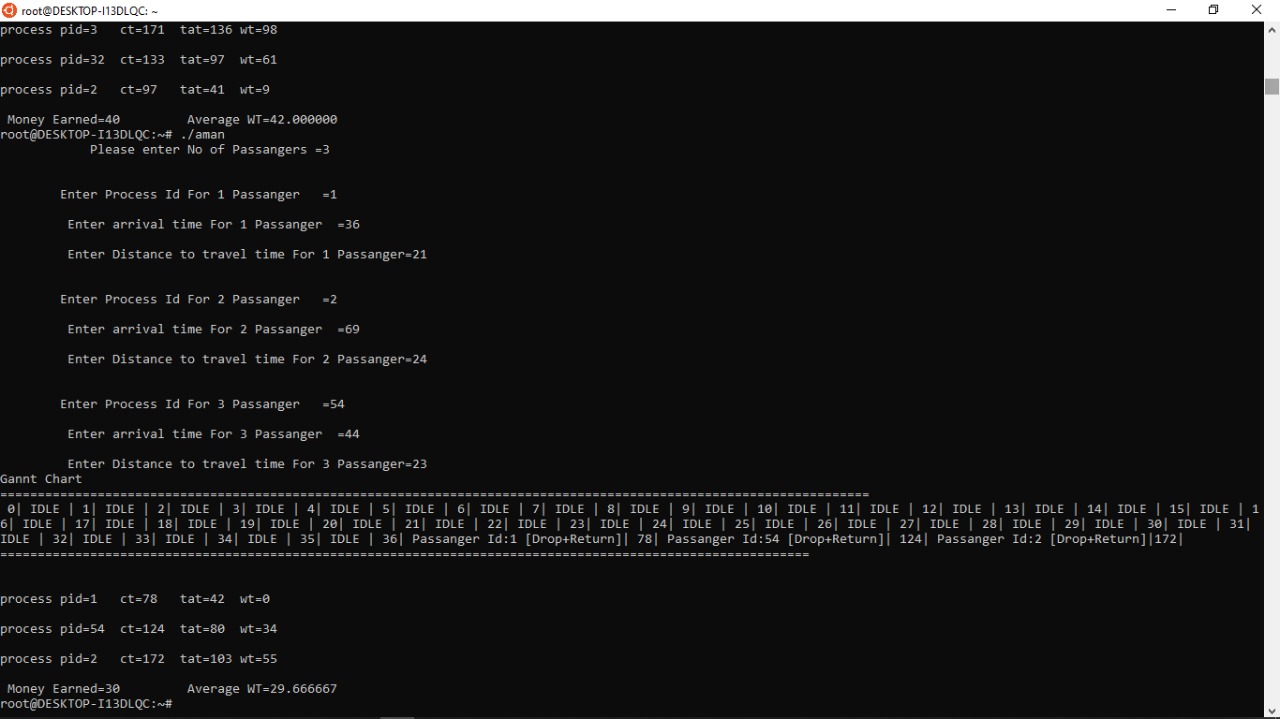
**printf("\n\n Money Earned=%d \t Average WT=%f\n",moneyearned,(sumwt/(n\*1.0)));**

**}**

**Complexity pf implemented algorithm:-**

Average case time complexity of SJF algorithm is O(logn).

**Code Snippet:-** 



**GitHub Revisions done: 5**

**Link to git repository:**