**INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN**



**Operating Systems Practical File BIT-202**

**Submitted By: Submitted To:**

Name: Meghavi Tomar Dr. Arunima Jaiswal

Enrollment No.: 18601012020 Assistant Professor

Batch: CSE2, B2 CSE Department

B.Tech, 4th Semester IGDTUW

Index

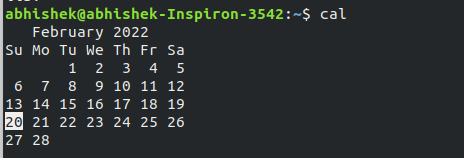
|  |  |  |
| --- | --- | --- |
| **S.No.** | **Experiment Name** | **Date** |
| 1. | Execute various LINUX commands:   1. clear, cal, who, date, pwd 2. cd, mkdir, rmdir, wc, cat | 24-01-2022 |
| 2. | Execute various LINUX commands:   1. ps, alias, ls, grep 2. cmp, comm, diff 3. chmod, chown, chgrp | 31-01-2022 |
| 3. | To write shell script to find average of three numbers. | 14-02-2022 |
| 4. | Write a program in C to implement First Come First Serve algorithm. | 14-02-2022 |
| 5. | Write a program in C to implement Shortest Job First Algorithm. | 21-02-2022 |
| 6. | Write a program in C to implement Shortest Run Time Next. | 21-02-2022 |
| 7. | Write a program in C to implement Priority Based Preemption algorithm. | 28-02-2022 |
| 8. | Write a program in C to implement Priority Based Non - Preemption algorithm. | 28-02-2022 |
| 9. | Write a program in C to implement Round Robin algorithm. | 07-03-2022 |
| 10. | Write a program in C to implement Banker’s algorithm. | 07-03-2022 |
| 11. | Write a program in C to implement Optimal Page Replacement algorithm. | 11-04-2022 |
| 12. | Write a program in C to implement FIFO algorithm. | 11-04-2022 |
| 13. | Write a program in C to implement LRU algorithm. | 18-04-2022 |
| 14. | Write a program in C to implement SCAN Disk Scheduling algorithm. | 18-04-2022 |
| 15. | Write a program in C to implement Shortest Seek Time First Disk Scheduling algorithm. | 25-04-2022 |
| 16. | Write a program in C to implement C-SCAN Disk Scheduling algorithm. | 25-04-2022 |

# Experiment 1

1. Execute various LINUX commands:

1.1

* 1. cal



* 1. who



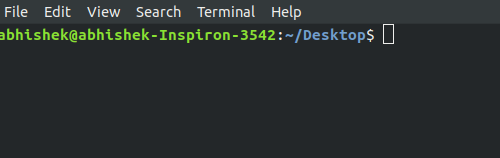
* 1. date



* 1. pwd

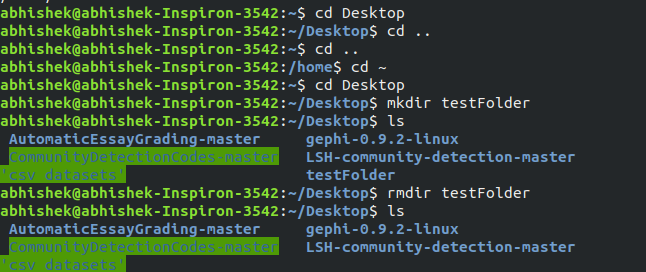


* 1. clear



1.2

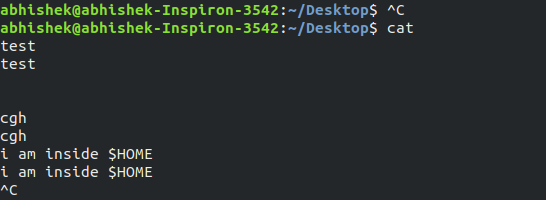
(i) cd, mkdir , rmdir



(ii ) wc

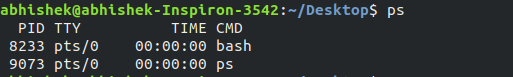


(iii ) cat

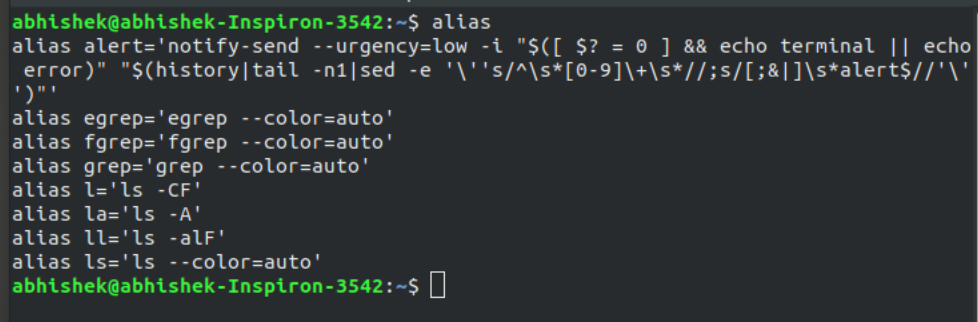


# Experiment 2

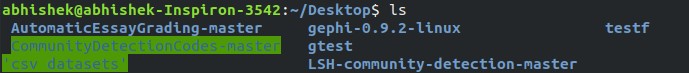
1. Execute various linux commands; 2.1
2. ps



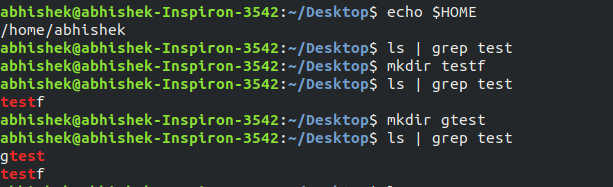
1. alias



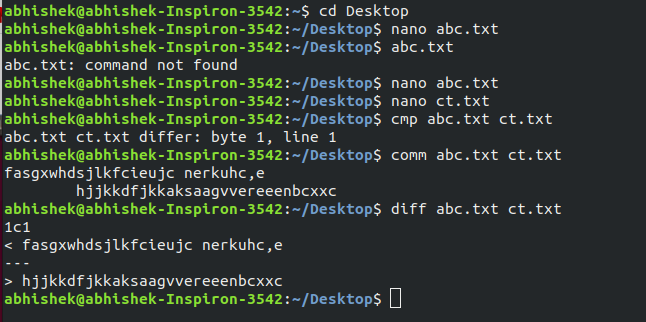
1. ls



1. grep

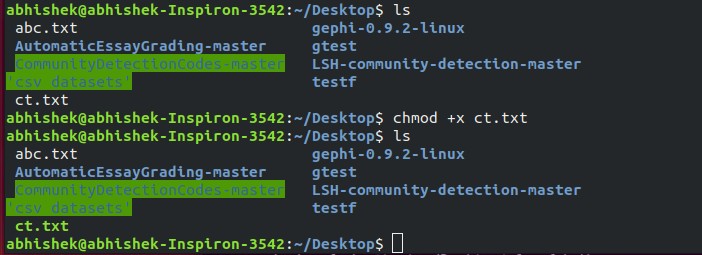


2.2 cmp, comm, diff

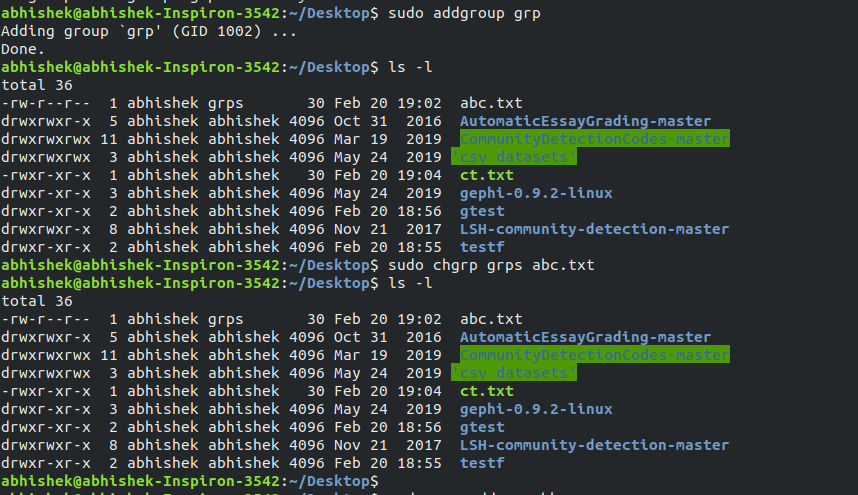


2.3

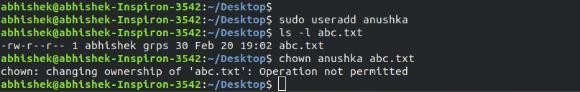
1. chmod



1. chown

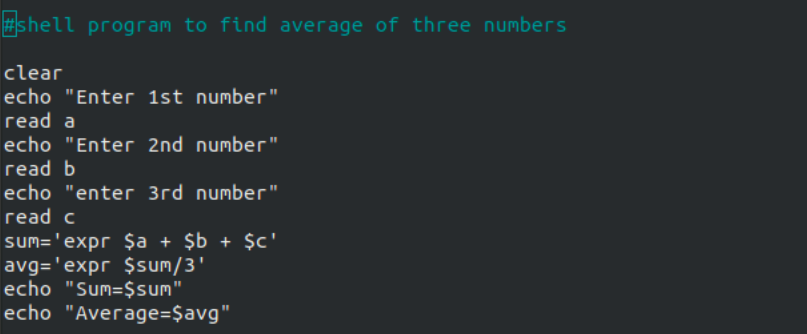


1. chgrp



# Experiment 3

Q. To write shell script to find average of three numbers.





# Experiment 4

Q. Write a program in C to implement First Come First Serve algorithm.

Code:

#include <stdio.h> using namespace std;

int waitingtime(int proc[], int n,

int burst\_time[], int wait\_time[]) { wait\_time[0] = 0;

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

wait\_time[i] = burst\_time[i-1] + wait\_time[i-1] ; return 0;

}

int turnaroundtime( int proc[], int n,

int burst\_time[], int wait\_time[], int tat[]) { int i;

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

tat[i] = burst\_time[i] + wait\_time[i]; return 0;

}

int avgtime( int proc[], int n, int burst\_time[]) {

int wait\_time[n], tat[n], total\_wt = 0, total\_tat = 0; int i;

waitingtime(proc, n, burst\_time, wait\_time); turnaroundtime(proc, n, burst\_time, wait\_time, tat); printf("Processes Burst Waiting Turn around \n");

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

total\_wt = total\_wt + wait\_time[i]; total\_tat = total\_tat + tat[i];

printf(" %d\t %d\t\t %d \t%d\n", i+1, burst\_time[i], wait\_time[i], tat[i]);

}

printf("Average waiting time = %f\n", (float)total\_wt / (float)n); printf("Average turn around time = %f\n", (float)total\_tat / (float)n); return 0;

}

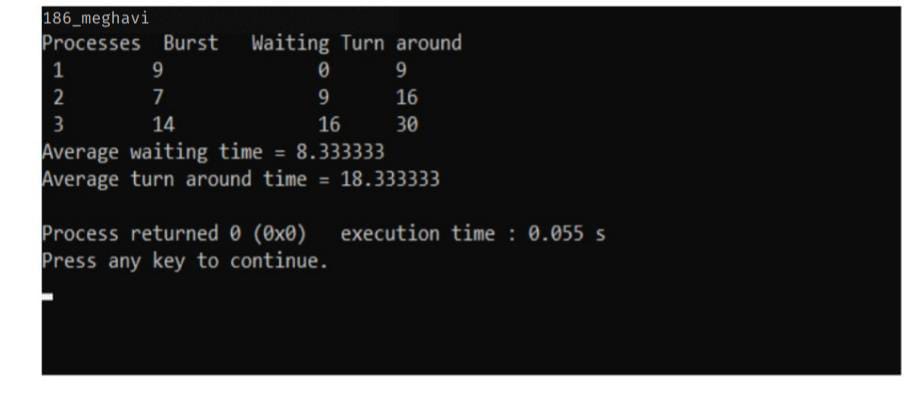
int main() {

printf (“186\_meghavi\n” ); int proc[] = { 1, 2, 3};

int n = sizeof proc / sizeof proc[0]; int burst\_time[] = {9, 7, 14}; avgtime(proc, n, burst\_time); return 0;

}

Output:



# Experiment 5

Q. Write a program in C to implement Shortest Job First Algorithm.

Code:

#include<stdio.h> using namespace std;

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp; float avg\_wt,avg\_tat;

printf (“186\_meghavi\n”) ; printf("Enter number of process:"); scanf("%d",&n);

printf("\nEnter Burst Time:\n"); for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&bt[i]); p[i]=i+1;

}

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

{

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

{

if(bt[j]<bt[pos]) pos=j;

}

temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp;

temp=p[i]; p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n; total=0;

printf("\nProcess\t Burst Time \t Waiting Time\t Turnaround Time"); for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; total+=tat[i];

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

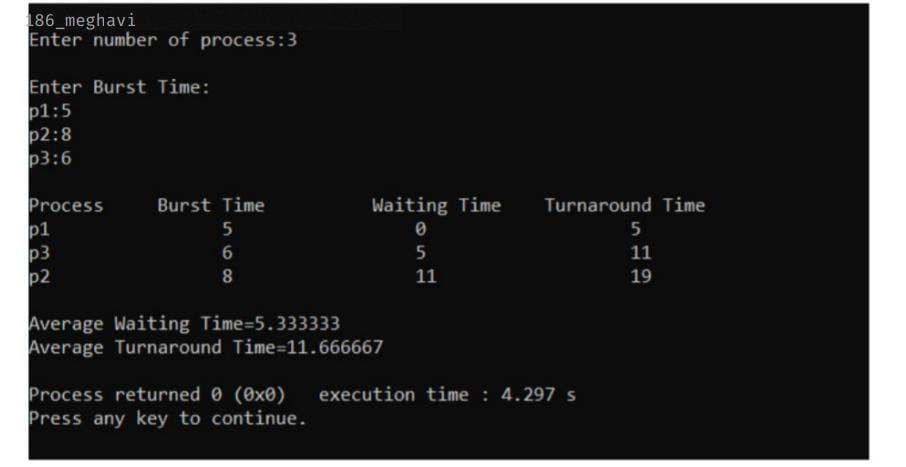
}

avg\_tat=(float)total/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt); printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

Output:



# Experiment 6

Q. Write a program to implement shortest run time next algorithm.

Code:

#include<stdio.h> int main()

{

int at[10],bt[10],rt[10],endTime,i,smallest;

int remain=0,n,time,sum\_wait=0,sum\_turnaround=0; printf("186\_meghavi\n ");

printf("Enter no of Processes : "); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("Enter arrival time for Process P%d : ",i+1); scanf("%d",&at[i]);

printf("Enter burst time for Process P%d : ",i+1); scanf("%d",&bt[i]);

rt[i]=bt[i];

}

printf("\n\nProcess\t|Turnaround Time| Waiting Time\n\n"); rt[9]=9999;

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

{

smallest=9; for(i=0;i<n;i++)

{

if(at[i]<=time && rt[i]<rt[smallest] && rt[i]>0)

{

smallest=i;

}

}

rt[smallest]--; if(rt[smallest]==0)

{

remain++; endTime=time+1;

printf("\nP[%d]\t|\t%d\t|\t%d",smallest+1,endTime-at[smallest],endTime-bt[smallest]- at[smallest]);

sum\_wait+=endTime-bt[smallest]-at[smallest]; sum\_turnaround+=endTime-at[smallest];

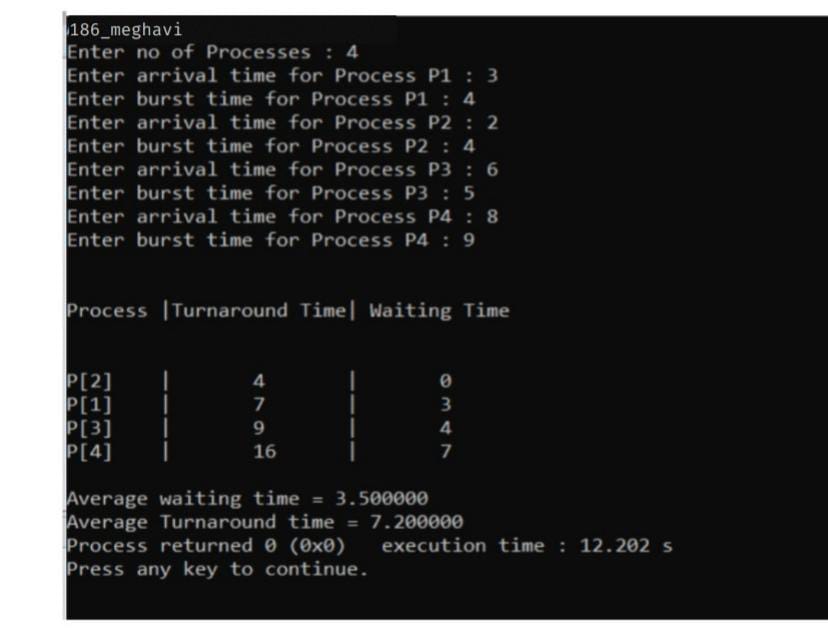
}

}

printf("\n\nAverage waiting time = %f\n",sum\_wait\*1.0/n); printf("Average Turnaround time = %f",sum\_turnaround\*1.0/5); return 0;

}

Output:



# Experiment 7

Q. Write a program in C to implement Priority Based Preemption algorithm.

Code:

#include<stdio.h> int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat; printf("186\_meghavi\n ");

printf("Enter Total Number of Process:"); scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n"); for(i=0;i<n;i++)

{

printf("\nP[%d]\n",i+1); printf("Burst Time:"); scanf("%d",&bt[i]); printf("Priority:"); scanf("%d",&pr[i]); p[i]=i+1;

}

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

{

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

{

if(pr[j]<pr[pos]) pos=j;

}

temp=pr[i]; pr[i]=pr[pos]; pr[pos]=temp; temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp; temp=p[i]; p[i]=p[pos]; p[pos]=temp;

} wt[0]=0;

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

{

wt[i]=0; for(j=0;j<i;j++) wt[i]+=bt[j]; total+=wt[i];

}

avg\_wt=total/n; total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; total+=tat[i];

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

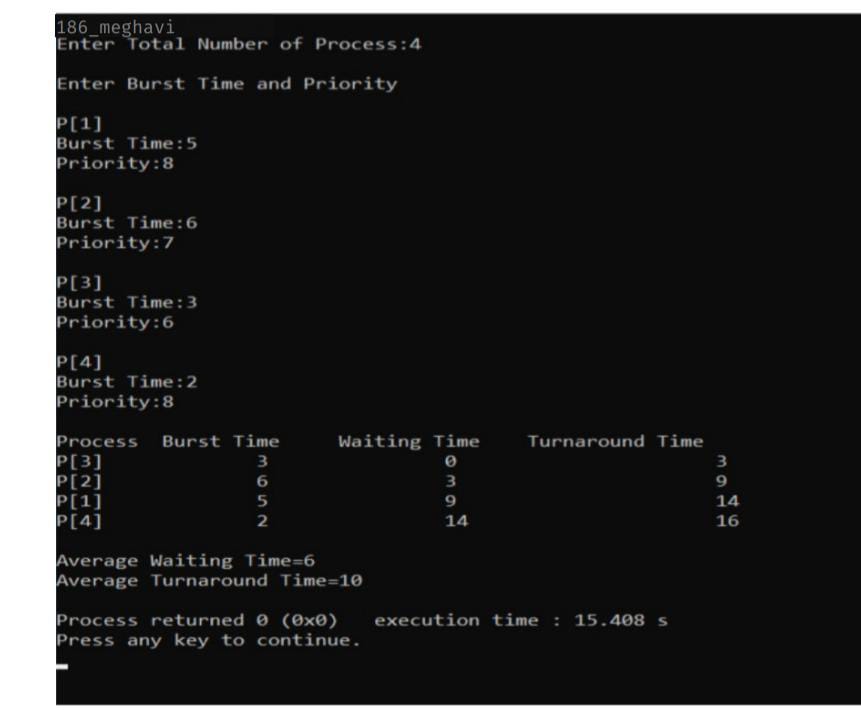
}

avg\_tat=total/n;

printf("\n\nAverage Waiting Time=%d",avg\_wt); printf("\nAverage Turnaround Time=%d\n",avg\_tat); return 0;

}

Output:



# Experiment 8

Q. Write a program in C to implement Priority Based Non - Preemption algorithm.

Code:

#include <stdio.h> int main()

{

int pn = 0; int CPU = 0;

int allTime = 0; printf("186\_meghavi\n "); printf("Enrer Processes Count: "); scanf("%d",&pn);

int AT[pn]; int ATt[pn]; int NoP = pn; int PT[pn]; int PP[pn];

int waittingTime[pn]; int turnaroundTime[pn]; for(int i=0 ;i<pn ;i++)

{

printf("\nProcessing time for P%d: ",i+1); scanf("%d",&PT[i]);

printf("Piriorty for P%d: ",i+1); scanf("%d",&PP[i]);

printf("Arrival Time for P%d: ",i+1); scanf("%d",&AT[i]);

ATt[i] = AT[i];

}

int LAT = 0;

for(int i = 0; i < pn; i++) if(AT[i] > LAT)

LAT = AT[i];

int ATv = AT[0]; int ATi = 0;

int P1 = PP[0]; int P2 = PP[0];

while(NoP > 0 && CPU <= 1000){ for(int i = 0; i < pn; i++){

if(ATt[i] < ATv){ ATi = i;

ATv = ATt[i];

P1 = PP[i];

P2 = PP[i];

}

else if(ATt[i] == ATv || ATt[i] <= CPU){

if(PP[i] != (pn+1)) P2 = PP[i];

if(P2 < P1){

ATi = i;

ATv = ATt[i];

P1 = PP[i];

P2 = PP[i];

}

}

}

if(CPU < ATv){ CPU = CPU+1;

continue;

}else{

waittingTime[ATi] = CPU - ATt[ATi]; CPU = CPU + PT[ATi];

turnaroundTime[ATi] = CPU - ATt[ATi]; ATt[ATi] = LAT +10;

ATv = LAT +10; ATi = 0;

PP[ATi] = pn + 1; P1 = PP[0];

P2 = PP[0];

printf("Iam in");

NoP = NoP - 1;

}

}

printf("\nPN\tPT\tPP\tWT\tTT\n\n"); for(int i = 0; i < pn; i++){

printf("P%d\t%d\t%d\t%d\t%d\n",i+1,PT[i],PP[i],waittingTime[i],turnaroundTime[i]);

}

int AvgWT = 0; int AVGTaT = 0;

for(int i = 0; i < pn; i++){

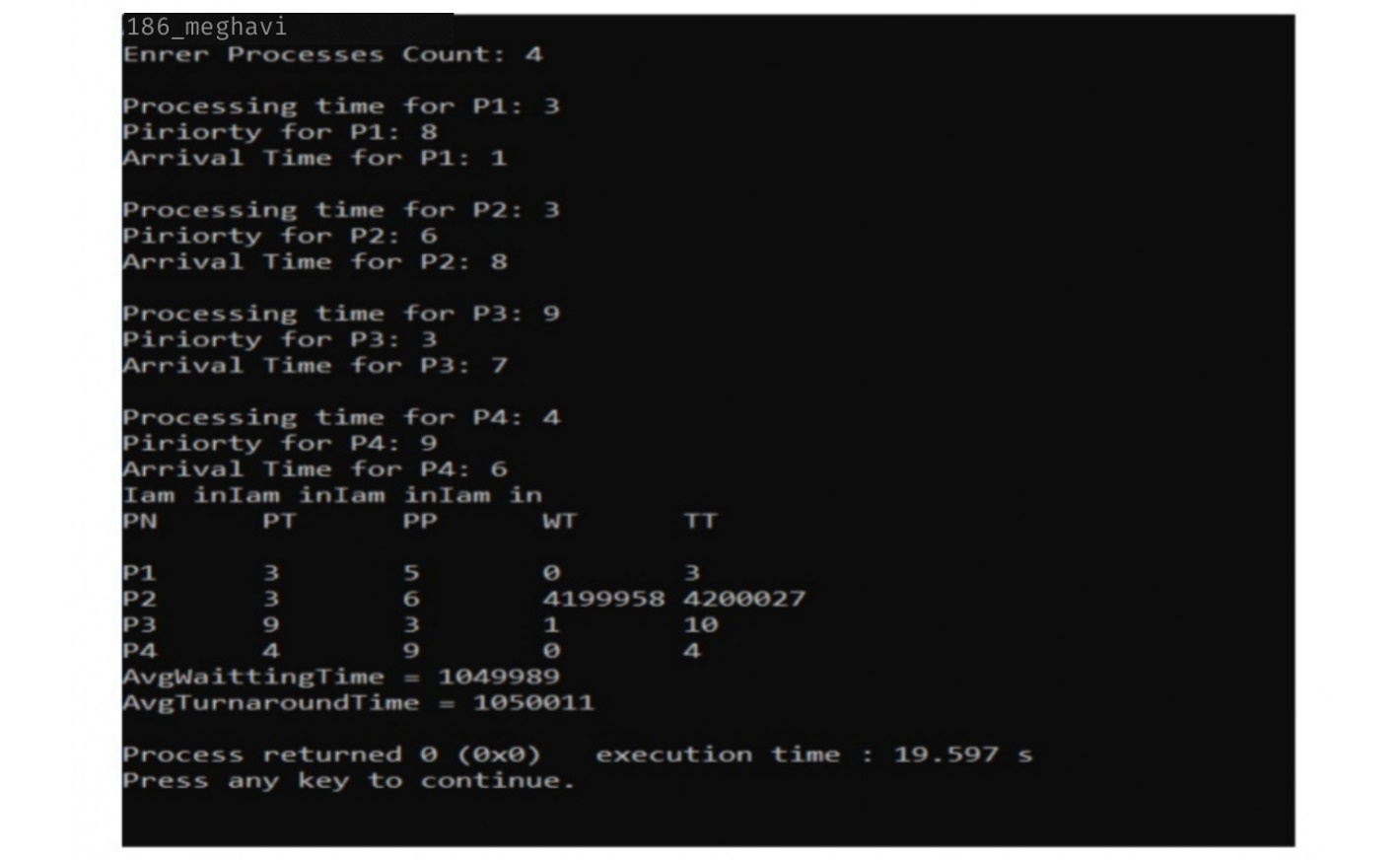
AvgWT = waittingTime[i] + AvgWT; AVGTaT = turnaroundTime[i] + AVGTaT;

}

printf("AvgWaittingTime = %d\nAvgTurnaroundTime = %d\n",AvgWT/pn,AVGTaT/pn); return 0;

}

Output :



# Experiment 9

Q. Write a program in C to implement Round Robin algorithm.

Code:

#include<stdio.h> int main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10]; float avg\_wt, avg\_tat;

printf("186\_meghavi\n ");

printf("Total number of process in the system: "); scanf("%d", &NOP);

y = NOP;

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

{

printf("\nEnter the Arrival and Burst time of the Process[%d]", i+1); printf("\nArrival time is: \t");

scanf("%d", &at[i]); printf("\nBurst time is: \t"); scanf("%d", &bt[i]); temp[i] = bt[i];

}

printf("\nEnter the Time Quantum for the process: \t"); scanf("%d", &quant);

printf("\nProcess No \t\t Burst Time \t\t TAT \t\t Waiting Time "); for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0)

{

sum = sum + temp[i]; temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant; sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--;

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]); wt = wt+sum-at[i]-bt[i];

tat = tat+sum-at[i]; count =0;

}

if(i==NOP-1)

{ i=0;

}

else if(at[i+1]<=sum)

{ i++;

}

else

{ i=0;

}

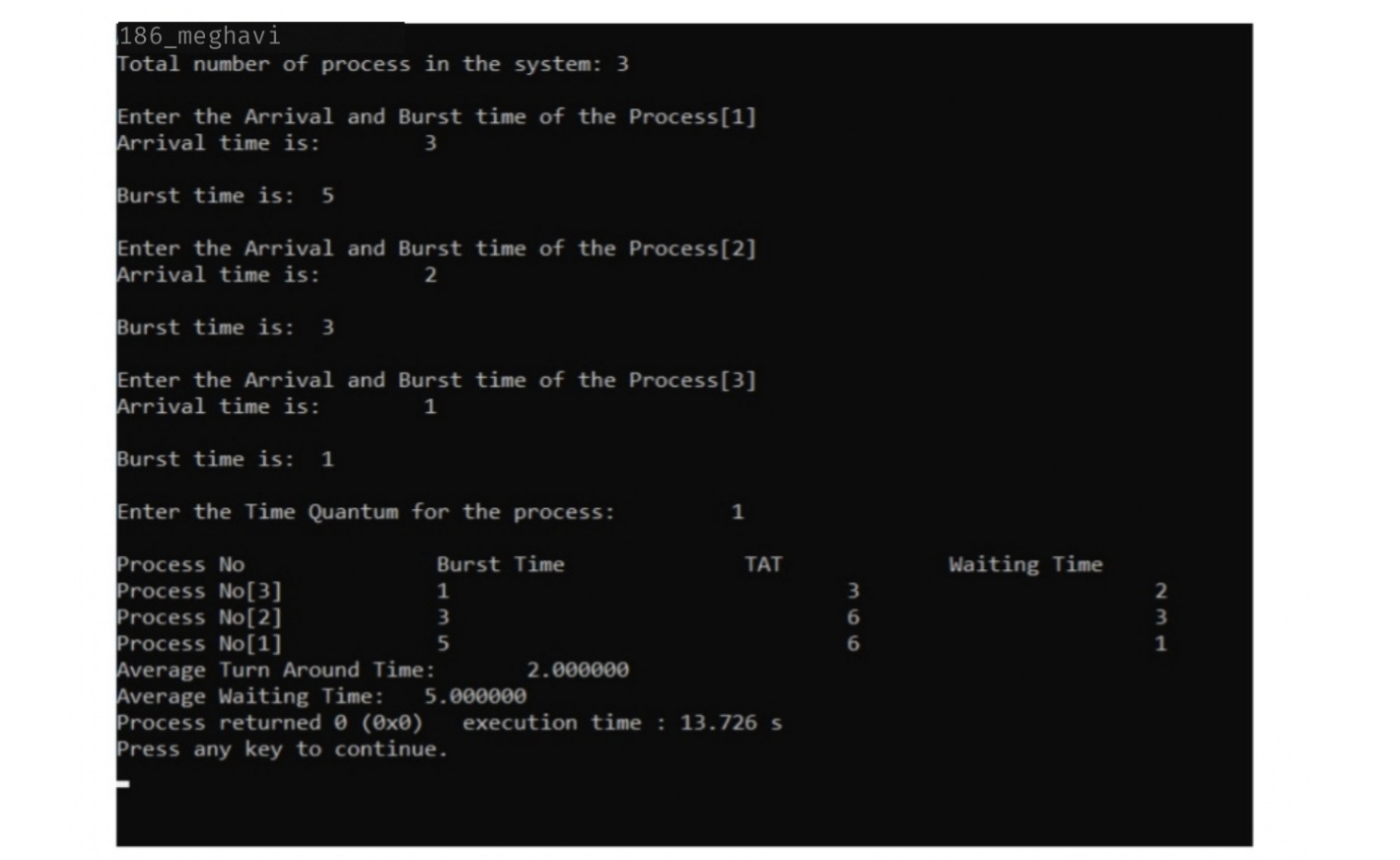
}

avg\_wt = wt \* 1.0/NOP; avg\_tat = tat \* 1.0/NOP;

printf("\nAverage Turn Around Time: \t%f", avg\_wt); printf("\nAverage Waiting Time: \t%f", avg\_tat); return 0;

}

Output:



# Experiment 10

Q. Write a program in C to implement Banker’s algorithm.

Code:

#include <stdio.h>

int curr[5][5], maxclaim[5][5], avl[5]; int alloc[5] = {0, 0, 0, 0, 0};

int maxres[5], running[5], safe=0; int count = 0, i, j, exec, r, p, k = 1; int main()

{

printf("186\_meghavi\n "); printf("\nEnter the number of processes: "); scanf("%d", &p);

for (i = 0; i < p; i++) { running[i] = 1; count++;

}

printf("\nEnter the number of resources: "); scanf("%d", &r);

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

printf("\nEnter the resource for instance%d: ", k++); scanf("%d", &maxres[i]);

}

printf("\nEnter maximum resource table:\n"); for (i = 0; i < p; i++) {

for(j = 0; j < r; j++) { scanf("%d", &maxclaim[i][j]);

}

}

printf("\nEnter allocated resource table:\n"); for (i = 0; i < p; i++) {

for(j = 0; j < r; j++) { scanf("%d", &curr[i][j]);

}

}

printf("\nThe resource of instances: "); for (i = 0; i < r; i++) {

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

}

printf("\nThe allocated resource table:"); for (i = 0; i < p; i++) {

for (j = 0; j < r; j++) { printf("\t%d", curr[i][j]);

}

printf("\n");

}

printf("\nThe maximum resource table:");

for (i = 0; i < p; i++) { for (j = 0; j < r; j++) {

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

}

printf("\n");

}

for (i = 0; i < p; i++) { for (j = 0; j < r; j++) { alloc[j] += curr[i][j];

}

}

printf("\nAllocated resources:\n"); for (i = 0; i < r; i++) { printf("\t%d", alloc[i]);

}

for (i = 0; i < r; i++) { avl[i] = maxres[i] - alloc[i];

}

printf("\nAvailable resources:\n"); for (i = 0; i < r; i++) { printf("/t%d", avl[i]);

}

printf("\n");

while (count != 0) { safe = 0;

for (i = 0; i < p; i++) { if (running[i]) {

exec = 1;

for (j = 0; j < r; j++) {

if (maxclaim[i][j] - curr[i][j] > avl[j]) { exec = 0;

break;

}

}

if (exec) {

printf("\nProcess%d is executing\n", i + 1); running[i] = 0;

count--; safe = 1;

for (j = 0; j < r; j++) { avl[j] += curr[i][j];

}

break;

}

}

}

if (!safe) {

printf("\nThe processes are in unsafe state."); break;

} else {

printf("\nThe process is in safe state"); printf("\nSafe sequence is:");

for (i = 0; i < r; i++) { printf("t%d", avl[i]);

}

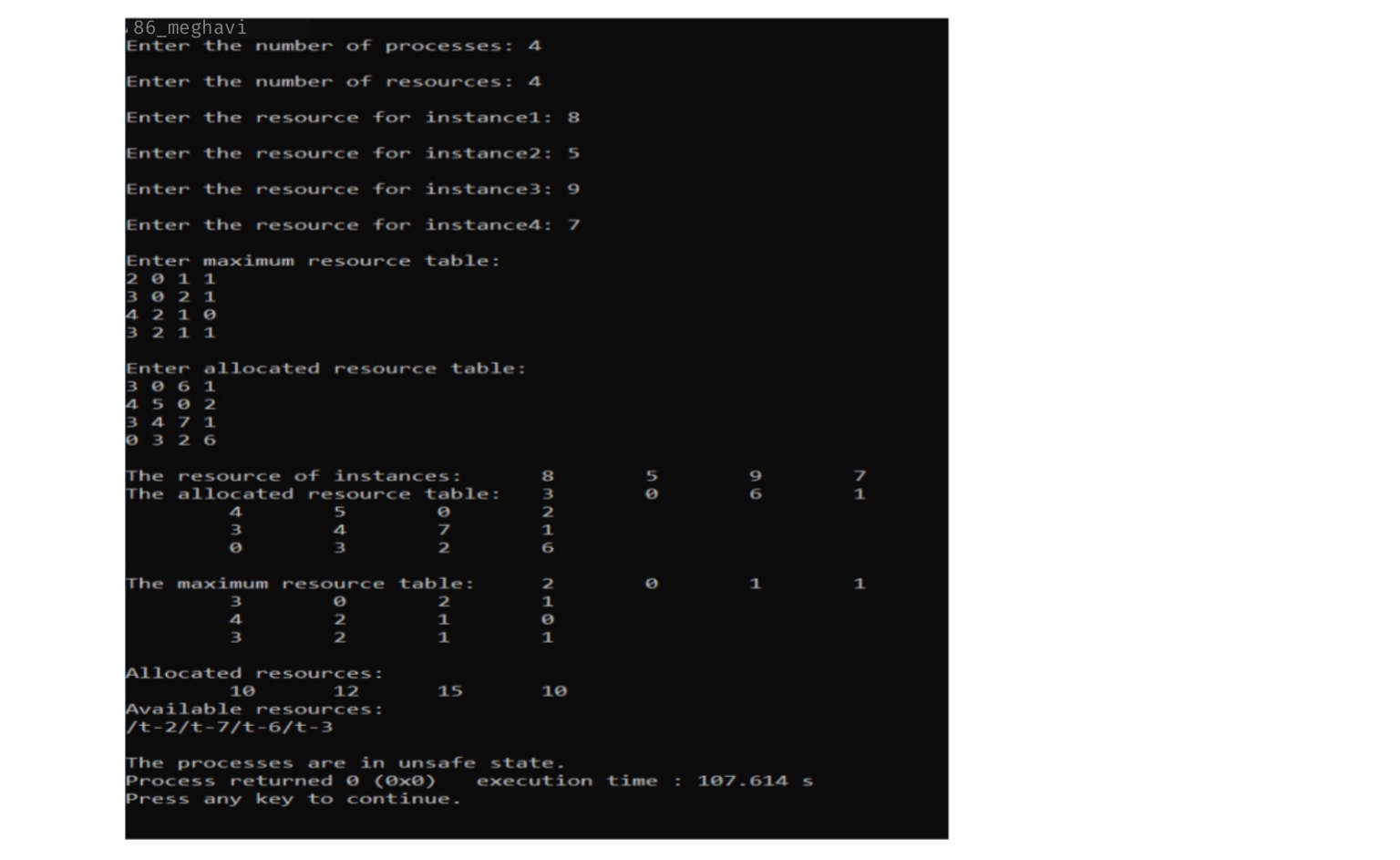
printf("\n");

}

}

}

Output:



# Experiment 11

Q. Write a program in C to implement Optimal Page Replacement algorithm.

Code :

#include<stdio.h> int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0; printf("186\_meghavi\n ");

printf("Enter number of frames: "); scanf("%d", &no\_of\_frames); printf("Enter number of pages: "); scanf("%d", &no\_of\_pages); printf("Enter page reference string: "); for(i = 0; i < no\_of\_pages; ++i){ scanf("%d", &pages[i]);

}

for(i = 0; i < no\_of\_frames; ++i){ frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){ flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == pages[i]){

flag1 = flag2 = 1; break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == -1){

faults++;

frames[j] = pages[i]; flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j < no\_of\_frames; ++j){ temp[j] = -1;

for(k = i + 1; k < no\_of\_pages; ++k){ if(frames[j] == pages[k])

{

temp[j] = k; break;

}

}

}

for(j = 0; j < no\_of\_frames; ++j){ if(temp[j] == -1){

pos = j; flag3 = 1; break;

}

}

if(flag3 ==0){ max = temp[0]; pos = 0;

for(j = 1; j < no\_of\_frames; ++j){ if(temp[j] > max){

max = temp[j]; pos = j;

}

}

}

frames[pos] = pages[i]; faults++;

}

printf("\n");

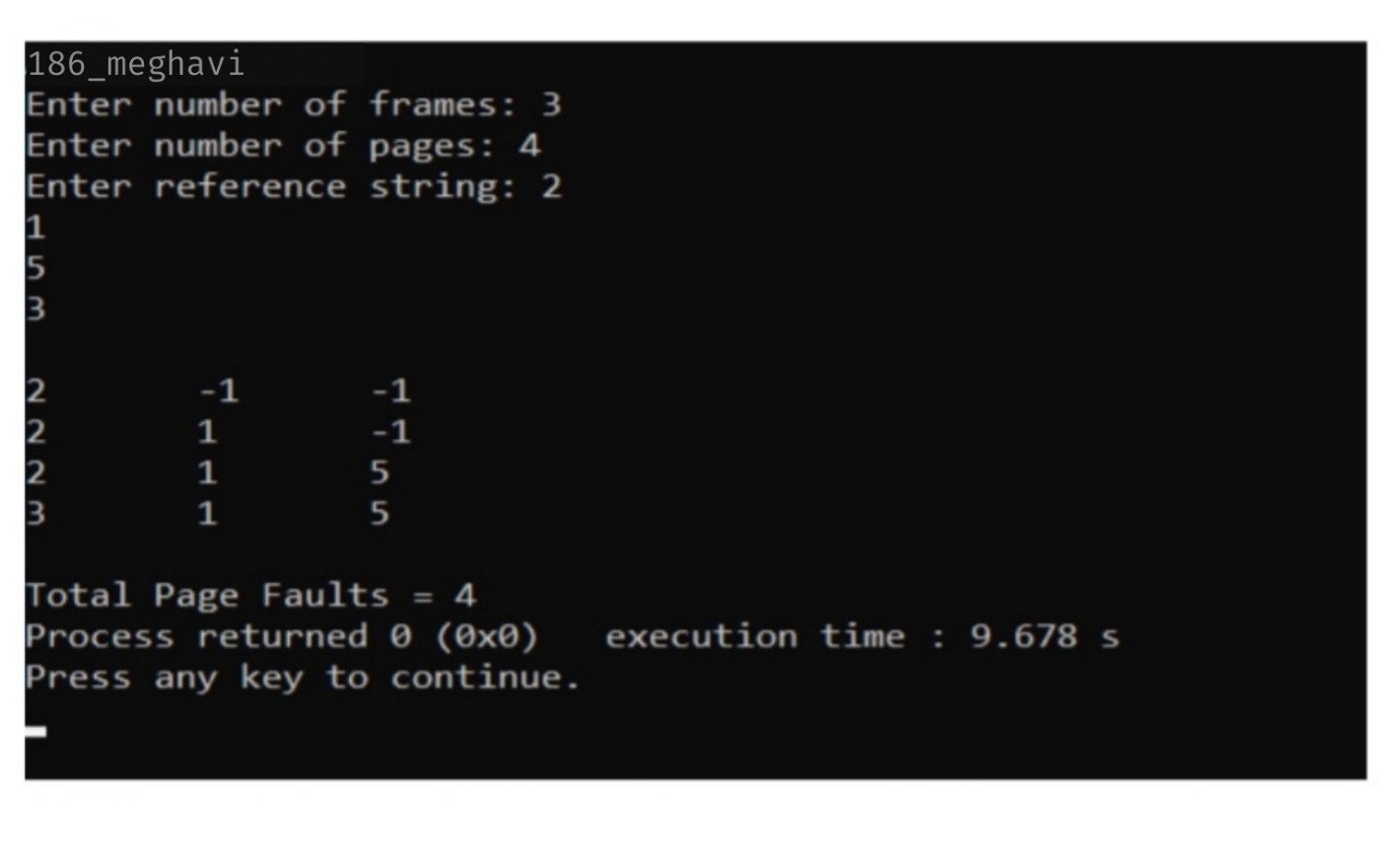
for(j = 0; j < no\_of\_frames; ++j){ printf("%d\t", frames[j]);

}

}

printf("\n\nTotal Page Faults = %d", faults); return 0;

}

Output:

# Experiment 12

Q. Write a program in C to implement FIFO algorithm.

Code :

#include <stdio.h> int main()

{

int referenceString[10], pageFaults = 0, m, n, s, pages, frames; printf("186\_meghavi\n”);

printf("\nEnter the number of Pages:\t"); scanf("%d", &pages);

printf("\nEnter reference string values:\n"); for( m = 0; m < pages; m++)

{

printf("Value No. [%d]:\t", m + 1); scanf("%d", &referenceString[m]);

}

printf("\n What are the total number of frames:\t");

{

scanf("%d", &frames);

}

int temp[frames];

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

{

temp[m] = -1;

}

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

{

s = 0;

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

{

if(referenceString[m] == temp[n])

{ s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = referenceString[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = referenceString[m];

}

printf("\n");

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

{

printf("%d\t", temp[n]);

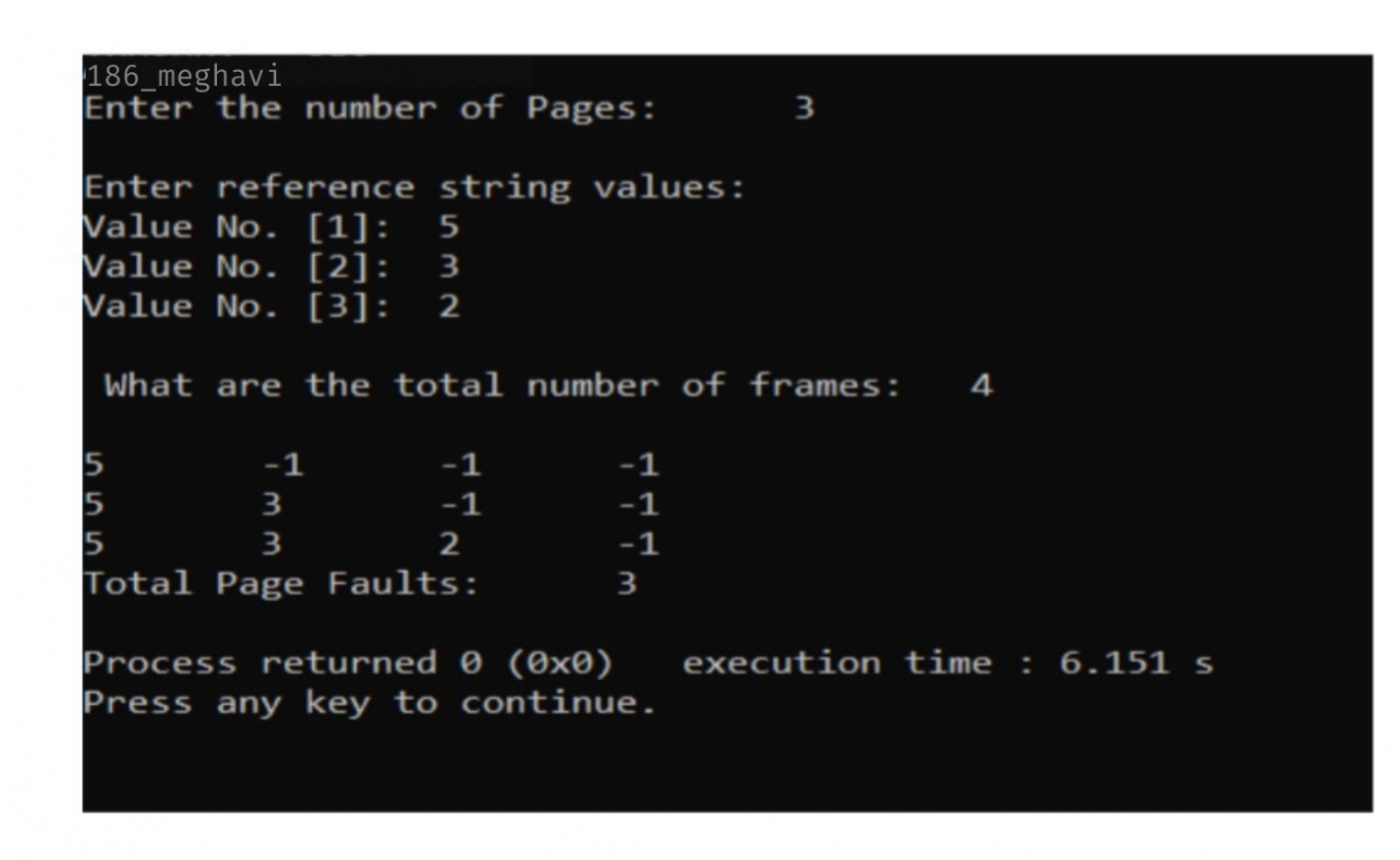
}

}

printf("\nTotal Page Faults:\t%d\n", pageFaults); return 0;

}

Output:



# Experiment 13

Q. Write a program in C to implement LRU algorithm.

Code :

#include<stdio.h>

int findLRU(int time[], int n)

{

int i, minimum = time[0], pos = 0; for(i = 1; i < n; ++i){

if(time[i] < minimum){ minimum = time[i]; pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("186\_meghavi\n"); printf("Enter number of frames: "); scanf("%d", &no\_of\_frames); printf("Enter number of pages: "); scanf("%d", &no\_of\_pages); printf("Enter reference string: "); for(i = 0; i < no\_of\_pages; ++i){ scanf("%d", &pages[i]);

}

for(i = 0; i < no\_of\_frames; ++i){ frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i)

{

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j)

{

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

{

counter++; time[j] = counter; flag1 = flag2 = 1; break;

}

}

if(flag1 == 0)

{

for(j = 0; j < no\_of\_frames; ++j)

{

if(frames[j] == -1)

{

counter++; faults++;

frames[j] = pages[i]; time[j] = counter; flag2 = 1;

break;

}

}

}

if(flag2 == 0)

{

pos = findLRU(time, no\_of\_frames); counter++;

faults++;

frames[pos] = pages[i]; time[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j)

{

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

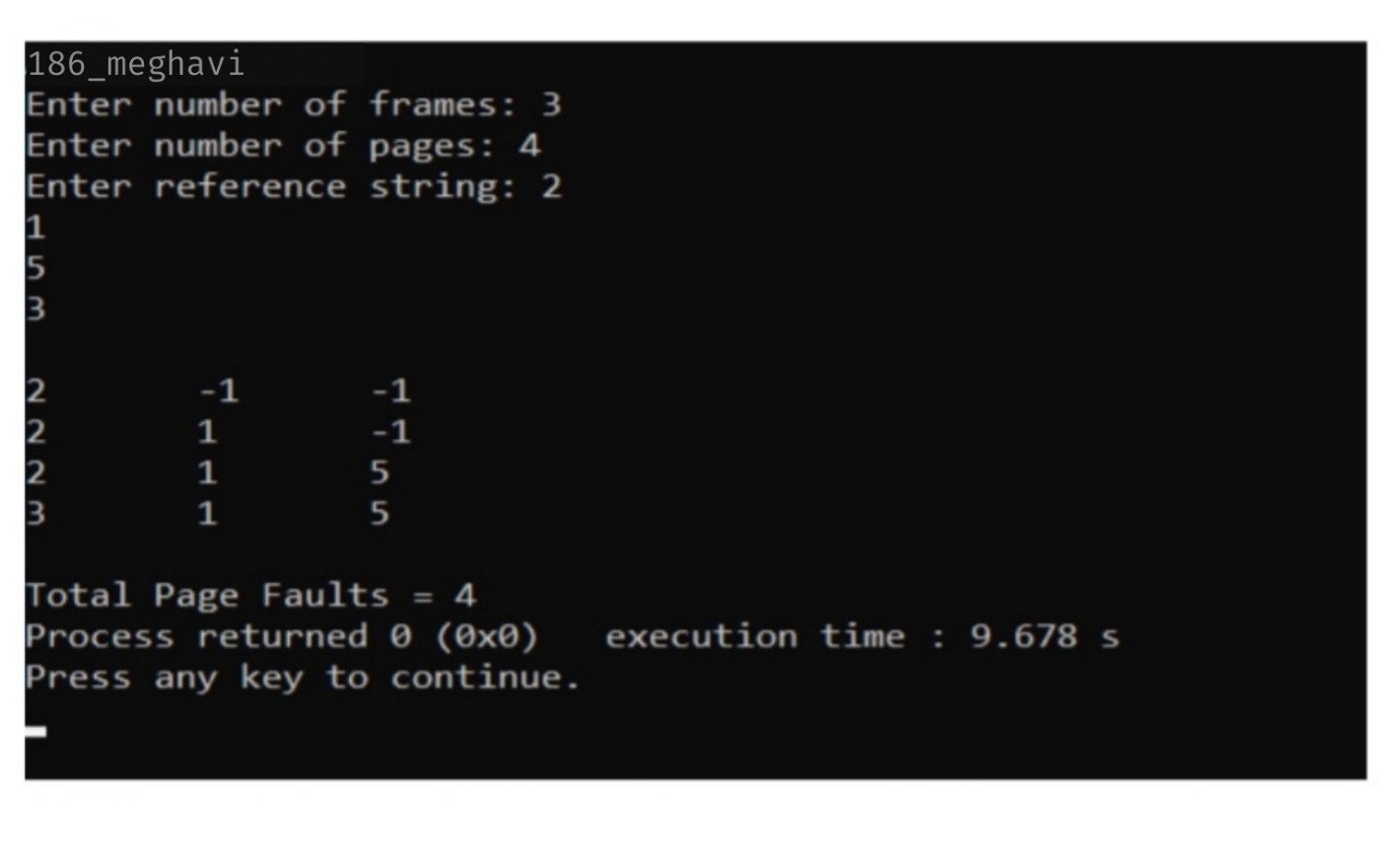
}

}

printf("\n\nTotal Page Faults = %d", faults); return 0;

}

Output:



# Experiment 14

Q. Write a program in C to implement SCAN Disk Scheduling algorithm.

Code:

#include<stdio.h> int main()

{

int d[20],i,j,sum=0,n,disk,temp,max,dloc; printf("186\_meghavi\n "); printf("enter number of location\t"); scanf("%d",&n);

printf("enter position of head\t"); scanf("%d",&disk);

printf("enter elements of disk queue\n"); for(i=0;i<n;i++)

{

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

}

d[n]=disk; n=n+1; for(i=0;i<n;i++)

{

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

{

if(d[i]>d[j])

{

temp=d[i]; d[i]=d[j]; d[j]=temp;

}

}

}

max=d[n]; for(i=0;i<n;i++)

{

if(disk==d[i])

{

dloc=i; break;

}

}

for(i=dloc;i>=0;i--)

{

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

}

printf("0 -->");

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

{

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

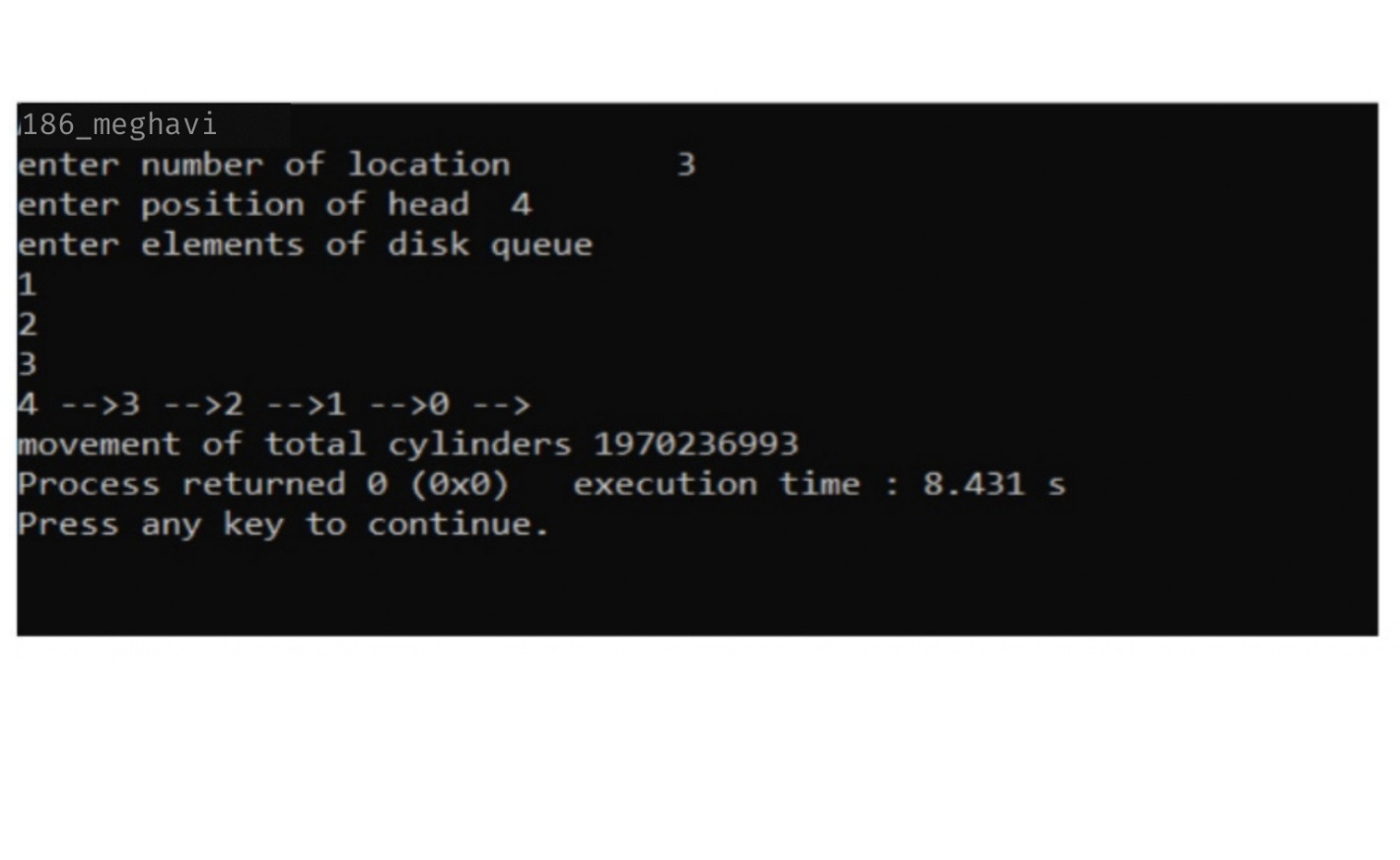
}

sum=disk+max;

printf("\nmovement of total cylinders %d",sum); return 0;

}

Output:



# Experiment 15

Q. Write a program in C to implement Shortest Seek Time First Disk Scheduling algorithm.

Code:

#include<stdio.h> #include<stdlib.h> int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial,count=0; printf("186\_meghavi\n ");

printf("Enter the number of Requests\n"); scanf("%d",&n);

printf("Enter the Requests sequence\n"); for(i=0;i<n;i++)

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

printf("Enter initial head position\n"); scanf("%d",&initial); while(count!=n)

{

int min=1000,d,index; for(i=0;i<n;i++)

{

d=abs(RQ[i]-initial); if(min>d)

{

min=d; index=i;

}

}

TotalHeadMoment=TotalHeadMoment+min; initial=RQ[index];

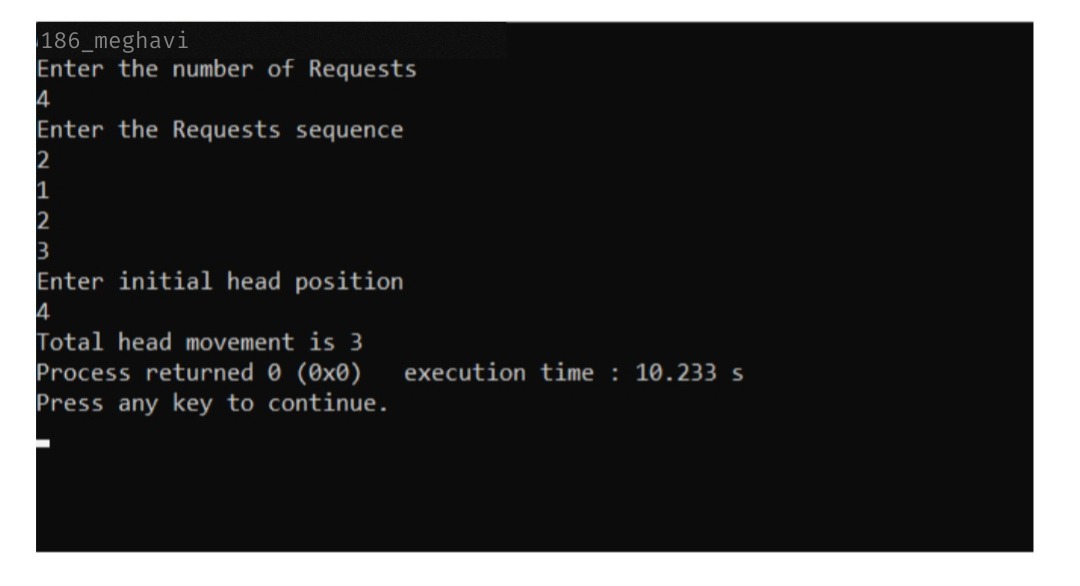
RQ[index]=1000; count++;

}

printf("Total head movement is %d",TotalHeadMoment); return 0;

}

Output :



# Experiment 16

Q. Write a program in C to implement C-SCAN Disk Scheduling algorithm.

Code :

#include<stdio.h> #include<stdlib.h> int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move; printf("186\_meghavi\n ");

printf("Enter the number of Requests\n"); scanf("%d",&n);

printf("Enter the Requests sequence\n"); for(i=0;i<n;i++)

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

printf("Enter initial head position\n"); scanf("%d",&initial);

printf("Enter total disk size\n"); scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n"); scanf("%d",&move);

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

{

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

{ if(RQ[j]>RQ[j+1])

{

int temp; temp=RQ[j]; RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

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

{

if(initial<RQ[i])

{

index=i; break;

}

}

if(move==1)

{

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

// last movement for max size TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);

/\*movement max to min disk \*/ TotalHeadMoment=TotalHeadMoment+abs(size-1-0); initial=0;

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

}

else

{

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0); TotalHeadMoment=TotalHeadMoment+abs(size-1-0); initial =size-1;

for(i=n-1;i>=index;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

}

printf("Total head movement is %d",TotalHeadMoment); return 0;

}

Output:

