**Shortest Job First(Preemptive) scheduling Algorithm:**

**Input-**

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes", limit);

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

{

printf("\nEnter Arrival Time:\t");

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

printf("Enter Burst Time:\t");

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

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

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

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

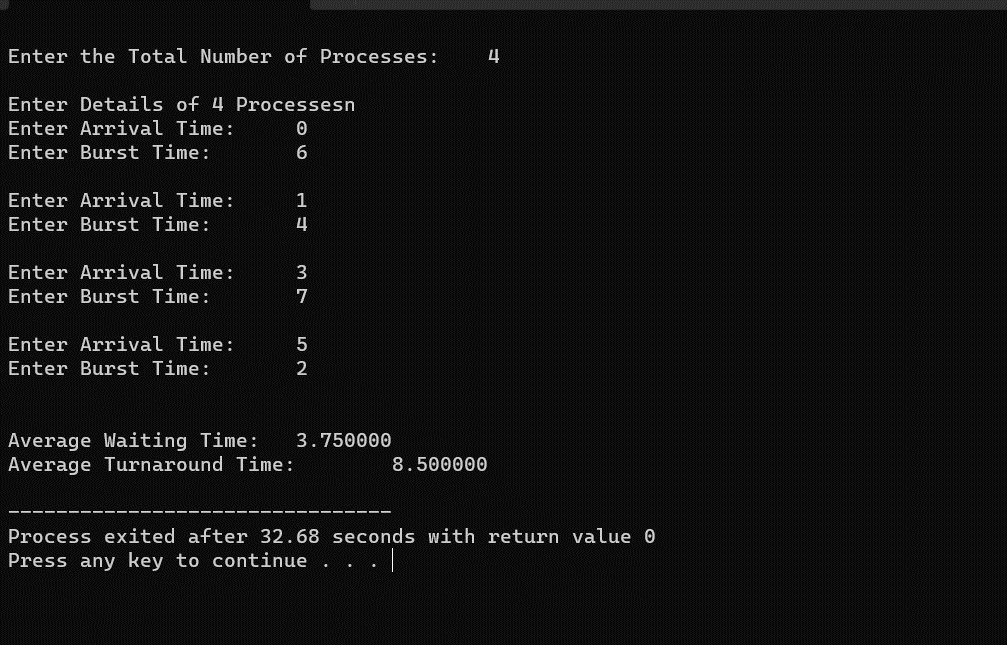
printf("\n\nAverage Waiting Time:\t%lf\n", average\_waiting\_time);

printf("Average Turnaround Time:\t%lf\n", average\_turnaround\_time);

return 0;

}

**Output-**



**Shortest Job First(Non-Preemptive) scheduling Algorithm:**

**Input –**

#include<stdio.h>

int main() {

int time, burst\_time[10], at[10], sum\_burst\_time = 0, smallest, n, i;

int sumt = 0, sumw = 0;

printf("enter the no of processes : ");

scanf("%d", & n);

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

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

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

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

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

sum\_burst\_time += burst\_time[i];

}

burst\_time[9] = 9999;

for (time = 0; time < sum\_burst\_time;) {

smallest = 9;

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

if (at[i] <= time && burst\_time[i] > 0 && burst\_time[i] < burst\_time[smallest])

smallest = i;

}

printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst\_time[smallest] - at[smallest], time - at[smallest]);

sumt += time + burst\_time[smallest] - at[smallest];

sumw += time - at[smallest];

time += burst\_time[smallest];

burst\_time[smallest] = 0;

}

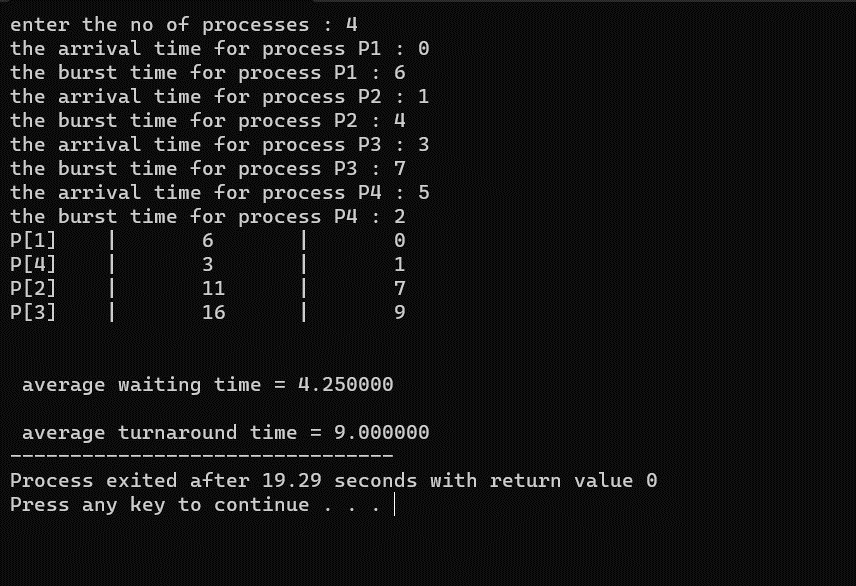
printf("\n\n average waiting time = %f", sumw \* 1.0 / n);

printf("\n\n average turnaround time = %f", sumt \* 1.0 / n);

return 0;

}

**Output-**



**FCFS Scheduling Algorithm:**

**Input-**

#include <stdio.h>

int main()

{

int pid[15];

int bt[15];

int n;

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

scanf("%d",&n);

printf("Enter process id of all the processes: ");

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

{

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

}

printf("Enter burst time of all the processes: ");

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

{

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

}

int i, wt[n];

wt[0]=0;

//for calculating waiting time of each process

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

{

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

}

printf("Process ID Burst Time Waiting Time TurnAround Time\n");

float twt=0.0;

float tat= 0.0;

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

{

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

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

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

//calculating and printing turnaround time of each process

printf("%d\t\t", bt[i]+wt[i]);

printf("\n");

//for calculating total waiting time

twt += wt[i];

//for calculating total turnaround time

tat += (wt[i]+bt[i]);

}

float att,awt;

//for calculating average waiting time

awt = twt/n;

//for calculating average turnaround time

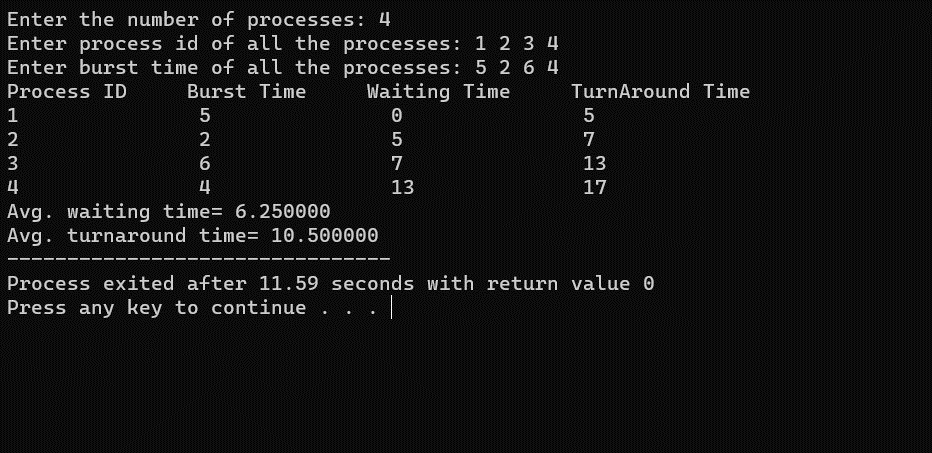
att = tat/n;

printf("Avg. waiting time= %f\n",awt);

printf("Avg. turnaround time= %f",att);

}

**Output-**



**Round Robin Scheduling Algorithm:**

**Input-**

#include<stdio.h>

void main()

{

// initlialize the variable name

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(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t"); // Accept arrival time

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

printf(" \nBurst time is: \t"); // Accept the Burst time

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

temp[i] = bt[i]; // store the burst time in temp array

}

// Accept the Time qunat

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time

printf("\n Process 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) // define the conditions

{

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--; //decrement the process no.

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;

}

}

// represents the average waiting time and Turn Around time

avg\_wt = wt \* 1.0/NOP;

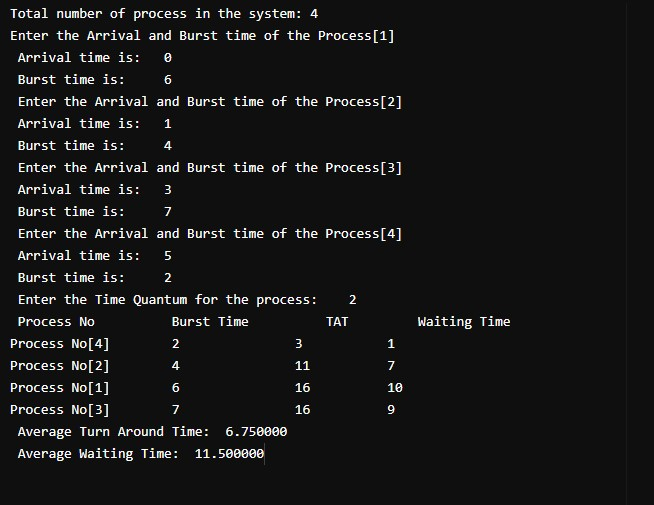
avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f", avg\_tat);

}

**Output-**



**Bankers Algorithm :**

**Input-**

#include<stdio.h>

void main()

{

char pro[10]={'A','B','C','D','E','F','G','H','I','J'},seq[10];

int avlbl[10],resrc[10],max[10][10],alloc[10][10],need[10][10],i,j,flag=0;

int proc,res,count=0,temp[10],temp1[10];

printf("ENTER THE NO. OF PROCESS=");

scanf("%d",&proc);

printf("ENTER THE NO. OF RESOURCE TYPES=");

scanf("%d",&res);

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

{

temp[i]=0;

temp1[i]=0;

}

printf("ENTER THE CURRENTLY AVAILABLE RESOURCES OF EACH PROCESS(ALLOCATION MATRIX):\n");

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

{

printf("FOR PROCESS %c",pro[i]);

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

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

}

printf("ENTER THE MAXIMUM REQUIRED RESOURCES OF EACH PROCESS(MAXIMUM MATRIX):\n");

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

{

printf("FOR PROCESS %c",pro[i]);

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

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

}

printf("NEED OF RESOURCES OF EACH PROCESS(NEED MATRIX):");

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

{

printf("\n FOR PROCESS %c",pro[i]);

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

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

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

}

printf("\n ENTER THE RESOURCE INSTANCES");

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

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

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

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

temp1[i]=temp1[i]+alloc[j][i];

printf("AVAILABLE:");

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

{

avlbl[i]=resrc[i]-temp1[i];

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

}

loop:for(i=0;i<proc;i++)

{

if(temp[i]!=1)

{

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

{

if(avlbl[j]<need[i][j])

{

flag=1;

}

}

if(flag==0)

{

printf("\n PROCESS %c EXECUTED",pro[i]);

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

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

{

avlbl[j]=avlbl[j]+alloc[i][j];

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

}

count++;

temp[i]=1;

seq[count-1]=pro[i];

}

else

flag=0;

}

}

if(count!=proc)

goto loop;

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

if(avlbl[i]==resrc[i])

{

printf("\n SAFE SEQUENCE:");

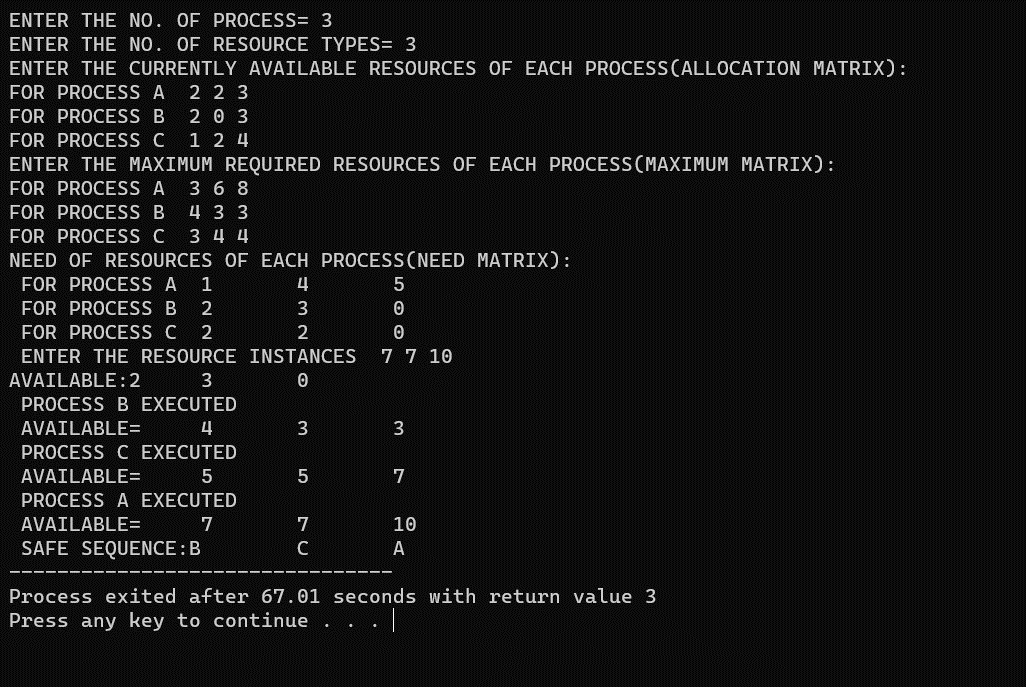
for(i=0;i<proc;i++)

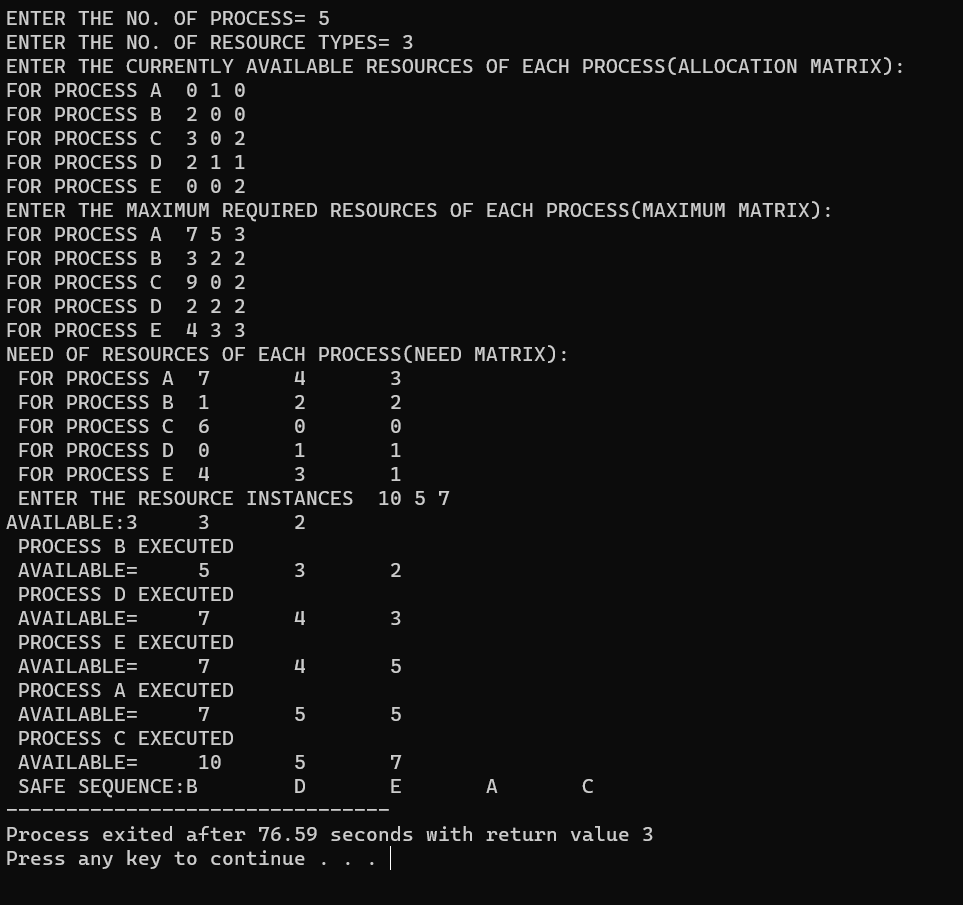
printf("%c\t",seq[i]);

}

}

**Output-**





**Exeeve program :**

**Input-**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

void sort(int a[11]);

int main(int argc, char \*argv[])

{

int pid;

int i = 0, n = 10, search;

int a[11];

char \*newarg[] = {"./sort\_program", NULL}; // Replace with correct path to your sorting program

FILE \*f;

printf("Enter array elements: ");

for (i = 1; i <= 10; i++)

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

printf("Enter value to find: ");

scanf("%d", &search);

pid = fork();

if (pid == 0)

{

printf("Child process executing %s\n", newarg[0]); // Debugging output

sleep(1);

execv(newarg[0], newarg);

perror("execv"); // Print error if execv fails

exit(1); // Terminate child process if execv fails

}

else

{

wait(NULL); // Wait for the child process to finish

sort(a);

f = fopen("sort.txt", "w");

fprintf(f, "%d\n", search);

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

{

fprintf(f, "%d ", a[i]);

}

fclose(f);

}

return 0;

}

void sort(int a[11])

{

int n = 10, i, j, temp;

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

{

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

{

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

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

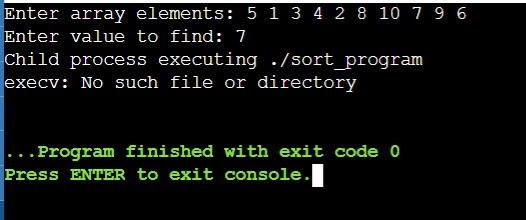
}

}

}

}

**Output-**





**Wait program :**

**Input-**

#include <stdio.h>

#include <sys/wait.h>

#include <unistd.h>

int main()

{

if (fork() == 0)

{

printf("HC: hello from child\n");

}

else

{

printf("HP: hello from parent\n");

wait(NULL);

printf("CT: child has terminated\n");

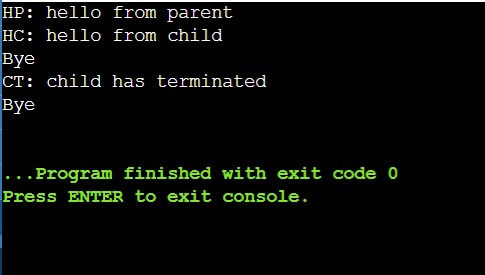
}

printf("Bye\n");

return 0;

}

**Output-**



**Producer Consumer :**

**Input-**

#include<stdio.h>

#include<stdlib.h>

void producer();

void consumer();

int wait(int);

int signal(int);

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

int main(){

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

int n;

while(1){

printf("Enter your choice\n");

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;

}

}

}

}

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 produced an item %d\n",x);

mutex=signal(mutex);

}

void consumer(){

mutex = wait(mutex);

full= wait(full);

empty = signal(empty);

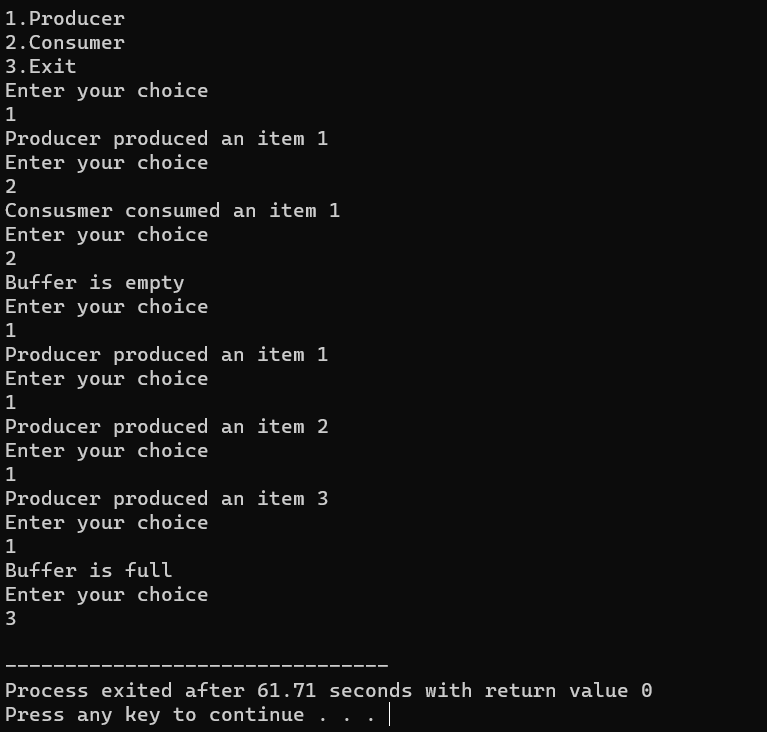
printf("Consusmer consumed an item %d\n",x);

x--;

mutex=signal(mutex);

}

**Output-**



**Student\_Record :**

**Input-**

#! /bin/bash

it=0

a=1

while [[ $op -lt 7 ]]

do

echo enter the option

echo "1 for create"

echo "2 for add"

echo "3 for display"

echo "4 for search"

echo "5 for delete"

echo "6 for modify"

echo "7 for exit"

echo "enter u r choice"

read op

word="$op"

case "$word" in

"1")

if [ "$op" == "1" ]

then

echo "Enter the name for the database"

read db

touch "$db"

fi

;;

"2")

if [ "$op" == "2" ]

then

echo "in which database u want to add records"

read db

echo "enter the no. of records"

read n

while [ $it -lt $n ]

do

echo "enter id:"

read id1

echo "enter name:"

read nm

pa1="^[A-Za-z]"

while [[ ! $add =~ $pa ]]

do

echo "enter valid address:"

read add

done

echo "enter address:"

read add

pa="^[A-Za-z0-9]"

while [[ ! $add =~ $pa ]]

do

echo "enter valid address:"

read add

done

#echo $add

echo "enter phone no.:"

read ph

pat="^[0-9]{10}$"

while [[ ! $ph =~ $pat ]]

do

echo "please enter phone number as XXXXXXXXXX:"

read ph

done

#echo $ph

echo "enter email:"

read em

patem="^[a-z0-9.\_%-+]+@[a-z]+\.[a-z]{2,4}$"

while [[ ! $em =~ $patem ]]

do

echo "please enter valid email address"

read em

done

#echo $em

echo "$id1,$nm,$add,$ph,$em" >> "$db"

it=`expr $it + 1`

echo "$it record entered"

done

fi

;;

"3")

if [ "$op" == "3" ]

then

echo "enter name of database from where data to be display:"

read db

cat $db

fi

;;

"4")

if [ "$op" == "4" ]

then

echo "enter name of database from where to search:"

read db

echo "enter email to be search:"

read em1

grep $em1 $db

echo "record found"

else

echo "not found"

fi

;;

"5")

if [ "$op" == "5" ]

then

echo "enter name of database:"

read db

echo "enter id:"

read id1

echo "enter line no. u want to delete:"

read linenumber

for line in `grep -n "$id1" $db`

do

number=`echo "$line" | cut -c1`

#echo $number

if [ $number == $linenumber ]

then

lineRemove="${linenumber}d"

sed -i -e "$lineRemove" $db

echo "record removed"

fi

#echo

cat $db

done

fi

;;

"6")

if [ "$op" == "6" ]

then

echo "enter name of database:"

read db

echo "enter id:"

read id1

echo "enter line u want to modify:"

read linenumber

for line in `grep -n "$id1" "$db"`

do

number=`echo "$line" | cut -c1`

if [ "$number" == "$linenumber" ]

then

echo "what would u like to change"

echo "\"id,name,address,mobile,email\""

read edit

linechange="${linenumber}s"

sed -i -e "$linechange/.\*/$edit/" $db

echo record edited

fi

done

fi

;;

"7")

echo "bye"

;;

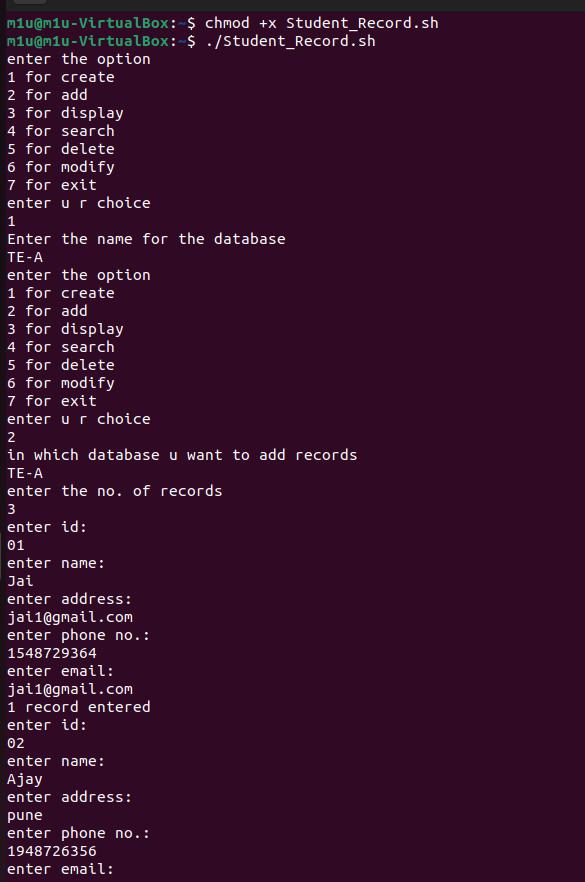
\*) echo invalid input

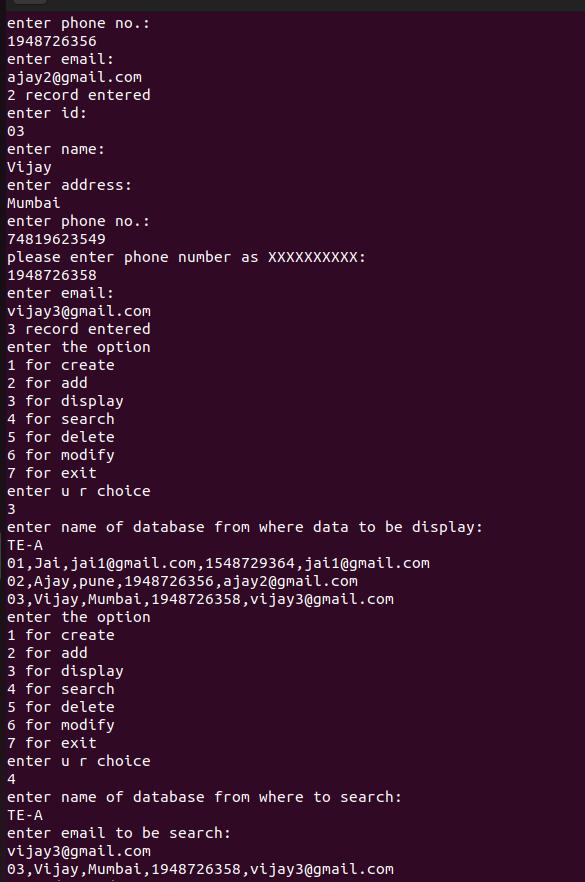
esac

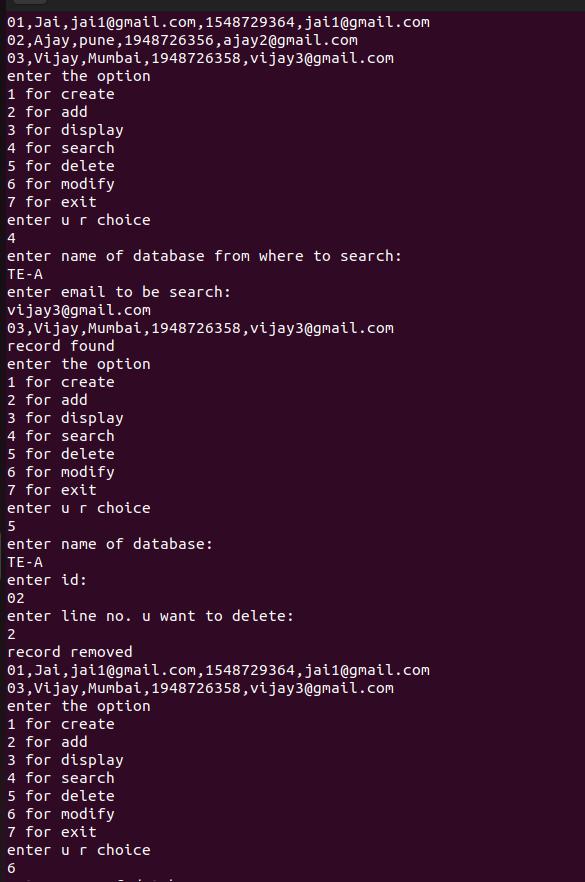
done

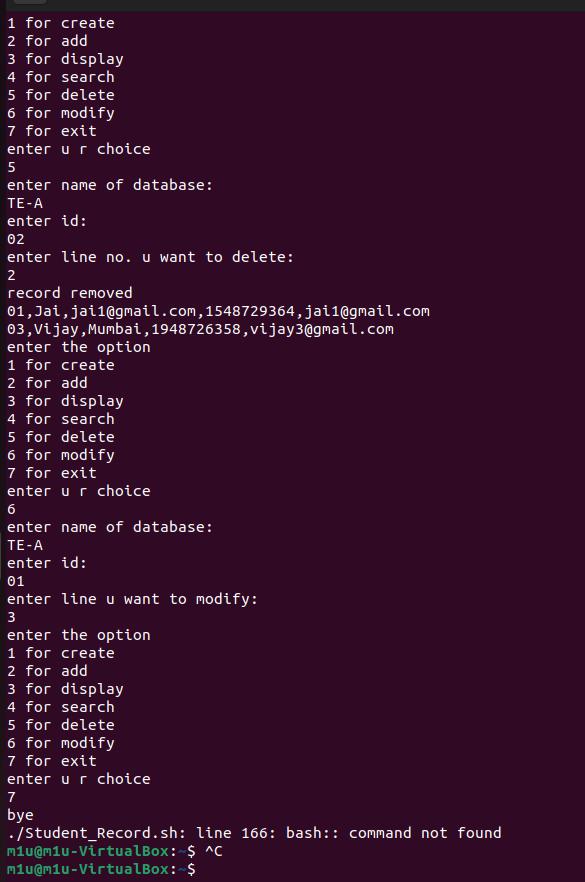
bash: line 1: chmod+x: command not found

**Output-**





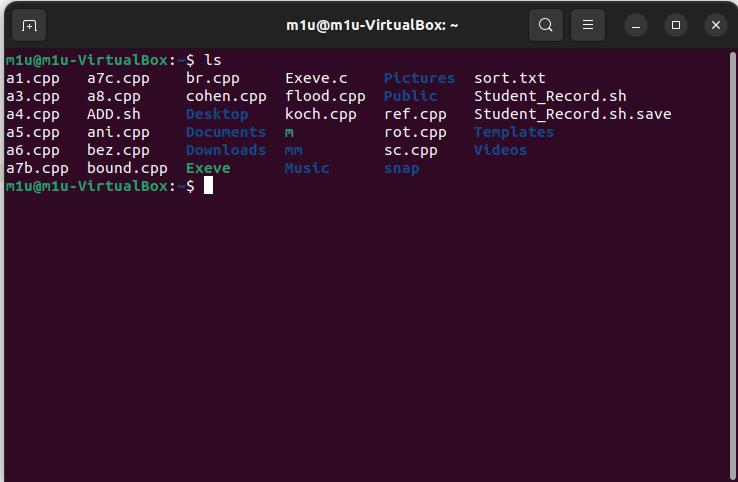




**Basic Linux Commands :**

**ls Command**

The [ls](https://www.javatpoint.com/linux-ls) command is used to display a list of content of a directory.

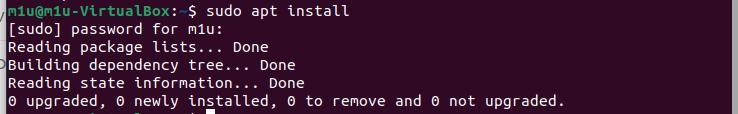


**Echo Command**



**Sudo apt install**

Used to install the libraries.



**pwd Command**

The [pwd](https://www.javatpoint.com/linux-pwd) command is used to display the location of the current working directory.



**mkdir Command**

The [mkdir](https://www.javatpoint.com/linux-mkdir) command is used to create a new directory under any directory.



**cd Command**

The [cd](https://www.javatpoint.com/linux-cd) command is used to change the current directory.



**nano Command**

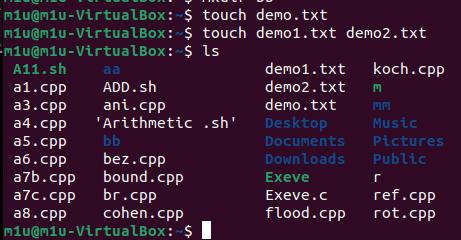
Opens the command line text editor.





**touch Command**

The [touch](https://www.javatpoint.com/linux-touch) command is used to create empty files. We can create multiple empty files by executing it once.



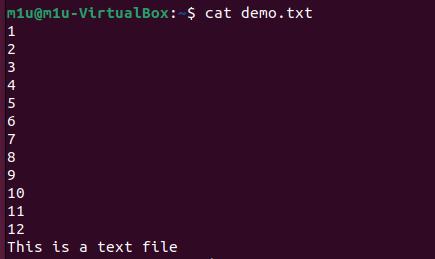
**Uname**

The command ‘uname‘ displays the information about the system.



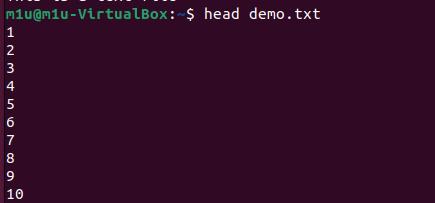
**cat Command**

The [cat](https://www.javatpoint.com/linux-cat) command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.



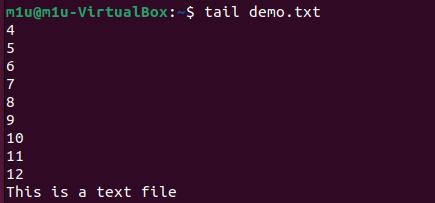
**head Command**

The [head](https://www.javatpoint.com/linux-head) command is used to display the content of a file. It displays the first 10 lines of a file.



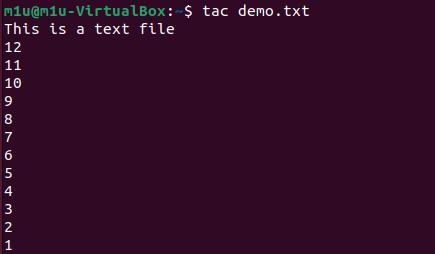
**tail Command**

The [tail](https://www.javatpoint.com/linux-tail) command is similar to the head command. The difference between both commands is that it displays the last ten lines of the file content. It is useful for reading the error message.



**tac Command**

The [tac](https://www.javatpoint.com/linux-tac) command is the reverse of cat command, as its name specified. It displays the file content in reverse order (from the last line).



**grep command**

**grep**or global regular expression print. It lets you find a word by searching through all the texts in a specific file.



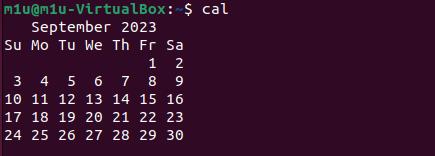
**date Command**

The [date](https://www.javatpoint.com/linux-date) command is used to display date, time, time zone, and more.



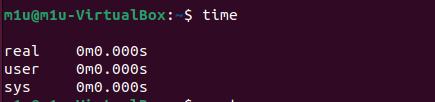
**cal Command**

The [cal](https://www.javatpoint.com/linux-cal) command is used to display the current month's calendar with the current date highlighted.



**time Command**

The [time](https://www.javatpoint.com/linux-time) command is used to display the time to execute a command.



**clear Command**

Linux **clear** command is used to clear the terminal screen.



**exit Command**

Linux [exit](https://javatpoint.com/linux-exit-command) command is used to exit from the current shell. It takes a parameter as a number and exits the shell with a return of status number.



**Reader Writer :**

**Input-**

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

sem\_t x, y;

pthread\_t tid;

pthread\_t writerthreads[100], readerthreads[100];

int readercount;

void \*reader(void \*param) {

sem\_wait(&x);

readercount++;

if (readercount == 1)

sem\_wait(&y);

sem\_post(&x);

printf("\n%d reader is inside", readercount);

sem\_wait(&x);

readercount--;

if (readercount == 0) {

sem\_post(&y);

}

sem\_post(&x);

printf("\n%d Reader is leaving", readercount + 1);

}

void \*writer(void \*param) {

printf("\nWriter is trying to enter");

sem\_wait(&y);

printf("\nWriter has entered");

sem\_post(&y);

printf("\nWriter is leaving");

}

int main() {

int n2, i;

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

scanf("%d", &n2);

int n1[n2];

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

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

for (i = 0; i < n2; i++) {

pthread\_create(&writerthreads[i], NULL, reader, NULL);

pthread\_create(&readerthreads[i], NULL, writer, NULL);

}

for (i = 0; i < n2; i++) {

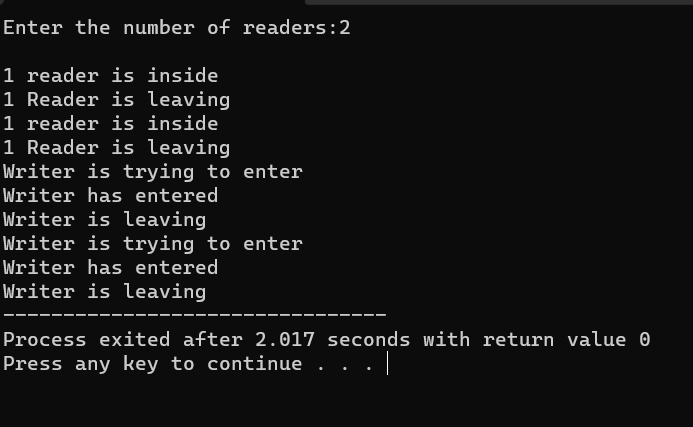
pthread\_join(writerthreads[i], NULL);

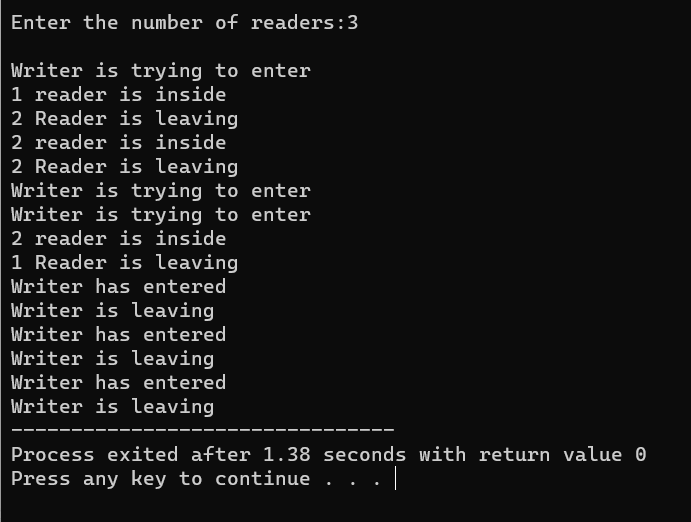
pthread\_join(readerthreads[i], NULL);

}

}

**Output-**





**FCFS Paging Algorithm:**

**Input-**

#include <stdio.h>

int main() {

int pg[20], pgs, frm, frms[10], i, j, k = 0, hit = 0, flag = 0;

float hitrt, missrt;

printf("ENTER THE NO. OF PAGES: ");

scanf("%d", &pgs);

printf("ENTER THE PAGE VALUES:\n");

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

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

printf("ENTER THE FRAME SIZE: ");

scanf("%d", &frm);

// Initialize frames to -1

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

frms[i] = -1;

printf("INITIAL PAGE VALUES:\n");

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

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

printf("\n");

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

flag = 0;

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

if (pg[i] == frms[j]) {

printf("HIT:\t");

hit++;

flag = 1;

break;

}

}

if (flag == 0) {

printf("MISS:\t");

frms[k] = pg[i];

k = (k + 1) % frm; // Move to the next frame using circular queue

}

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

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

printf("\n");

}

printf("NO. OF HITS = %d\n", hit);

hitrt = (float)hit / (float)pgs;

missrt = 1 - hitrt;

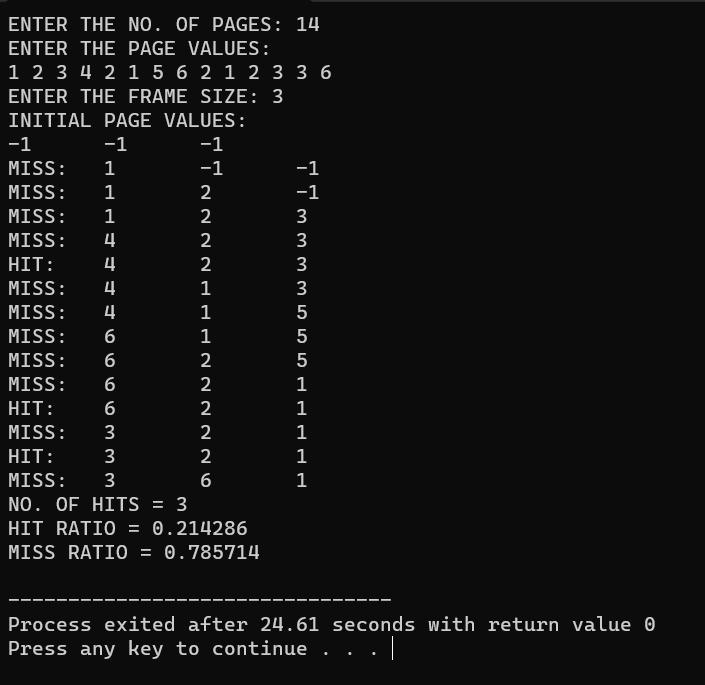
printf("HIT RATIO = %f\n", hitrt);

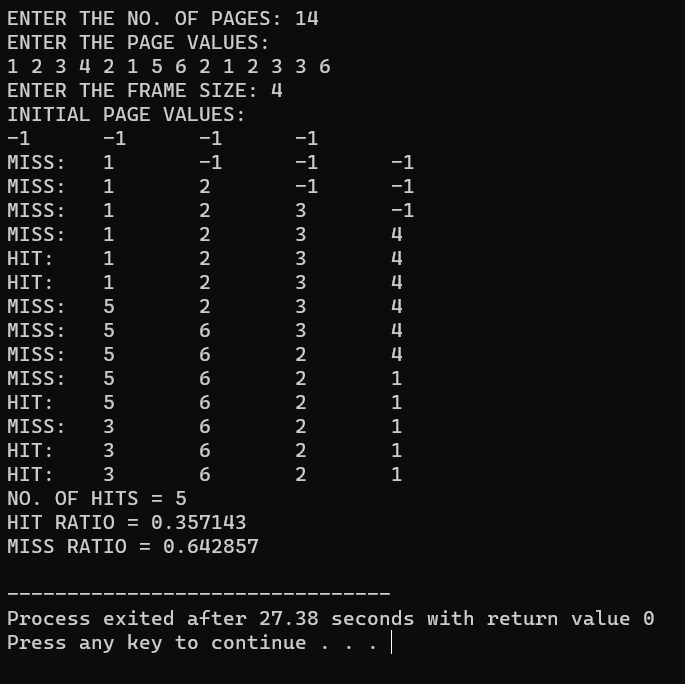
printf("MISS RATIO = %f\n", missrt);

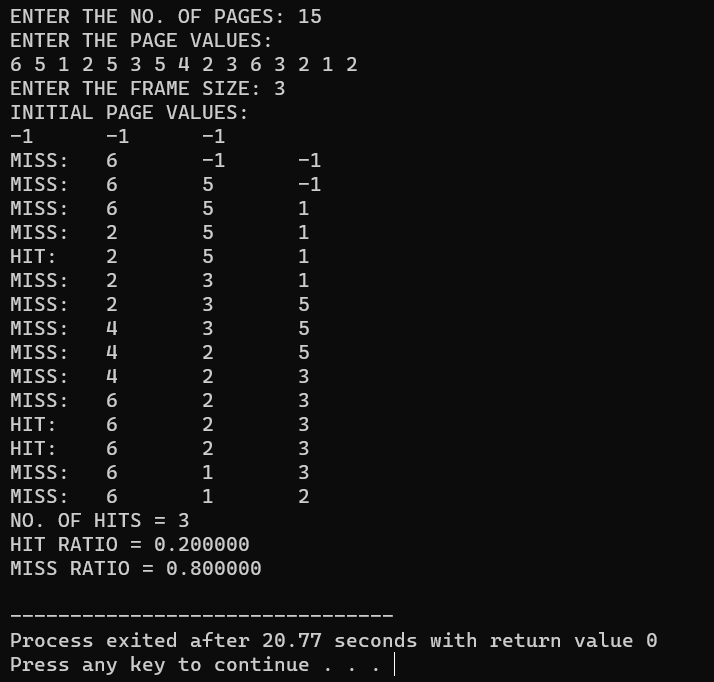
return 0;

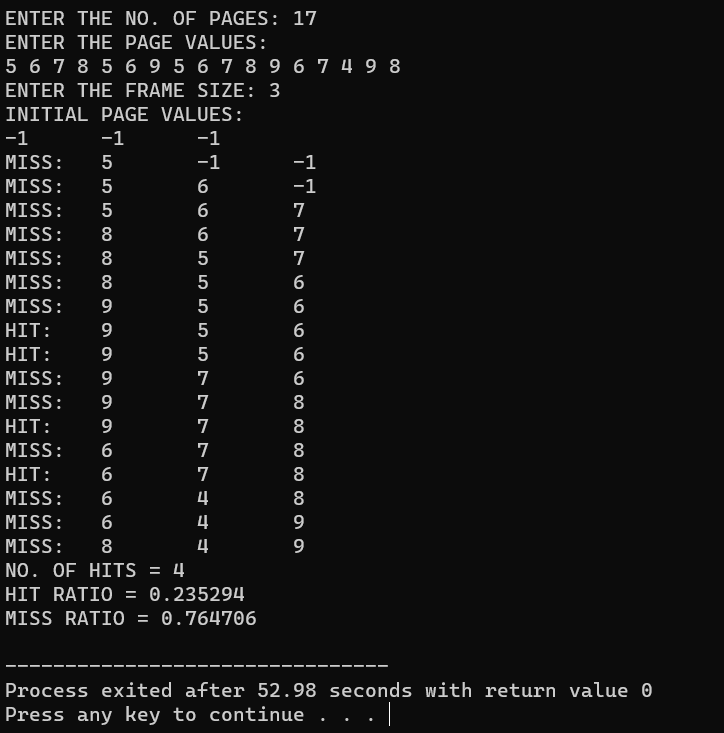
}

**Output-**









**Least Recently Used (LRU) Paging Algorithm :**

**Input-**

#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("Enter the number of frames: ");

scanf("%d", &no\_of\_frames);

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

scanf("%d", &no\_of\_pages);

printf("Enter the 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) {

frames[j] = pages[i];

time[j] = counter;

counter++;

faults++;

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]);

}

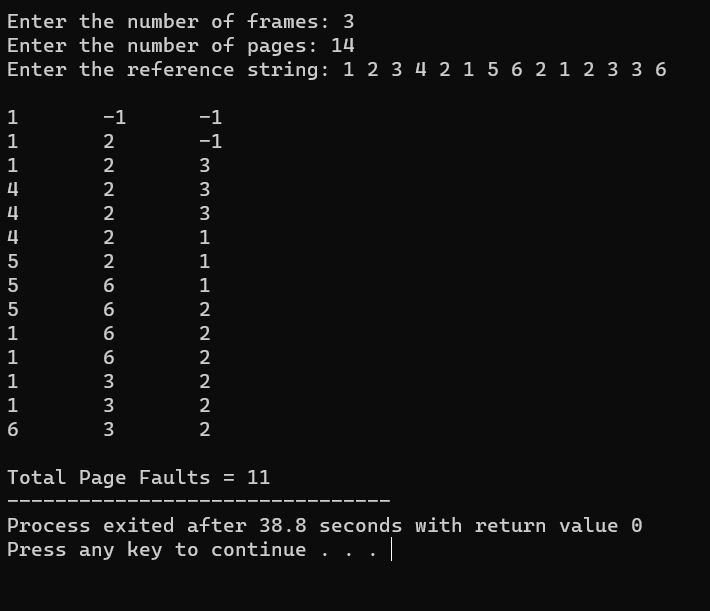
}

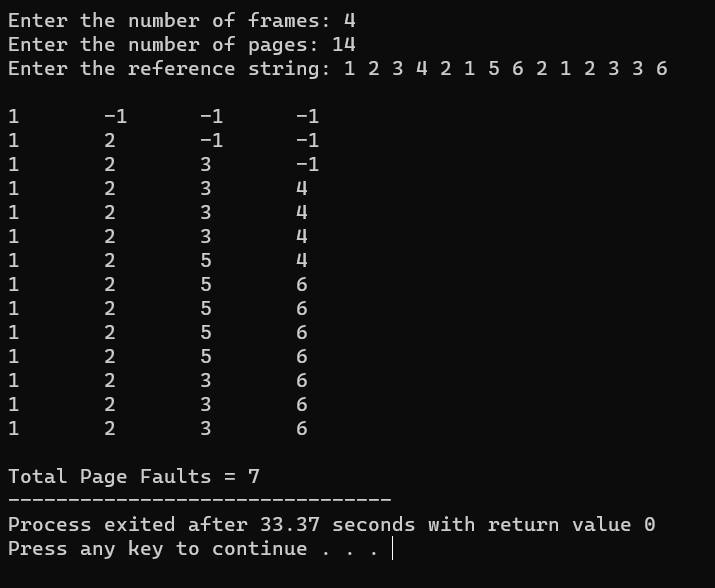
printf("\n\nTotal Page Faults = %d", faults);

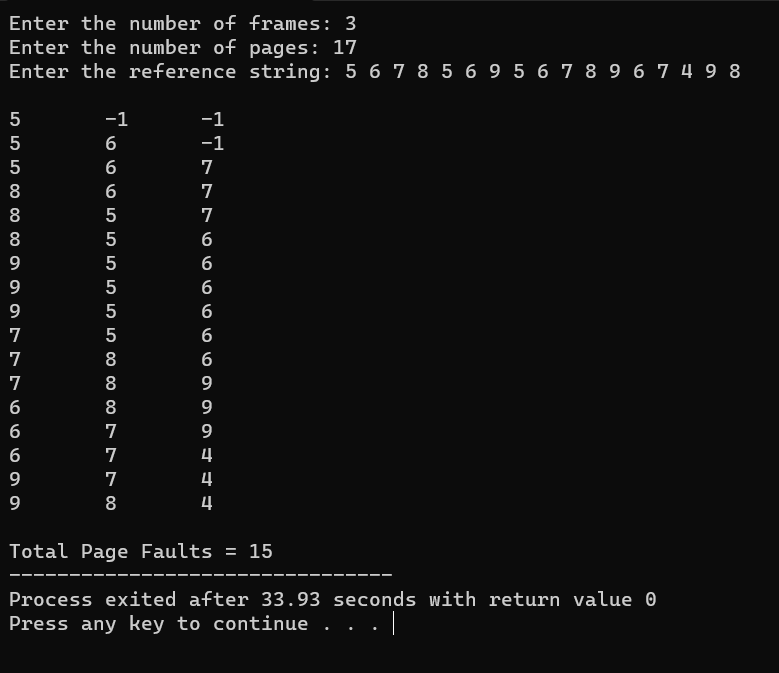
return 0;

}

**Output-**







**Optimal Paging Algorithm :**

**Input-**

#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("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]);

}

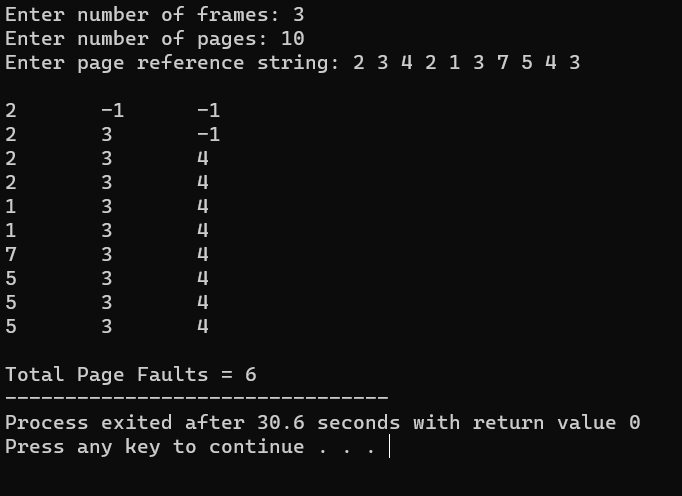
}

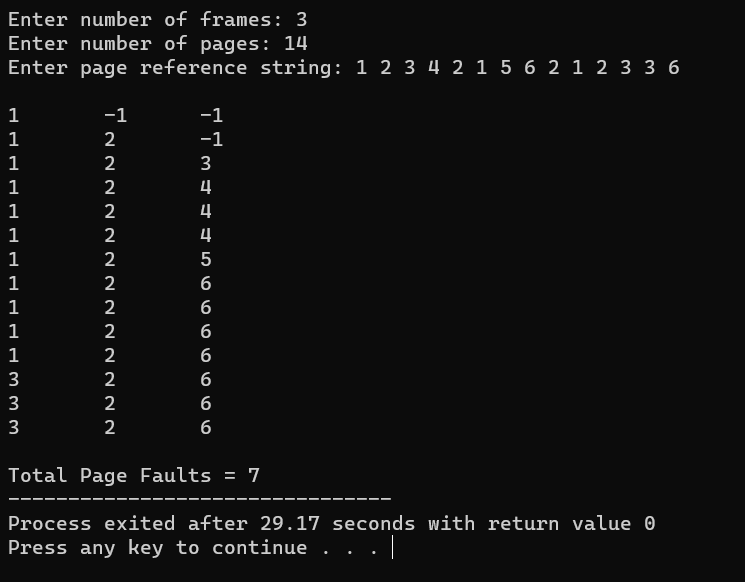
printf("\n\nTotal Page Faults = %d", faults);

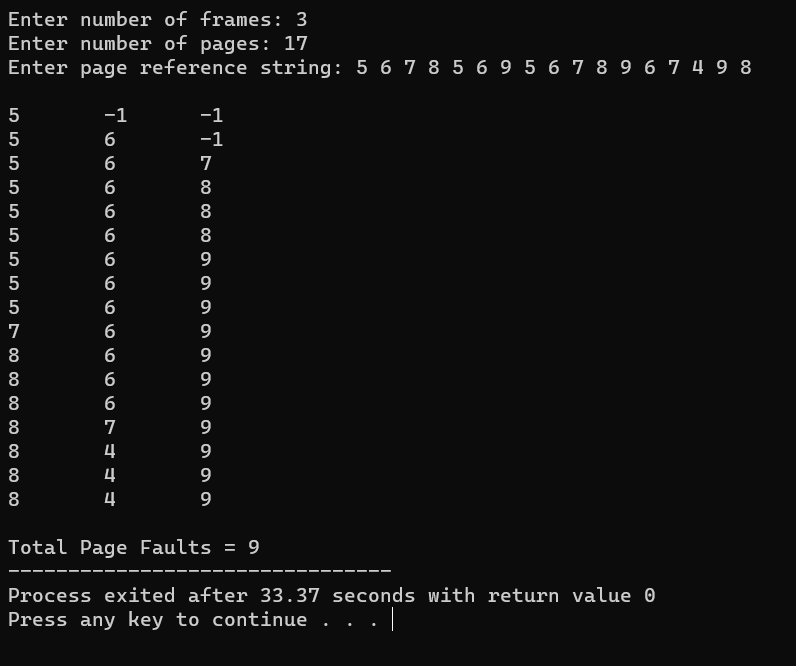
return 0;

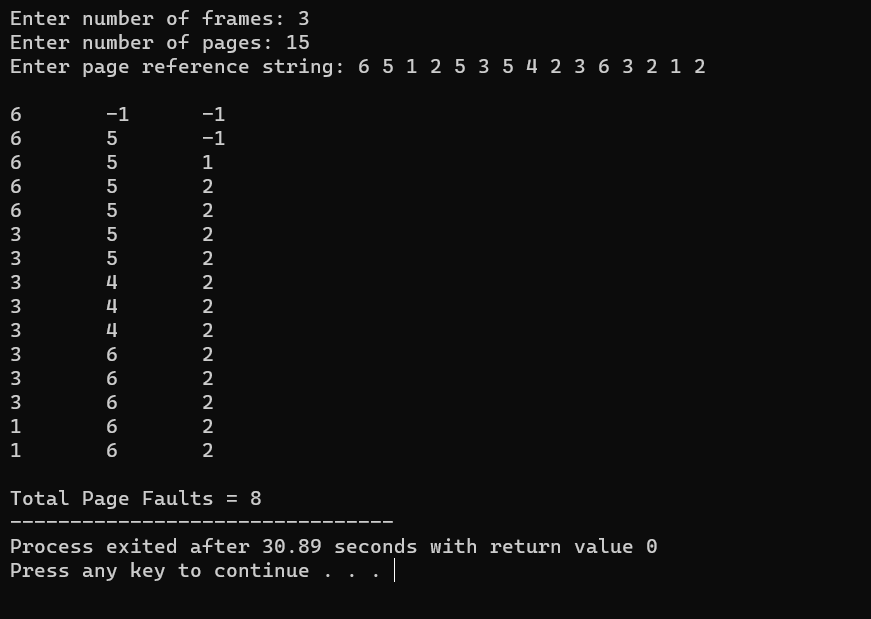
}

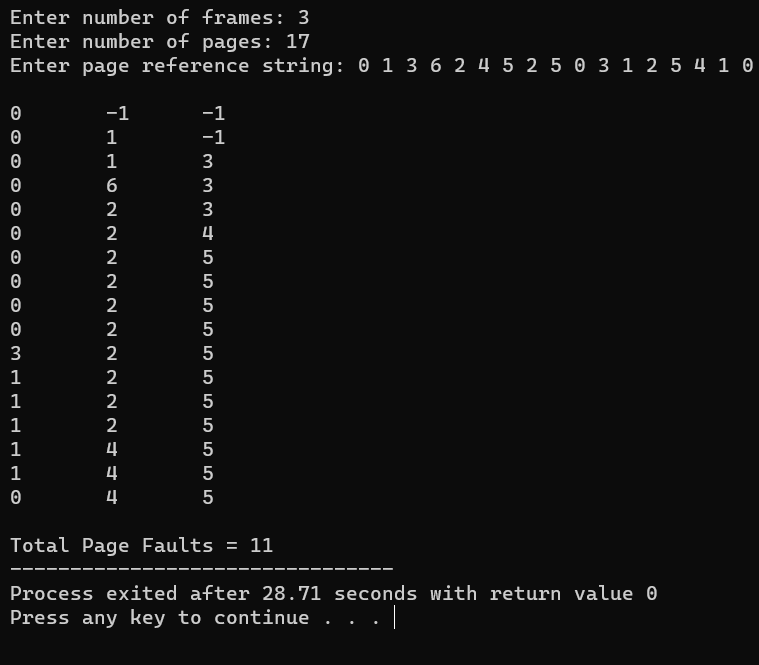
**Output-**











**Shortest Job First(Preemptive) scheduling Algorithm:**

**Input-**

#include <stdio.h>

int main() {

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes\n", limit);

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

printf("\nEnter Arrival Time for Process %d:\t", i + 1);

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

printf("Enter Burst Time for Process %d:\t", i + 1);

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

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

printf("\nProcess No\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

for (time = 0; count != limit; time++) {

smallest = 9;

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

if (arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0) {

smallest = i;

}

}

burst\_time[smallest]--;

if (burst\_time[smallest] == 0) {

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

printf("%d\t\t%d\t\t%d\t\t%lf\t\t%lf\n", smallest + 1, arrival\_time[smallest], temp[smallest],

end - arrival\_time[smallest] - temp[smallest], end - arrival\_time[smallest]);

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

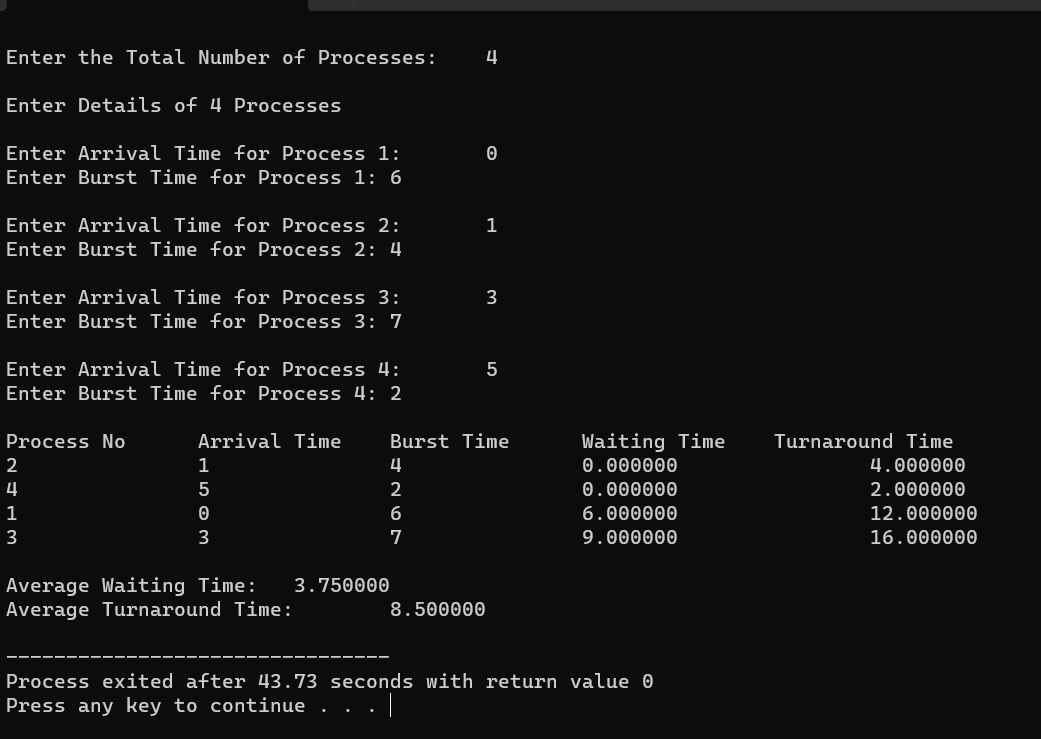
printf("\nAverage Waiting Time:\t%lf\n", average\_waiting\_time);

printf("Average Turnaround Time:\t%lf\n", average\_turnaround\_time);

return 0;

}

**Output-**



**Fork :**

**Input-**

# include <stdio.h>

# include <sys/wait.h>

int main()

{

int p1,p2;

p1= fork();

if(p1==-1)

{

printf("error");

return 0;

}

else

{

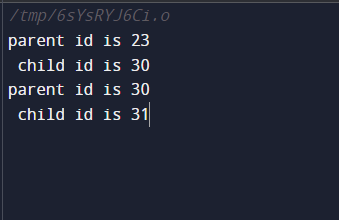
printf("parent id is %d\n" , getppid());

printf(" child id is %d\n" , getpid());

}

}

**Output-**

****

**Inter Process Communication (IPC) :**

**Input-**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

int main(void) {

    int fd1[2], nbytes, fd2[2], a = 0;

    pid\_t tpid;

    char string[80];

    char readbuffer[80];

    char ch = 'a', ch1 = '\n';

    FILE \*fp;

    pipe(fd1); // PIPE CREATED

    pipe(fd2); // PIPE CREATED

    /\* Error in fork \*/

    if ((tpid = fork()) == -1) {

        perror("fork");

        exit(1);

    }

    // Child Process

    if (tpid == 0) {

        close(fd1[1]); /\* closing write end of Pipe 1 \*/

        read(fd1[0], readbuffer, sizeof(readbuffer)); /\* reading filename through Pipe 1 \*/

        printf("\nFilename '%s' is being read by Child Process through Pipe 1...\n", readbuffer);

        fp = fopen(readbuffer, "r");

        close(fd1[0]); /\* closing read end of Pipe 1 \*/

        close(fd2[0]); /\* closing read end of Pipe 2 \*/

        printf("\nContents of %s are being sent to Parent Process through Pipe 2\n", readbuffer);

        while (a != -1) {

            a = fscanf(fp, "%c", &ch);

            write(fd2[1], &ch, sizeof(ch)); /\* writing contents of the file on Pipe 2 \*/

        }

        close(fd2[1]); /\* closing write end of Pipe 2 \*/

        exit(0);

    }

    // Parent process

    else {

        close(fd1[0]); /\* closing read end of Pipe 1 \*/

        printf("IN PARENT PROCESS\n");

        printf("\nEnter the name of the file: ");

        scanf("%s", string);

        printf("Filename is being sent by Parent Process to Child Process through Pipe 1.\n");

        write(fd1[1], string, (strlen(string) + 1)); /\* writing filename on Pipe 1 \*/

        wait(0);

        close(fd1[1]); /\* closing write end of Pipe 1 \*/

        close(fd2[1]); /\* closing write end of Pipe 2 \*/

        printf("\nContents of %s are being received by Parent Process through Pipe 2...\n\n", string);

        printf("IN PARENT PROCESS\n");

        printf("\nReceived Message:\n");

        while (nbytes != 0) {

            printf("%c", ch1);

            nbytes = read(fd2[0], &ch1, sizeof(ch1)); /\* reading contents of the file from Pipe 2 \*/

        }

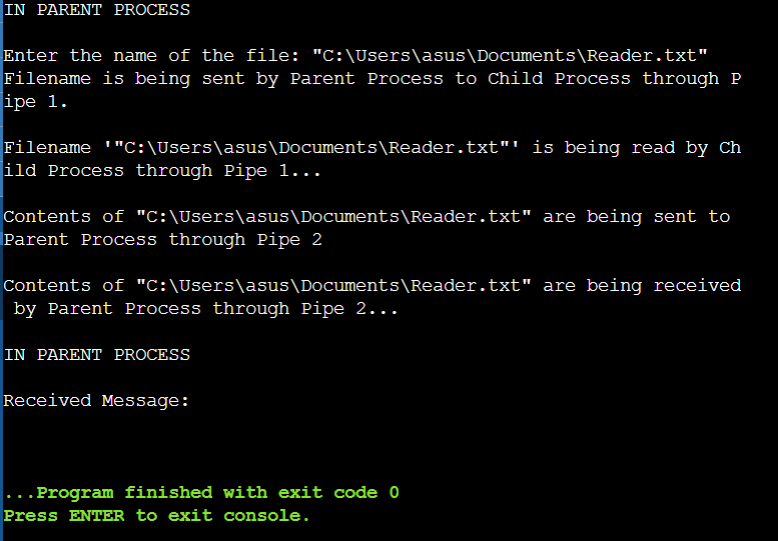
        close(fd2[0]); /\* closing read end of Pipe 2 \*/

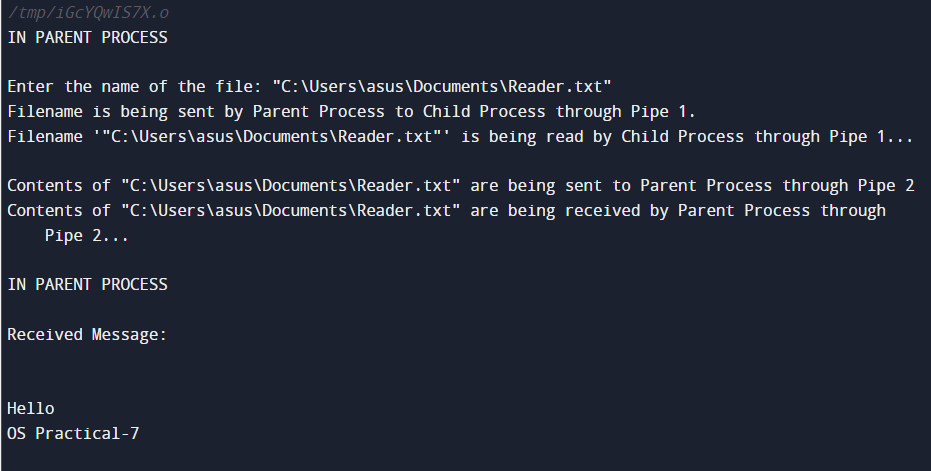
    }

    return 0;

}

**Output-**





**Disk Scheduling Algorithm :**

**Input-**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int choice, track, no\_req, head, head1, distance;

int disc\_reql[100], finish[100];

void menu() {

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

printf("\n1. FCFS\n2. SSTF\n3. SCAN\n4. C-LOOK\n5. Exit");

printf("\n\nEnter your choice: ");

scanf("%d", &choice);

}

void input() {

int i;

printf("Enter the total number of tracks: ");

scanf("%d", &track);

printf("Enter the total number of disc requests: ");

scanf("%d", &no\_req);

printf("\nEnter disc requests in FCFS order:\n");

for (i = 0; i < no\_req; i++) {

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

}

printf("\nEnter current head position: ");

scanf("%d", &head1);

}

void sstf() {

int min, diff;

int pending = no\_req;

int i, distance = 0, index;

head = head1;

for (i = 0; i < no\_req; i++) {

finish[i] = 0;

}

printf("\n%d -> ", head);

while (pending > 0) {

min = 9999;

for (i = 0; i < no\_req; i++) {

diff = abs(head - disc\_reql[i]);

if (finish[i] == 0 && diff < min) {

min = diff;

index = i;

}

}

finish[index] = 1;

distance += min;

head = disc\_reql[index];

pending--;

printf("%d -> ", head);

}

printf("End");

printf("\n\nTotal Distance Traversed: %d", distance);

}

void sort() {

int temp, i, j;

for (i = 0; i < no\_req; i++) {

for (j = 0; j < no\_req; j++) {

if (disc\_reql[i] < disc\_reql[j]) {

temp = disc\_reql[i];

disc\_reql[i] = disc\_reql[j];

disc\_reql[j] = temp;

}

}

}

}

void scan() {

int index, dir, i, distance = 0;

head = head1;

printf("\nEnter the direction of head (1 - Towards higher disc / 0 - Towards lower disc): ");

scanf("%d", &dir);

sort();

printf("\nSorted Disc requests are: ");

for (i = 0; i < no\_req; i++) {

printf("%d ", disc\_reql[i]);

}

for (i = 0; i < no\_req; i++) {

if (head < disc\_reql[i]) {

index = i;

break;

}

}

printf("\nIndex: %d", index);

printf("%d -> ", head);

if (dir == 1) {

sort();

for (i = index; i < no\_req; i++) {

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

distance += abs(head - (track - 1));

head = track - 1;

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

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

} else {

sort();

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

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

distance += abs(head - 0);

head = 0;

for (i = index; i < no\_req; i++) {

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

}

printf("End");

printf("\nTotal Distance Traversed: %d", distance);

}

void clook() {

int index, dir, i, distance = 0;

head = head1;

printf("\nEnter the direction of head (1 - Towards higher disc / 0 - Towards lower disc): ");

scanf("%d", &dir);

sort();

printf("\nSorted Disc requests are: ");

for (i = 0; i < no\_req; i++) {

printf("%d ", disc\_reql[i]);

}

for (i = 0; i < no\_req; i++) {

if (head < disc\_reql[i]) {

index = i;

break;

}

}

printf("\nIndex: %d");

printf("%d -> ", head);

if (dir == 1) {

sort();

for (i = index; i < no\_req; i++) {

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

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

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

} else {

sort();

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

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

for (i = index; i < no\_req; i++) {

distance += abs(head - disc\_reql[i]);

head = disc\_reql[i];

printf("%d -> ", head);

}

}

printf("End");

printf("\nTotal Distance Traversed: %d", distance);

}

int main() {

while (1) {

menu();

switch (choice) {

case 1:

input();

break;

case 2:

sstf();

break;

case 3:

scan();

break;

case 4:

clook();

break;

case 5:

exit(0);

break;

default:

printf("Enter a valid choice.\n");

break;

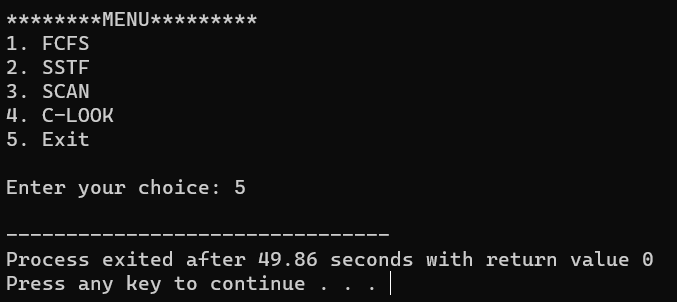
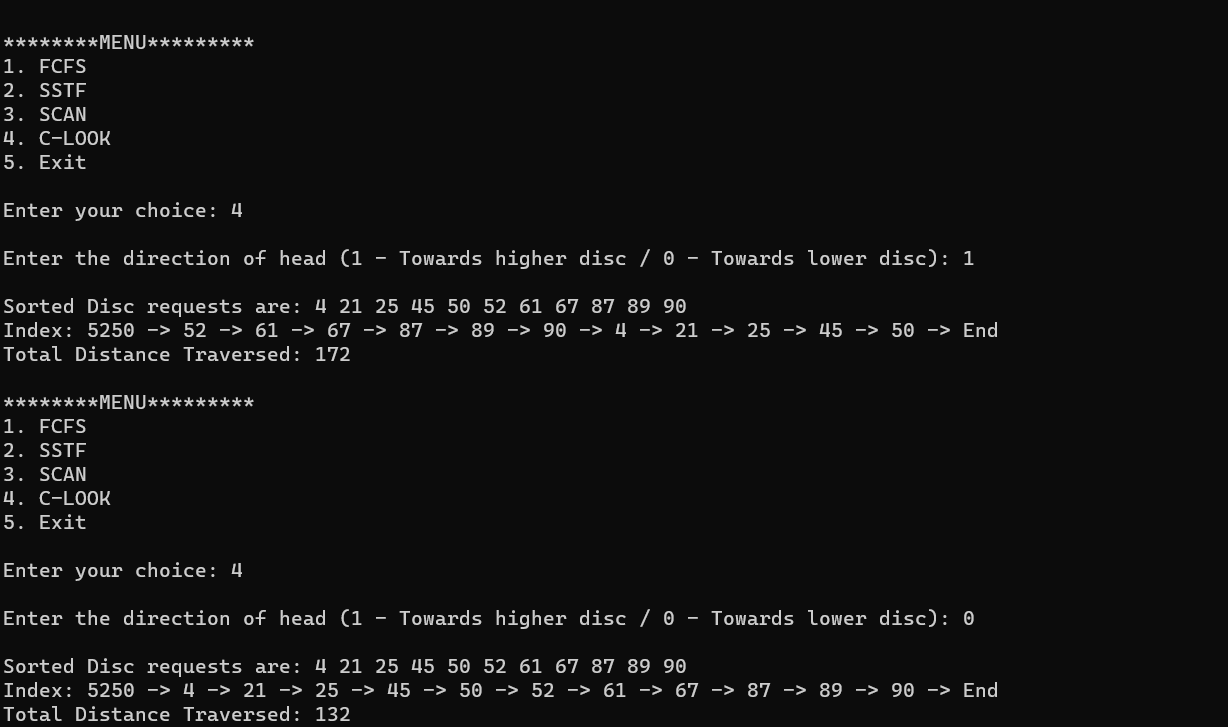
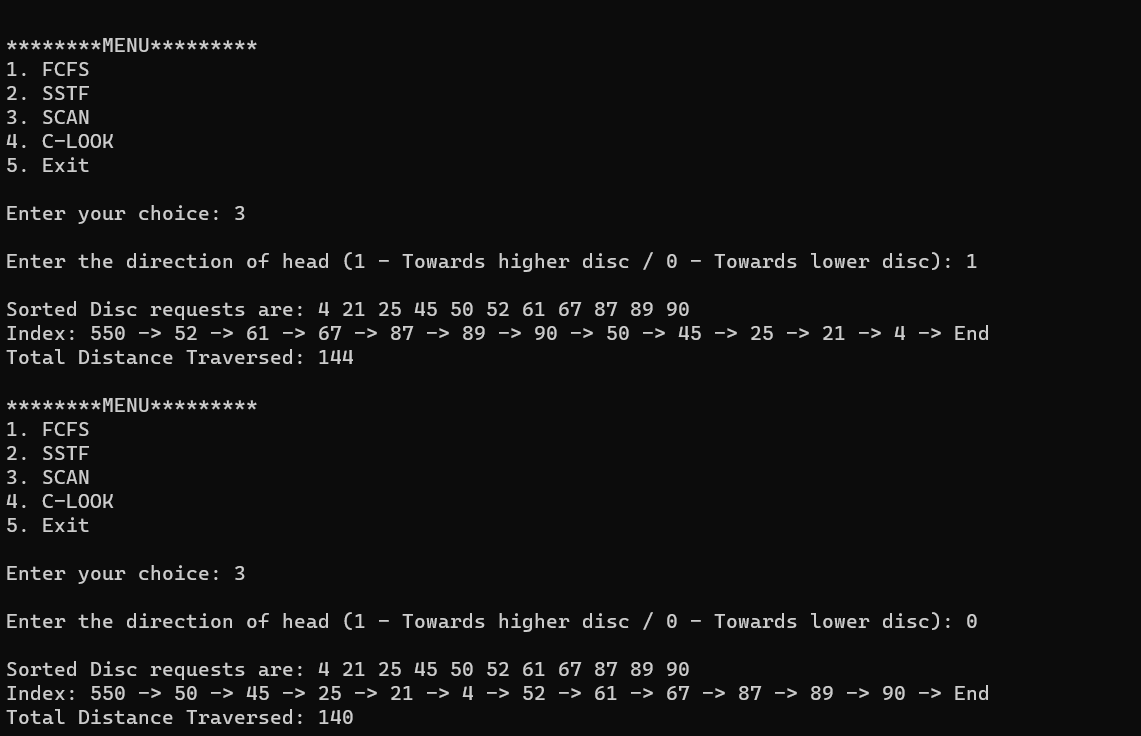
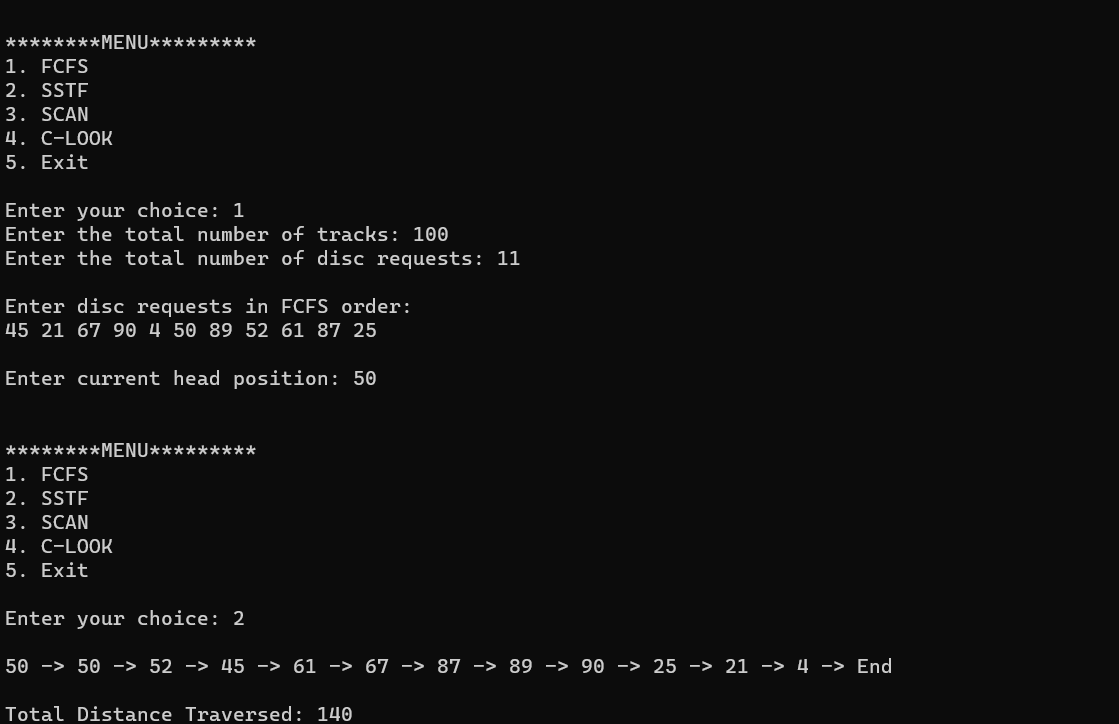
}

}

return 0;

}

**Output-**

****