**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**OPERATING SYSTEMS**

***Submitted by***

**ARCHIT MEHROTRA (1BM21CS031)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**June-2023 to September-2023**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “OPERATING SYSTEMS” carried out by **ARCHIT MEHROTRA (1BM21CS031),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS **(22CS4PCOPS)** work prescribed for the said degree.

Name of the Lab-In charge:               Dr. Jyothi S Nayak

Designation Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Lab Program No.** | **Program Details** | **Page No.** |
| 1 | WEEK 1 (Matrices) | 5-11 |
| 2 | WEEK 2 (FCFS) | 12-14 |
| 3 | WEEK 3 (Priority Scheduling Non Pre-emptive, SJR Non Pre-emptive, SRTF) | 15-23 |
| 4 | WEEK 4 (Priority Scheduling Pre-emptive, Round Robin) | 24-29 |
| 5 | WEEK 5 (Multilevel, Rate Monotonic Scheduling) | 30-36 |
| 6 | WEEK 6 (Producer Consumer) | 37-45 |
| 7 | WEEK 7 (Bankers Algorithm) | 46-48 |
| 8 | WEEK 8 (Best, Worst, First Hit) | 49-54 |
| 9 | WEEK 9 (Disc Scheduling, Paging) | 55-72 |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Apply the different concepts and functionalities of Operating System |
| CO2 | Analyse various Operating system strategies and techniques |
| CO3 | Demonstrate the different functionalities of Operating System |
| CO4 | Conduct practical experiments to implement the functionalities of Operating system. |

**WEEK 1**

**Matrices**

#include<stdio.h>

#include<stdlib.h>

int A[3][3];

int B[3][3];

int C[3][3];

void add(int a[3][3],int b[3][3])

{

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

C[i][j]=a[i][j]+b[i][j];

printf("Resultant matrix\n");

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

printf("%d\t",C[i][j]);

printf("\n");

}

}

void subtract(int a[3][3],int b[3][3])

{

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

C[i][j]=a[i][j]-b[i][j];

printf("Resultant matrix\n");

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

printf("%d\t",C[i][j]);

printf("\n");

}

}

void transpose(int a[3][3])

{

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

C[i][j]=a[j][i];

printf("\n");

}

printf("Resultant matrix\n");

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

printf("%d\t",C[i][j]);

printf("\n");

}

}

void multiply(int a[3][3],int b[3][3])

{

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

{

C[i][j]=0;

for(int k=0;k<3;k++)

C[i][j]+=a[i][k]\*b[k][j];

}

}

printf("Resultant matrix\n");

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

printf("%d\t",C[i][j]);

printf("\n");

}

}

int main()

{

printf("enter the elements for matrix A\n");

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

scanf("%d",&A[i][j]);

printf("enter the elements for matrix B\n");

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

scanf("%d",&B[i][j]);

printf("matrix A\n");

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

printf("%d\t",A[i][j]);

printf("\n");

}

printf("matrix B\n");

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

printf("%d\t",B[i][j]);

printf("\n");

}

int ch=0;

while(ch!=5)

{

printf("1.add\n2.subtract\n3.transpose\n4.multiply\n5.exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

add(A,B);

break;

case 2:

subtract(A,B);

break;

case 3:

printf("enter matrix to transpose(A->1/B->2)\n");

int c1;

scanf("%d",&c1);

if(c1==1)

{

transpose(A);

break;

}

else

{

transpose(B);

break;

}

break;

case 4:

multiply(A,B);

break;

case 5:

exit(0);

break;

default:

printf("wrong choice entered\n");

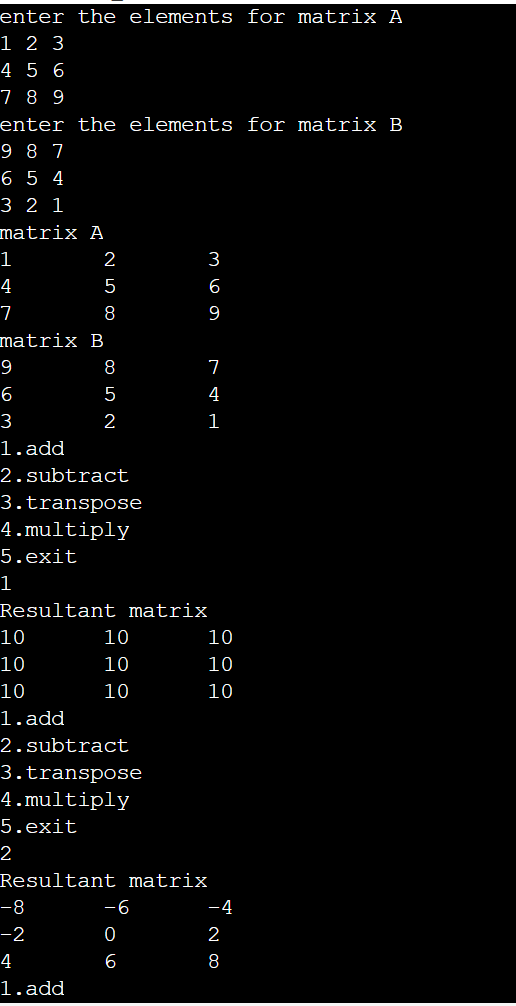
break;

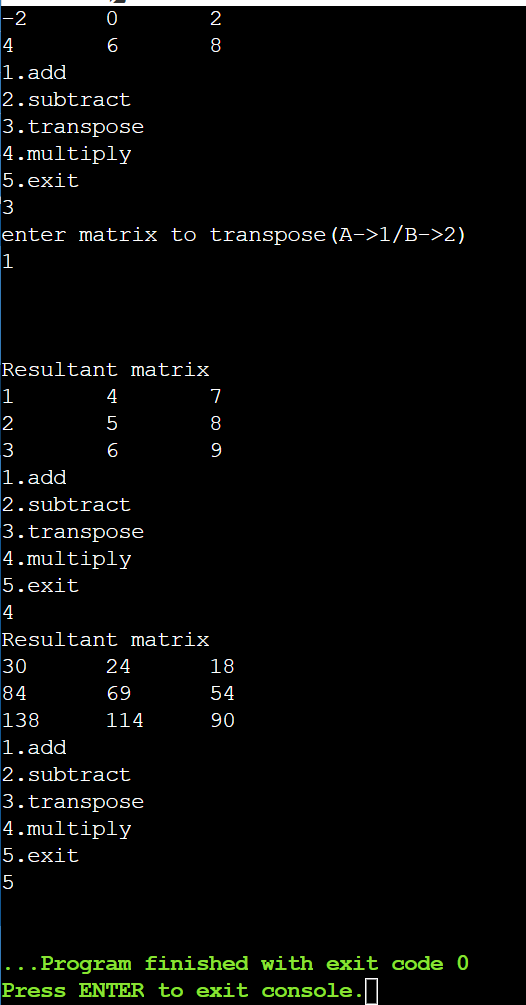
}

}

}

OUTPUT





**WEEK 2**

**FCFS**

#include <stdio.h>

struct process

{

int B\_time;

int arr\_time;

int turnAround\_time;

int waiting\_time;

};

typedef struct process procs;

int main()

{

int n;

printf("enter the number of the processes\n");

scanf("%d",&n);

procs processes[n];

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

{

printf("enter the burst time for process:%d\n",i+1);

scanf("%d",&processes[i].B\_time);

printf("enter the arrival time for process:%d\n",i+1);

scanf("%d",&processes[i].arr\_time);

}

int comp\_time=0;

int avg\_tat=0;

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

{

comp\_time+=processes[i].B\_time;

processes[i].turnAround\_time=comp\_time;

processes[i].arr\_time;

avg\_tat+=processes[i].turnAround\_time;

}

printf("printing turn around time\n");

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

printf("the Turn Around time for process:%d is %d\n",i+1,processes[i].turnAround\_time);

printf("avg turn around time of all the processes is %d\n",avg\_tat/n);

int avg\_wait=0;

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

{

processes[i].waiting\_time=processes[i].turnAround\_time;

processes[i].B\_time;

avg\_wait+=processes[i].waiting\_time;

}

printf("\n printing waiting time\n");

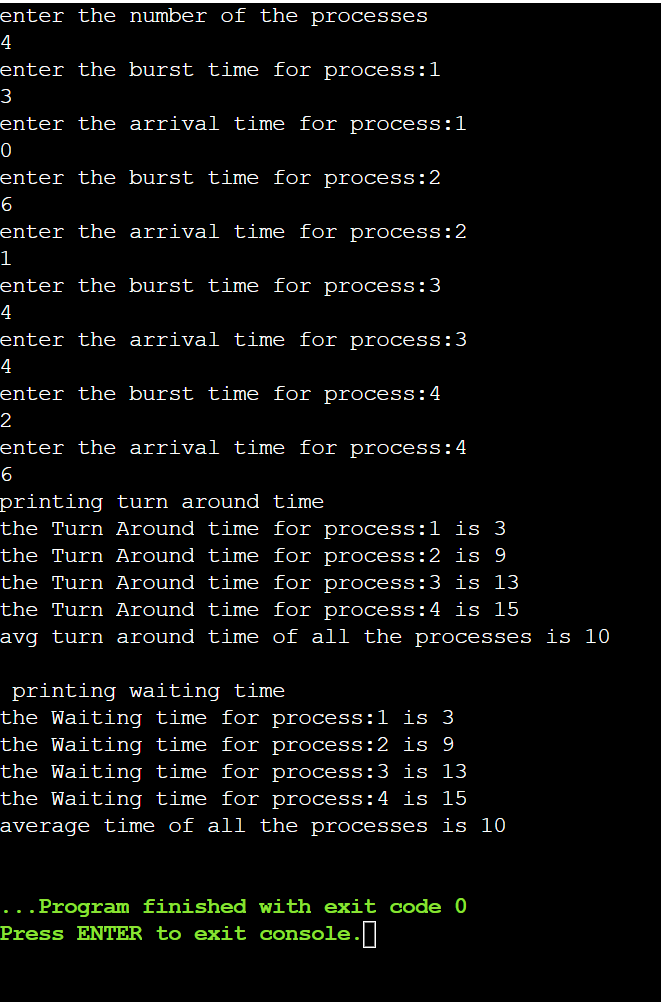
for(int i=0;i<n;i++)

printf("the Waiting time for process:%d is %d\n",i+1,processes[i].waiting\_time);

printf("average time of all the processes is %d\n",avg\_wait/n);

return 0;

}



**WEEK 3**

**Priority Scheduling(Non Pre-emtive)**

#include <stdio.h>

struct process

{

int burst;

int arr\_time;

int waiting\_time;

int priority;

int turn\_time;

};

typedef struct process proc;

void priority(proc processes[],int n)

{

int comp\_time=0;

float avg\_tat=0;

float avg\_wait=0;

proc temp;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(processes[j+1].priority<processes[j].priority)

{

temp=processes[j];

processes[j]=processes[j+1];

processes[j+1]=temp;

}

}

}

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

{

comp\_time+=processes[i].burst;

processes[i].turn\_time=comp\_time-processes[i].arr\_time;

avg\_tat+=processes[i].turn\_time;

}

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

{

processes[i].waiting\_time=processes[i].turn\_time-processes[i].burst;

avg\_wait+=processes[i].waiting\_time;

}

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

{

printf("\nburst, arrival time for process:%d\t",i+1);

printf("%d\t",processes[i].burst);

printf("%d\t",processes[i].arr\_time);

printf("%d\t",processes[i].turn\_time);

printf("%d\n",processes[i].waiting\_time);

}

printf("average waiting time: %f\n",avg\_wait/n);

printf("average turn around time: %f\n",avg\_tat/n);

}

int main()

{

int n;

printf("enter the number of processes:\t");

scanf("%d",&n);

proc processes[n];

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

{

printf("enter the burst, arrival time, priority for process:%d\n",i+1);

scanf("%d",&processes[i].burst);

scanf("%d",&processes[i].arr\_time);

scanf("%d",&processes[i].priority);

}

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

{

printf("burst, arrival time for process:%d\t",i+1);

printf("%d\t",processes[i].burst);

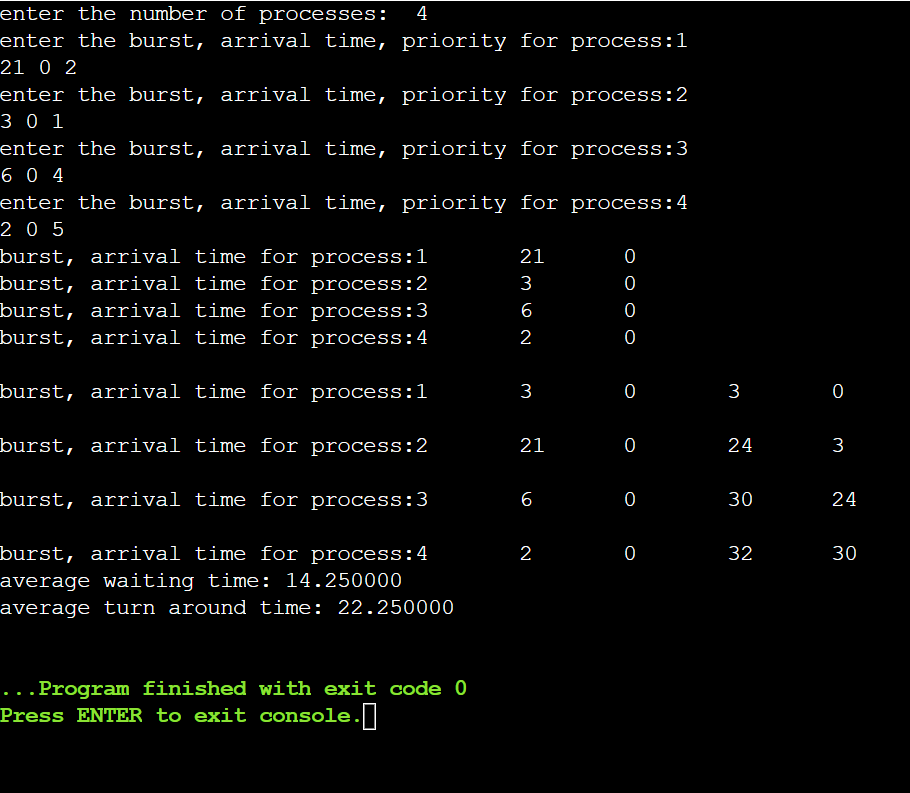
printf("%d\n",processes[i].arr\_time);

}

priority(processes,n);

}

**OUTPUT**



**SJF(Non Pre-emtive)**

#include <stdio.h>

struct process

{

int burst;

int arr\_time;

int waiting\_time;

int turn\_time;

};

typedef struct process proc;

void sjf(proc processes[],int n)

{

int comp\_time=0;

float avg\_tat=0;

float avg\_wait=0;

proc temp;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(processes[j+1].burst<processes[j].burst)

{

temp=processes[j];

processes[j]=processes[j+1];

processes[j+1]=temp;

}

}

}

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

{

comp\_time+=processes[i].burst;

processes[i].turn\_time=comp\_time-processes[i].arr\_time;

avg\_tat+=processes[i].turn\_time;

}

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

{

processes[i].waiting\_time=processes[i].turn\_time-processes[i].

burst;

avg\_wait+=processes[i].waiting\_time;

}

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

{

printf("\nburst, arrival time for process:%d\t",i+1);

printf("%d\t",processes[i].burst);

printf("%d\t",processes[i].arr\_time);

printf("%d\t",processes[i].turn\_time);

printf("%d\n",processes[i].waiting\_time);

}

printf("average waiting time: %f\n",avg\_wait/n);

printf("average turn around time: %f\n",avg\_tat/n);

}

int main()

{

int n;

printf("enter the number of processes:\t");

scanf("%d",&n);

proc processes[n];

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

{

printf("enter the burst, arrival time for process:%d\n",i+1);

scanf("%d",&processes[i].burst);

scanf("%d",&processes[i].arr\_time);

}

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

{

printf("burst, arrival time for process:%d\t",i+1);

printf("%d\t",processes[i].burst);

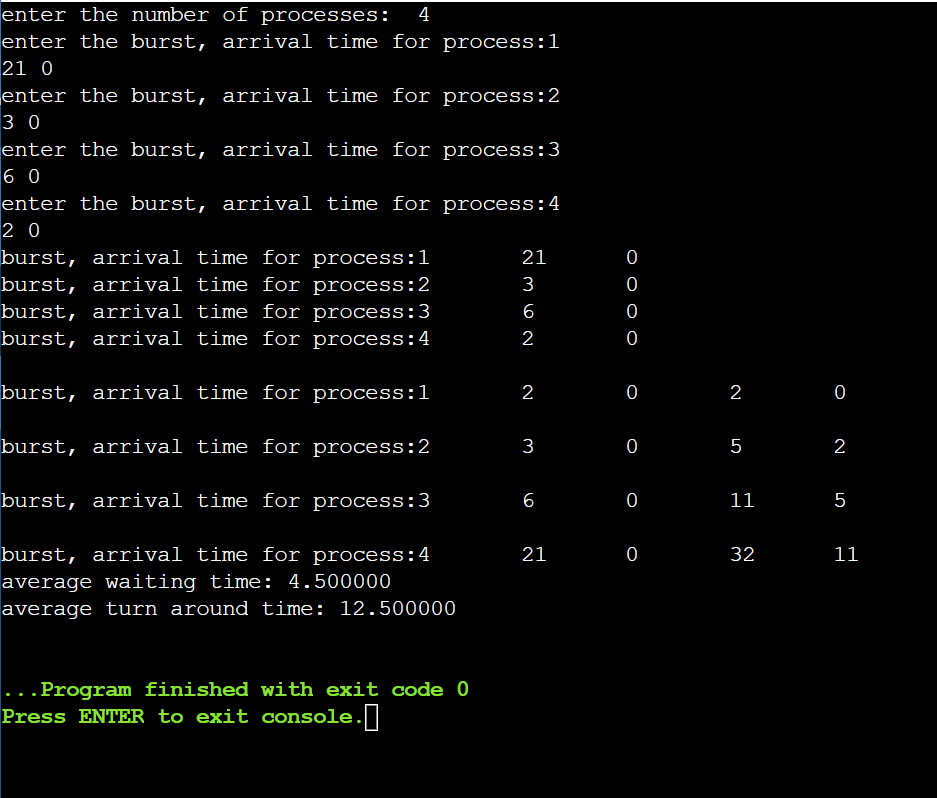
printf("%d\n",processes[i].arr\_time);

}

sjf(processes,n);

}

**OUTPUT**



**SRTF**

#include <stdio.h>

int main()

{

int a[10],b[10],x[10],i,j,smallest,count=0,time,n;

double avg=0,tt=0,end;

printf("enter the number of Processes:\n");

scanf("%d",&n);

printf("enter arrival time\n");

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

scanf("%d",&a[i]);

printf("enter burst time\n");

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

scanf("%d",&b[i]);

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

x[i]=b[i];

b[9]=9999;

for(time=0;count!=n;time++)

{

smallest=9;

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

{

if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )

smallest=i;

}

b[smallest]--;

if(b[smallest]==0)

{

count++;

end=time+1;

avg=avg+end-a[smallest]-x[smallest];

tt= tt+end-a[smallest];

}

}

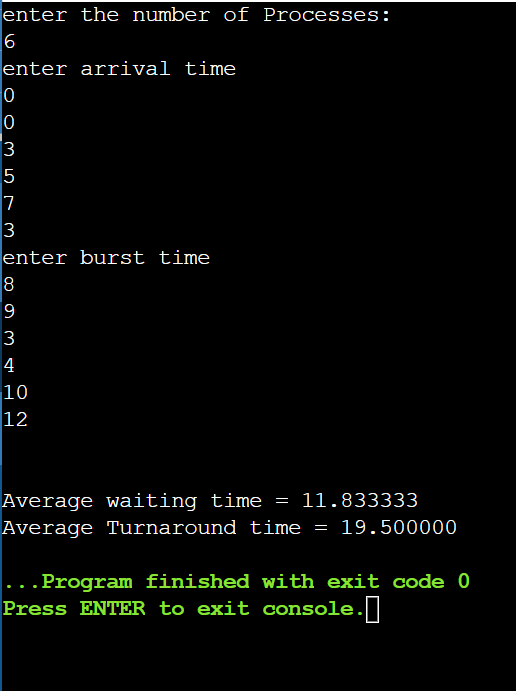
printf("\n\nAverage waiting time = %lf\n",avg/n);

printf("Average Turnaround time = %lf",tt/n);

return 0;

}

**OUTPUT**



**WEEK 4**

**Priority Scheduling (Pre-emtive)**

#include<stdio.h>

struct process

{

int WT,AT,BT,TAT,PT;

};

struct process a[10];

int main()

{

int n,temp[10],t,count=0,short\_p;

float total\_WT=0,total\_TAT=0,Avg\_WT,Avg\_TAT;

printf("Enter the number of the process\n");

scanf("%d",&n);

printf("Enter the arrival time , burst time and priority of the process\n");

printf("AT BT PT\n");

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

{

scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

temp[i]=a[i].BT;

}

a[9].PT=10000;

for(t=0;count!=n;t++)

{

short\_p=9;

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

if(a[short\_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)

short\_p=i;

a[short\_p].BT=a[short\_p].BT-1;

if(a[short\_p].BT==0)

{

count++;

a[short\_p].WT=t+1-a[short\_p].AT-temp[short\_p];

a[short\_p].TAT=t+1-a[short\_p].AT;

total\_WT=total\_WT+a[short\_p].WT;

total\_TAT=total\_TAT+a[short\_p].TAT;

}

}

Avg\_WT=total\_WT/n;

Avg\_TAT=total\_TAT/n;

printf("ID WT TAT\n");

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

printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);

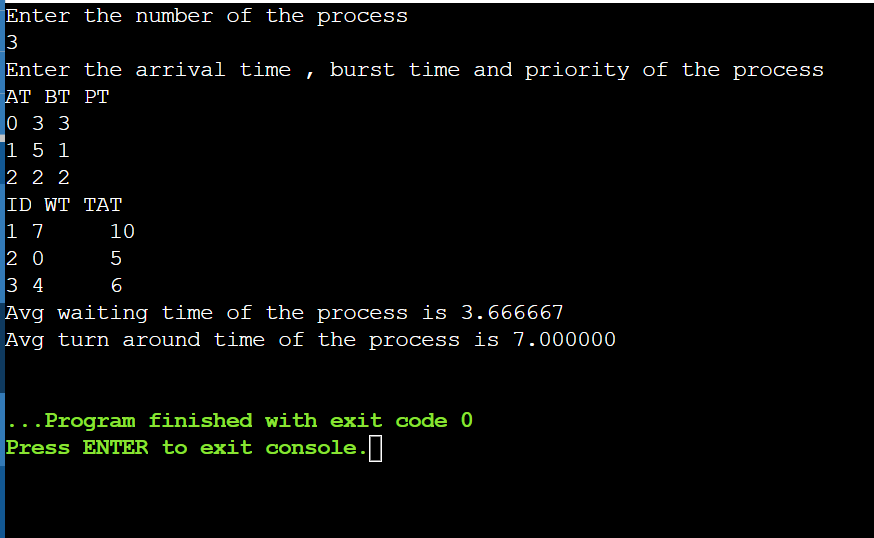
printf("Avg waiting time of the process is %f\n",Avg\_WT);

printf("Avg turn around time of the process is %f\n",Avg\_TAT);

return 0;

}

**OUTPUT**



**Round Robin**

#include <stdio.h>

#define MAX\_SIZE 100

struct Process

{

int pid;

int burst\_time;

int remaining\_time;

int waiting\_time;

};

void RoundRobin(struct Process processes[], int n, int time\_quantum)

{

int total\_time = 0;

int completed = 0;

int ready\_queue[MAX\_SIZE];

int front = 0, rear = -1;

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

ready\_queue[++rear] = i;

while (completed < n)

{

int current\_process = ready\_queue[front++];

if (processes[current\_process].remaining\_time > 0)

{

if (processes[current\_process].remaining\_time <=time\_quantum)

{

total\_time += processes[current\_process].remaining\_time;

processes[current\_process].remaining\_time = 0;

}

else

{

total\_time += time\_quantum;

processes[current\_process].remaining\_time -=

time\_quantum;

}

printf("Time %d: Process %d\n", total\_time,

processes[current\_process].pid);

}

if (processes[current\_process].remaining\_time == 0)

{

completed++;

processes[current\_process].waiting\_time = total\_time-processes[current\_process].burst\_time;

}

else

ready\_queue[++rear] = current\_process;

}

}

int main()

{

int n;

int time\_quantum;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter the time quantum: ");

scanf("%d", &time\_quantum);

struct Process processes[n];

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

{

printf("Enter burst time for process %d: ", i + 1);

scanf("%d", &processes[i].burst\_time);

processes[i].remaining\_time = processes[i].burst\_time;

processes[i].pid = i + 1;

}

printf("Scheduling order:\n");

RoundRobin(processes, n, time\_quantum);

double total\_waiting\_time = 0;

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

total\_waiting\_time += processes[i].waiting\_time;

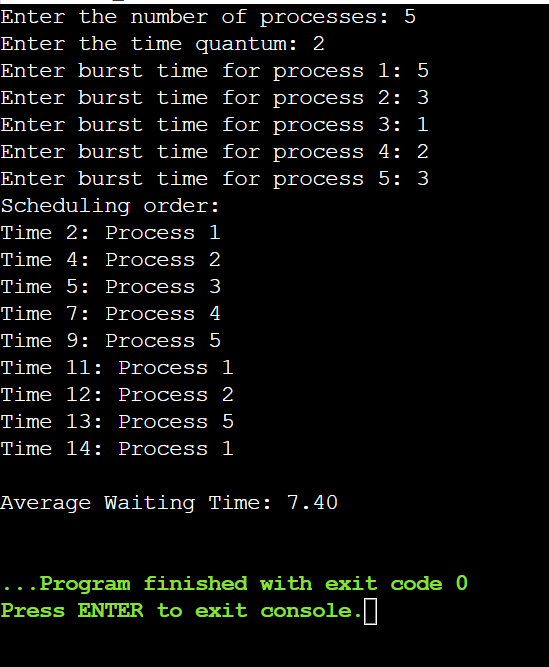
double avg\_waiting\_time = total\_waiting\_time / n;

printf("\nAverage Waiting Time: %.2lf\n", avg\_waiting\_time);

return 0;

}

**OUTPUT**

****

**WEEK 5**

**Multilevel**

#include<stdio.h>

int main()

{

int p[20],bt[20], su[20],at[20], wt[20],tat[20],i, k, n, temp;

float wtavg, tatavg;

printf("Enter the number of processes:");

scanf("%d",&n);

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

{

p[i] = i;

printf("Enter the Burst Time of Process%d:", i);

scanf("%d",&bt[i]);

printf("Enter the arrival time of process%d:",i);

scanf("%d",&at[i]);

printf("System/User Process (0/1) ? ");

scanf("%d", &su[i]);

}

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

for(k=i+1;k<n;k++)

if(su[i] > su[k])

{

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=su[i];

su[i]=su[k];

su[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

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

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tARRIVAL TIME\t\tSYSTEM/USER PROCESS\t\tBURST TIME\t\tWAITING TIME\t\tTURNAROUND TIME");

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

printf("\n%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t %d ",p[i],at[i],su[i],bt[i],wt[i],tat[i]);

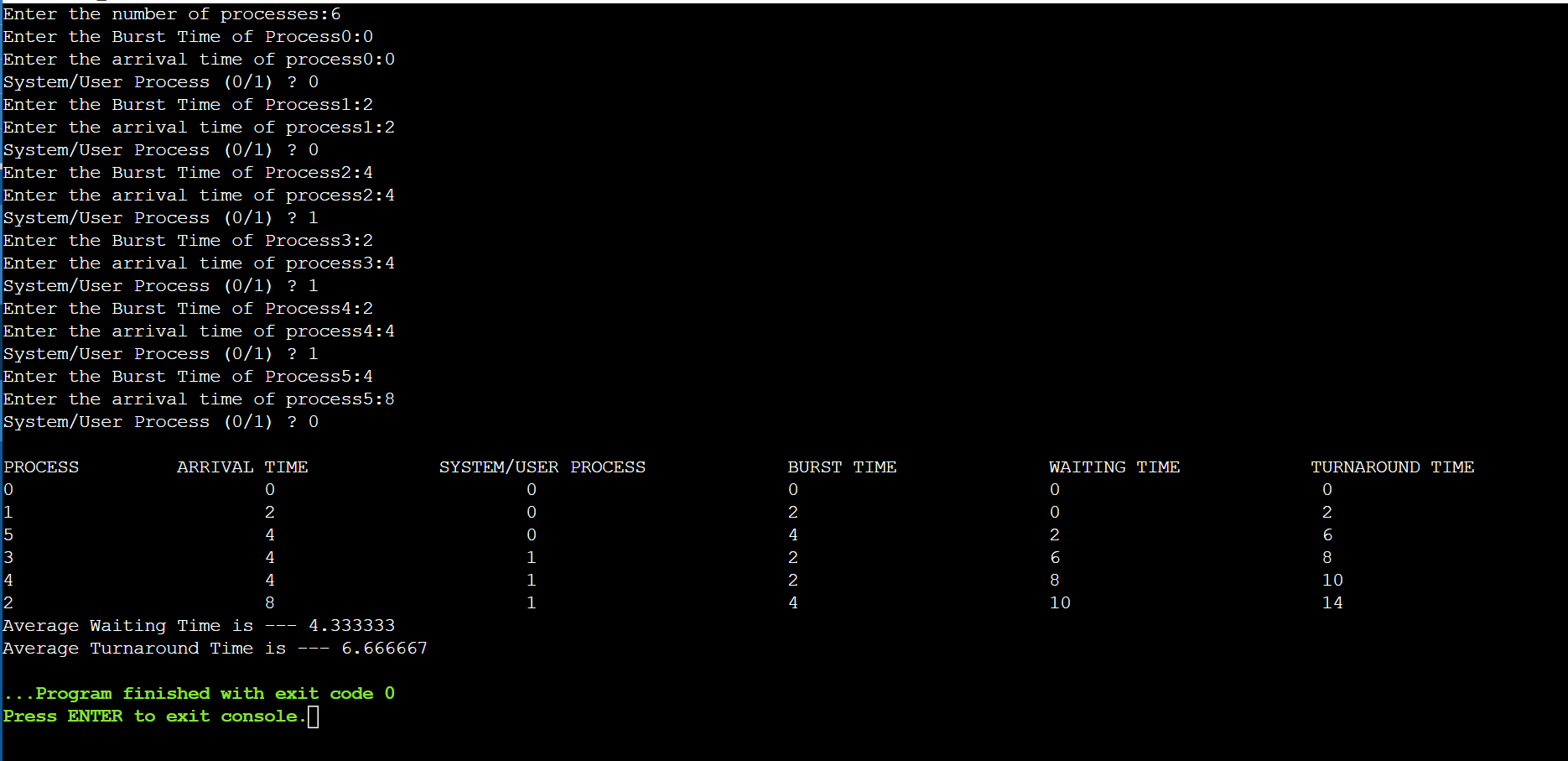
printf("\nAverage Waiting Time is --- %f",wtavg/n);

printf("\nAverage Turnaround Time is --- %f",tatavg/n);

return 0;

}

**OUTPUT**

****

**Rate Monotonic Scheduling**

#include <stdio.h>

struct process

{

int B\_time;

int period;

int pid;

int count;

};

typedef struct process procs;

int lcm(int n1, int n2, int n3)

{

int max = n1;

if (n2 > max)

max = n2;

if (n3 > max)

max = n3;

for (int i = max;; i++)

if (i % n1 == 0 && i % n2 == 0 && i % n3 == 0)

return i;

}

int main()

{

int n;

printf("enter the number of the processes\n");

scanf("%d", &n);

procs processes[n];

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

{

printf("enter the execution time for process:%d\n", i + 1);

scanf("%d", &processes[i].B\_time);

printf("enter time period for process:%d\n", i + 1);

scanf("%d", &processes[i].period);

processes[i].pid = i + 1;

}

procs temp;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (processes[j + 1].period < processes[j].period)

{

temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

printf("Processes\n");

int Ftime = lcm(processes[0].period, processes[1].period, processes[2].period);

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

{

printf("Process: %d\t execution Time:%d\t Time Period:%d\n", processes[i].pid, processes[i].B\_time, processes[i].period);

processes[i].count = Ftime / processes[i].period;

}

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

{

int j = 0;

if (i == 0)

{

int cnt = processes[i].count;

while (j < cnt)

{

printf("Process: %d at time:%d\n", processes[i].pid, (processes[i].period) \* j);

j++;

}

}

if (i == 1)

{

int cnt = processes[i].count;

while (j < cnt)

{

printf("Process: %d at time:%d\n", processes[i].pid, ((processes[i].period) \* j)+processes[i-1].B\_time);

j++;

}

}

if(i==2)

{

int cnt = processes[i].count;

while (j < cnt)

{

int time = 8;

printf("Process: %d at time:%d\n", processes[i].pid, time);

j++;

}

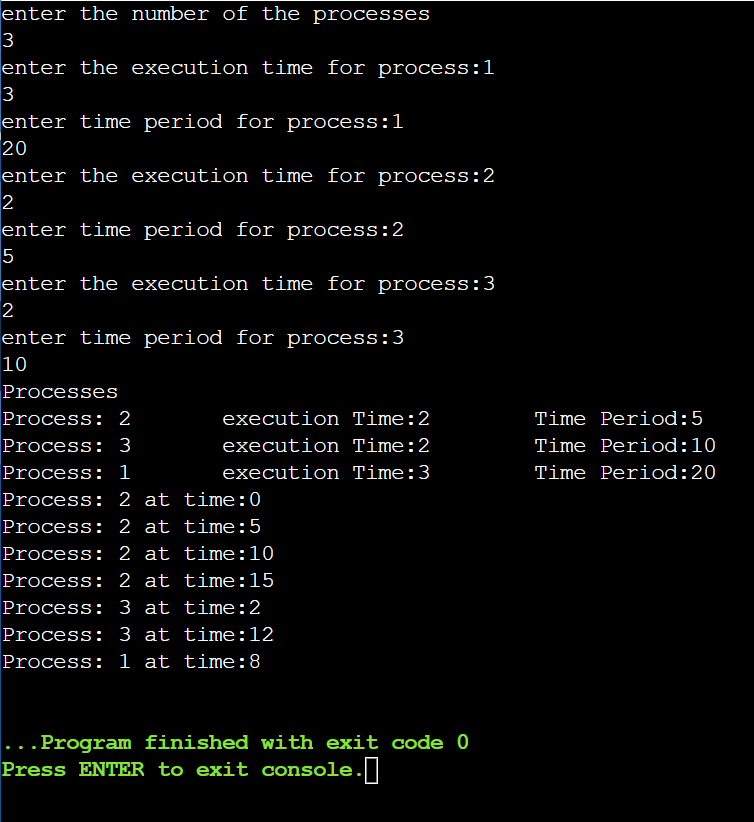
}

}

return 0;

}

**OUTPUT**

****

**WEEK 6**

**Producer Consumer**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1, full = 0, empty = 3, x = 0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1. Producer \n2.Consumer\n3.exit\n");

while (1)

{

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n)

{

case 1:

if ((mutex == 1) && (empty != 0))

producer();

else

printf("buffer is full\n");

break;

case 2:

if ((mutex == 1) && (full != 0))

consumer();

else

printf("buffer is empty\n");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return (++s);

}

void producer()

{

mutex = wait(mutex);

full = signal(full);

empty = wait(empty);

x++;

printf("Producer produces the item %d", x);

mutex = signal(mutex);

}

void consumer()

{

mutex= wait(mutex);

full = wait(full);

empty = signal(empty);

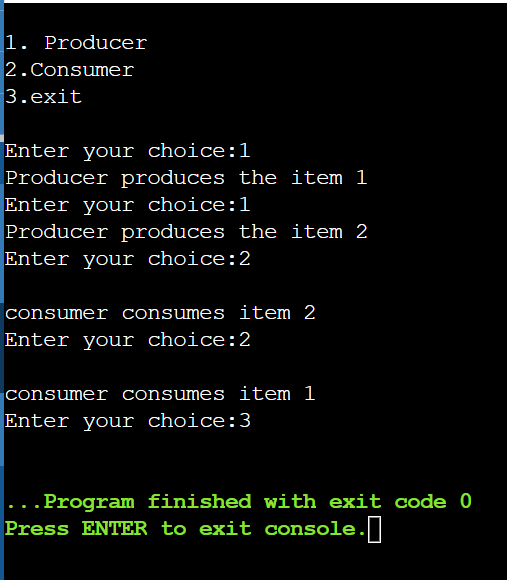
printf("\nconsumer consumes item %d", x);

x--;

mutex = signal(mutex);

}

**OUTPUT**

****

**Dinning – Philosophers Problem**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect

// during takefork

// used to wake up hungry philosophers

// during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

for (i = 0; i < N; i++)

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philosopher, &phil[i]);

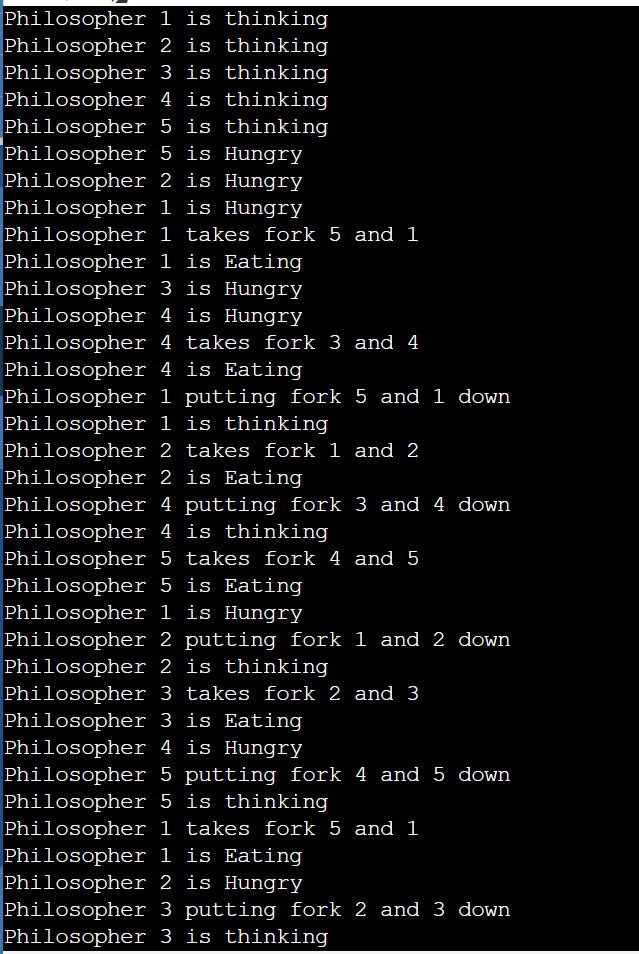
printf("Philosopher %d is thinking\n", i + 1);

}

for (i = 0; i < N; i++)

pthread\_join(thread\_id[i], NULL);

}



**WEEK 7**

**Bankers Algorithm**

#include <stdio.h>

int main()

{

int n, m, i, j, k;

printf("enter the number of processes\n");

scanf("%d", &n);

printf("enter the number of resources\n");

scanf("%d", &m);

int alloc[n][m], max[n][m];

printf("enter allocation matrix\n");

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

{

for (j = 0; j < m; j++)

{

scanf("%d", &alloc[i][j]);

}

}

printf("enter the max matrix");

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

for (j = 0; j < m; j++)

scanf("%d", &max[i][j]);

int avail[m];

printf("enter the available resources\n");

for (i = 0; i < m; i++)

scanf("%d", &avail[i]);

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++)

f[k] = 0;

int need[n][m];

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

for (j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

int y = 0;

for (k = 0; k < 5; k++)

{

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

{

if (f[i] == 0)

{

int flag = 0;

for (j = 0; j < m; j++)

{

if (need[i][j] > avail[j])

{

flag = 1;

break;

}

}

if (flag == 0)

{

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

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

{

if (f[i] == 0)

{

flag = 0;

printf("The following system is not safe");

break;

}

}

if (flag == 1)

{

printf("Following is the SAFE Sequence\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

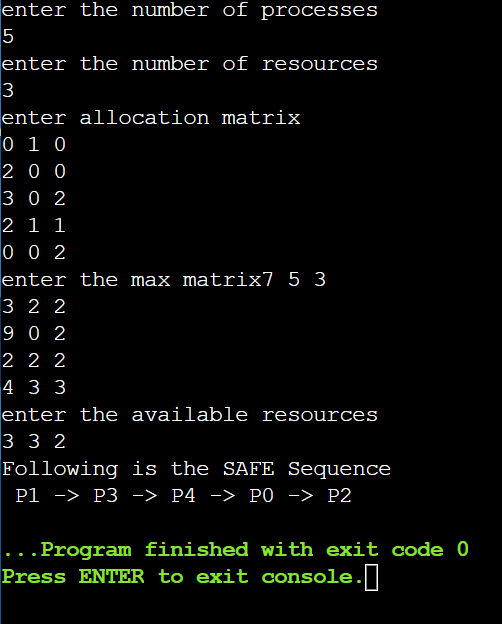
printf(" P%d", ans[n - 1]);

}

return (0);

}

**OUTPUT**

****

**WEEK 8**

**Best, Worst, First Hit**

#include <stdio.h>

int holes[10];

int holes\_free[10];

int request[10];

int completed[10];

int h,n,temp;

void best\_fit(int n, int h)

{

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

completed[i]=0;

for(int i=0;i<h;i++)

holes\_free[i]=1;

printf("Best Fit:\n");

for(int i=0;i<h-1;i++)

{

for(int j=0;j<h-i-1;j++)

{

if(holes[j+1]<holes[j])

{

temp=holes[j+1];

holes[j+1]=holes[j];

holes[j]=temp;

}

}

}

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

{

for(int j=0;j<h;j++)

{

if(request[i]<=holes[j] && holes\_free[j]==1 && completed[i]==0)

{

completed[i]=1;

holes\_free[j]=0;

printf("%dk in %dk\n",request[i],holes[j]);

}

}

}

}

void worst\_fit(int n, int h)

{

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

completed[i]=0;

for(int i=0;i<h;i++)

holes\_free[i]=1;

printf("Worst Fit:\n");

for(int i=0;i<h-1;i++)

{

for(int j=0;j<h-i-1;j++)

{

if(holes[j+1]>holes[j])

{

temp=holes[j+1];

holes[j+1]=holes[j];

holes[j]=temp;

}

}

}

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

{

for(int j=0;j<h;j++)

{

if(request[i]<=holes[j] && holes\_free[j]==1 && completed[i]==0)

{

completed[i]=1;

holes\_free[j]=0;

printf("%dk in %dk\n",request[i],holes[j]);

}

}

}

}

void first\_fit(int n, int h)

{

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

completed[i]=0;

for(int i=0;i<h;i++)

holes\_free[i]=1;

printf("First Fit:\n");

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

{

for(int j=0;j<h;j++)

{

if(request[i]<=holes[j] && holes\_free[j]==1 && completed[i]==0)

{

completed[i]=1;

holes\_free[j]=0;

printf("%dk in %dk\n",request[i],holes[j]);

}

}

}

}

int main()

{

printf("enter the number of holes:\t");

scanf("%d",&h);

printf("Enter the holes sizes:\n");

for(int i=0;i<h;i++)

scanf("%d",&holes[i]);

printf("enter the number of requests:\t");

scanf("%d",&n);

printf("Enter the request segments:\n");

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

scanf("%d",&request[i]);

best\_fit(n,h);

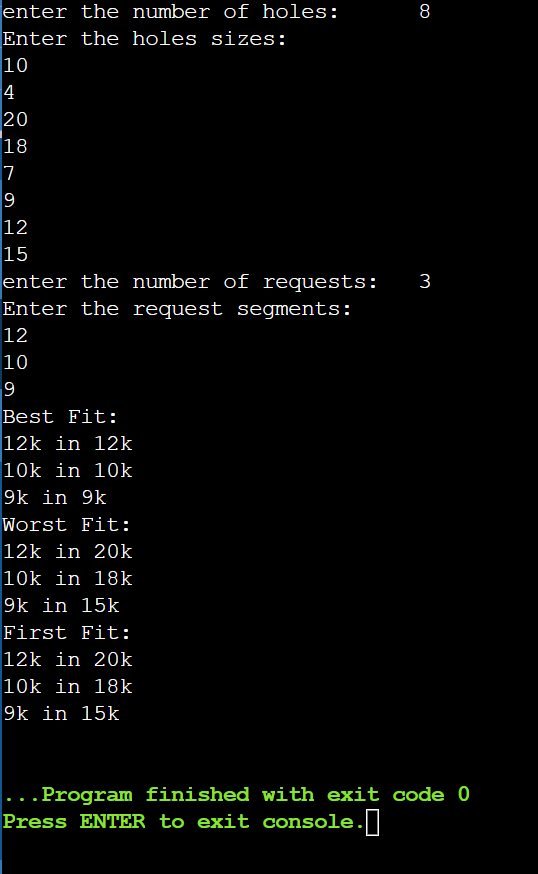
worst\_fit(n,h);

first\_fit(n,h);

return 0;

}

**OUTPUT**

****

**WEEK 9**

**Disk Scheduling**

#include <stdio.h>

#include <stdlib.h>

int queue[10];

int n;

int head;

void sort(int arr[])

{

int t;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(arr[j+1]<arr[j])

{

t=arr[j+1];

arr[j+1]=arr[j];

arr[j]=t;

}

}

}

return ;

}

int fcfs()

{

int move=0;

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

{

move+=abs(head-queue[i]);

head=queue[i];

}

return move;

}

int sstf()

{

int move=0;

int vis[n];

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

{

vis[i]=0;

}

int count=0;

int u;

while(count!=n)

{

int min=999;

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

{

if(abs(head-queue[i])<min && vis[i]==0)

{

u=i;

min = abs(head-queue[i]);

}

}

vis[u]=1;

move+=abs(head-queue[u]);

head=queue[u];

count++;

}

return move;

}

int scan()

{

int move=0;

int l,u,d;

printf("enter the lower and upper limit:\n");

scanf("%d%d",&l,&u);

printf("enter the direction:(1->UP and 0->down):\n");

scanf("%d",&d);

sort(queue);

if(d==0)

{

move=(head-l)+(queue[n-1]-l);

}

else

{

move=(u-head)+(u-queue[0]);

}

return move;

}

int look()

{

int move=0;

int d;

printf("enter the direction:(1->UP and 0->down):\n");

scanf("%d",&d);

sort(queue);

if(d==1)

{

move=(queue[n-1]-head)+(queue[n-1]-queue[0]);

}

else

{

move=(head-queue[0])+(queue[n-1]-queue[0]);

}

return move;

}

int Clook()

{

int move=0,d,u,count=0;

sort(queue);

printf("enter the direction:(1->UP and 0->down):\n");

scanf("%d",&d);

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

{

if(queue[i]>head){

u=i;

break;

}

}

if(d==1)

{

while(count!=n)

{

move+=abs(queue[u]-head);

head=queue[u];

u=(u+1)%n;

count++;

}

}

else

{

u--;

while(count!=n)

{

if(u<0)

{

u=n-1;

}

move+=abs(queue[u]-head);

head=queue[u];

u=(u-1)%n;

count++;

}

}

return move;

}

int Cscan()

{

int move=0;

int l,u;

printf("enter the lower and upper limit:\n");

scanf("%d%d",&l,&u);

move=Clook();

move+=2\*((u-queue[n-1])+(queue[0]-l));

return move;

}

int main()

{

int ch;

printf("enter the number of the containers:\n");

scanf("%d",&n);

printf("enter the queue of addresses:\n");

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

{

scanf("%d",&queue[i]);

}

printf("enter the head location:\n");

scanf("%d",&head);

do

{

printf("1.FCFS\n2.SSTF\n3.SCAN\n4.LOOK\n5.C-LOOK\n6.C-SCAN\n7.EXIT\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Disk Movement using fcfs Algorithm= %d",fcfs());

break;

case 2:

printf("Disk Movement using sstf Algorithm= %d",sstf());

break;

case 3:

printf("Disk Movement using scan Algorithm= %d",scan());

break;

case 4:

printf("Disk Movement using look Algorithm= %d",look());

break;

case 5:

printf("Disk Movement using C-look Algorithm= %d",Clook());

break;

case 6:

printf("Disk Movement using C-Scan Algorithm= %d",Cscan());

break;

case 7:

exit(0);

default:

printf("enter Valid choice!!!\n");

break;

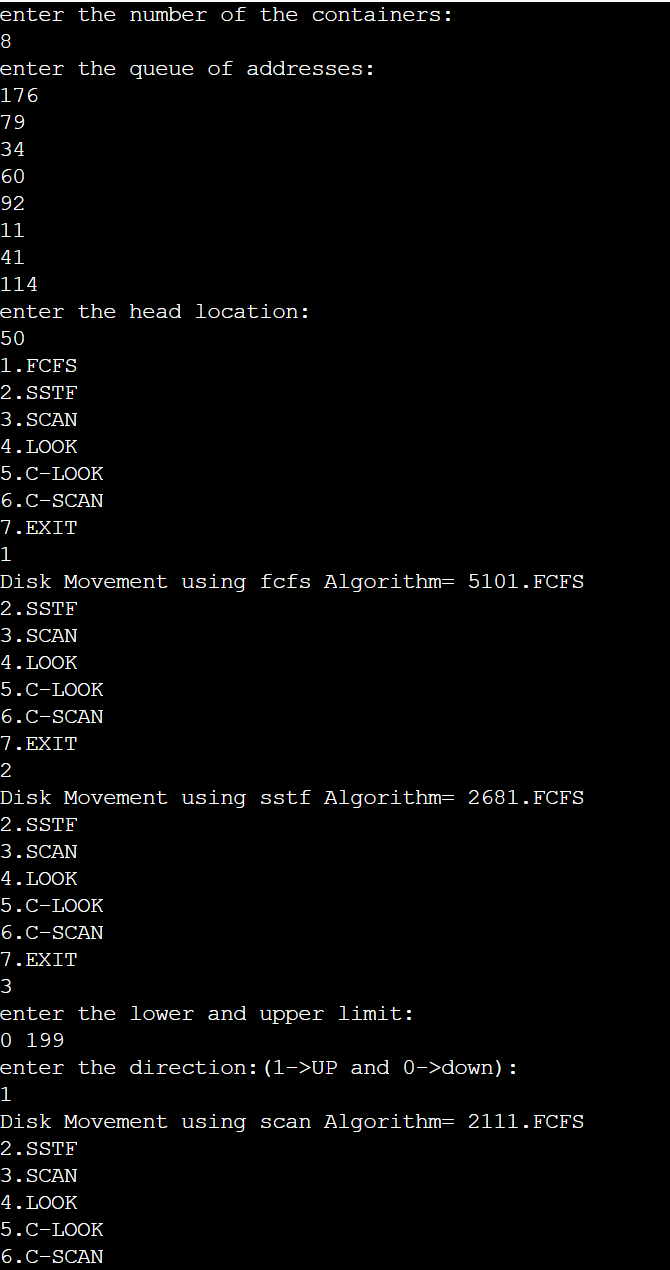
}

}while(ch!=7);

return 0;

}

**OUTPUT**

****

**Paging**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_FRAMES 4

void printFrames(int frames[], int n)

{

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

{

if (frames[i] == -1)

printf("- ");

else

printf("%d ", frames[i]);

}

printf("\n");

}

int findLRUIndex(int counters[], int n)

{

int minIndex = 0;

for (int i = 1; i < n; i++)

{

if (counters[i] < counters[minIndex])

minIndex = i;

}

return minIndex;

}

int findOptimalIndex(int pages[], int frames[], int n, int start)

{

int index = -1;

int farthest = start;

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

{

int j;

for (j = start; j < n; j++)

{

if (frames[i] == pages[j])

{

if (j > farthest)

{

farthest = j;

index = i;

}

break;

}

}

if (j == n)

return i;

}

return (index == -1) ? 0 : index;

}

void fifo(int pages[], int n)

{

int frames[MAX\_FRAMES];

int frameIndex = 0;

for (int i = 0; i < MAX\_FRAMES; i++)

frames[i] = -1;

printf("FIFO Page Replacement Algorithm:\n");

int pageFaults = 0;

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

{

int page = pages[i];

bool pageFound = false;

for (int j = 0; j < MAX\_FRAMES; j++)

{

if (frames[j] == page)

{

pageFound = true;

break;

}

}

if (!pageFound)

{

frames[frameIndex] = page;

frameIndex = (frameIndex + 1) % MAX\_FRAMES;

pageFaults++;

}

printFrames(frames, MAX\_FRAMES);

}

printf("Total Page Faults: %d\n\n", pageFaults);

}

void lru(int pages[], int n)

{

int frames[MAX\_FRAMES];

int counters[MAX\_FRAMES] = {0};

for (int i = 0; i < MAX\_FRAMES; i++)

frames[i] = -1;

printf("LRU Page Replacement Algorithm:\n");

int pageFaults = 0;

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

{

int page = pages[i];

bool pageFound = false;

for (int j = 0; j < MAX\_FRAMES; j++)

{

if (frames[j] == page)

{

pageFound = true;

counters[j] = i;

break;

}

}

if (!pageFound)

{

int lruIndex = findLRUIndex(counters, MAX\_FRAMES);

frames[lruIndex] = page;

counters[lruIndex] = i;

pageFaults++;

}

printFrames(frames, MAX\_FRAMES);

}

printf("Total Page Faults: %d\n\n", pageFaults);

}

void optimal(int pages[], int n)

{

int frames[MAX\_FRAMES];

for (int i = 0; i < MAX\_FRAMES; i++)

frames[i] = -1;

printf("Optimal Page Replacement Algorithm:\n");

int pageFaults = 0;

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

{

int page = pages[i];

bool pageFound = false;

for (int j = 0; j < MAX\_FRAMES; j++)

{

if (frames[j] == page)

{

pageFound = true;

break;

}

}

if (!pageFound)

{

int optimalIndex = findOptimalIndex(pages, frames, n, i + 1);

frames[optimalIndex] = page;

pageFaults++;

}

printFrames(frames, MAX\_FRAMES);

}

printf("Total Page Faults: %d\n\n", pageFaults);

}

int main()

{

int n;

printf("enter the number of pages:\n");

scanf("%d",&n);

int pages[n] ;

printf("enter the page indexes:\n");

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

scanf("%d",&pages[i]);

int choice;

do {

printf("Page Replacement Algorithms:\n");

printf("1. FIFO\n");

printf("2. LRU\n");

printf("3. Optimal\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

fifo(pages, n);

break;

case 2:

lru(pages, n);

break;

case 3:

optimal(pages, n);

break;

case 4:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please select a valid option.\n");

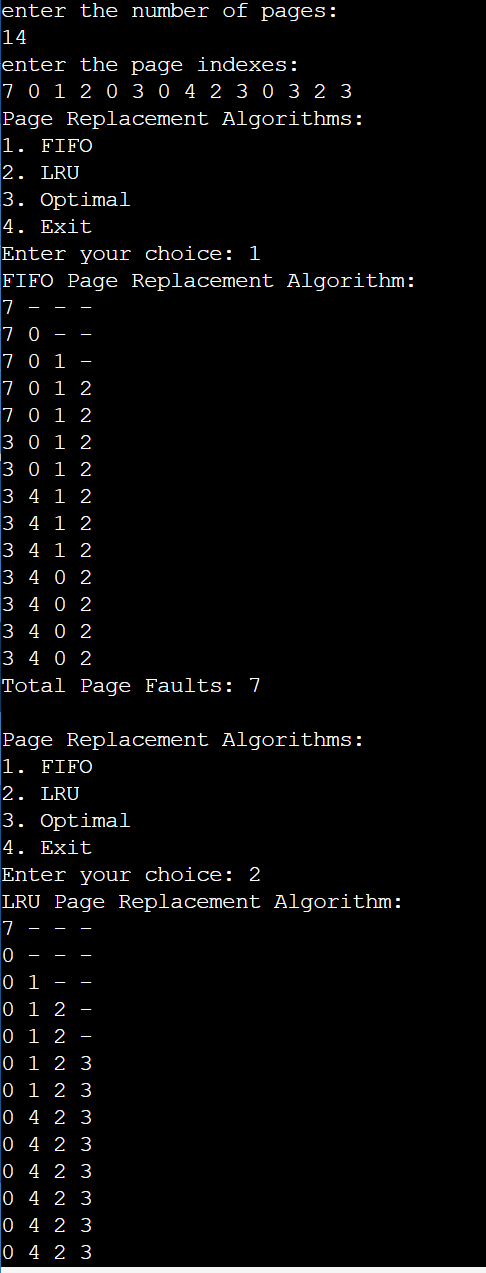
}

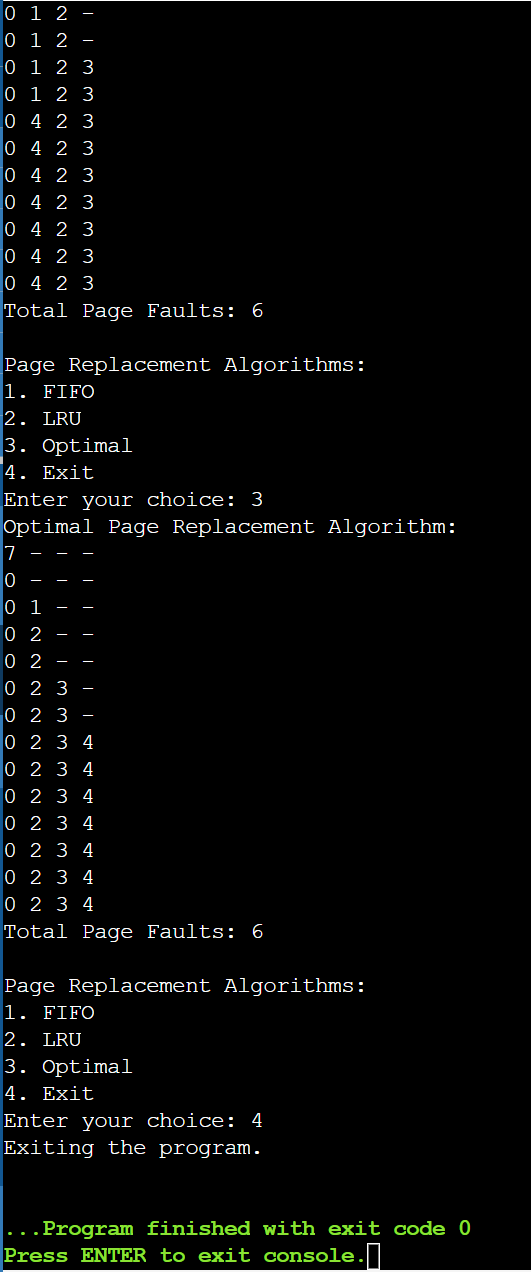
} while (choice != 4);

return 0;

}

**OUTPUT**

****

****