**1.Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.**

**ALGORITHM:**

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2.

STEP 3: Call fork() system call to create process.

STEP 4: If pid==-1, exit.

STEP 5: Ifpid!=-1 , get the process id using getpid().

STEP 6: Print the process id.

STEP 7:Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<unistd.h>

main()

{

intpid,pid1,pid2;

pid=fork();

if(pid==-1)

{

printf("ERROR IN PROCESS CREATION\n");

exit(1);

}

if(pid!=0)

{

pid1=getpid();

printf("\n the parent process ID is %d\n",pid1);

}

else

{

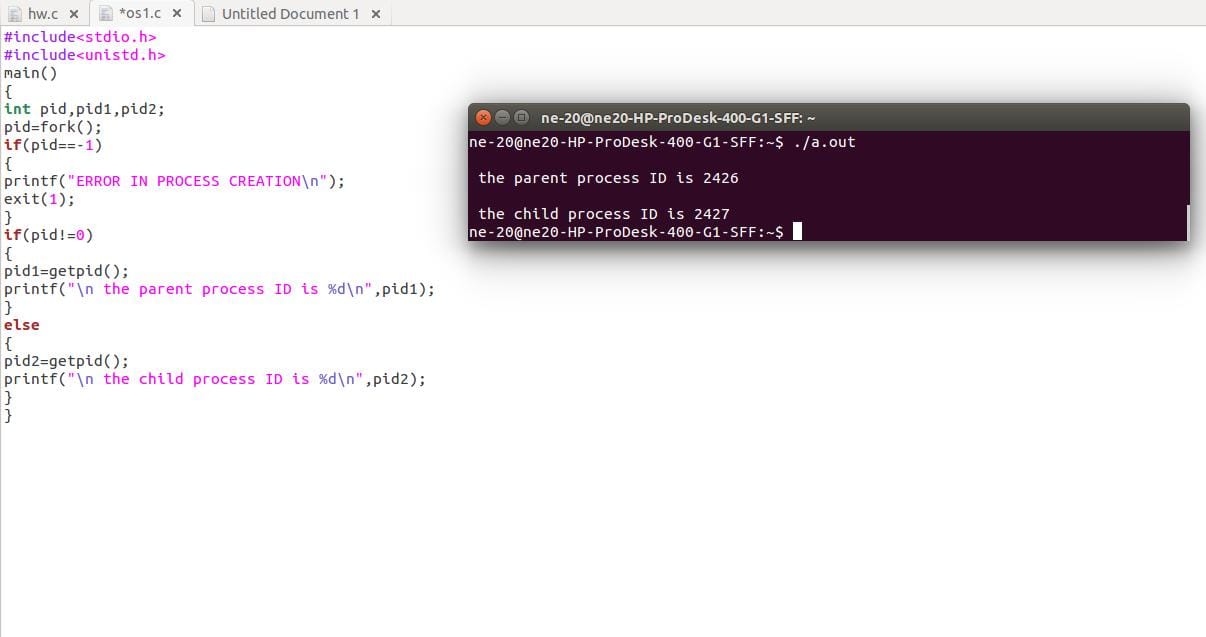
pid2=getpid();

printf("\n the child process ID is %d\n",pid2);

}

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

2.**Identify the system calls to copy the content of one file to another and illustrate the same using a C program.**

**ALGORITHM:**

STEP 1: Start the program.

STEP 2: Declare the variables \*fptr1,\*fptr2

STEP 3: if (fptr1 == NULL),exit

STEP 4: if (fptr1 != NULL),open the file

STEP 5: if (fptr2 == NULL),exit

STEP 6:Print the contents copied to the File.

STEP 7: Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<unistd.h>

intmain()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

printf("\nContents copied to %s", filename);

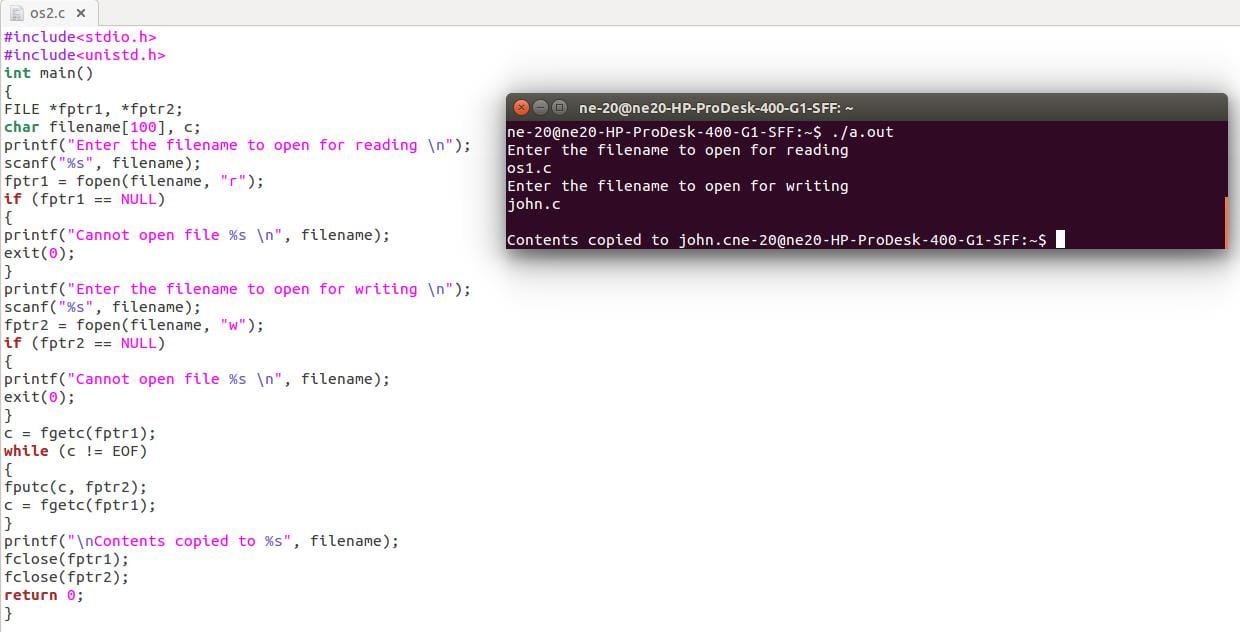
fclose(fptr1);

fclose(fptr2);

return 0;

}

**OUTPUT:**



**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices.**

**ALGORITHM:**

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i, j as integer,totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time

and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of

process.

Step 6: Print total wait time and total turnaround time.

Step 7: Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

structfcfs

{

intpid;

intbtime;

intwtime;

intttime;

}

p[10];

intmain()

{

inti,n;

inttotwttime=0,totttime=0;

printf("\n fcfs scheduling...\n");

printf("enter the no of process");

scanf("%d",&n);

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

{

p[i].pid=1;

printf("\n burst time of the process");

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

}

p[0].wtime=0;

p[0].ttime=p[0].btime;

totttime+=p[i].ttime;

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

{

p[i].wtime=p[i-1].wtime+p[i-1].btime;

p[i].ttime=p[i].wtime+p[i].btime;

totttime+=p[i].ttime;

totwttime+=p[i].wtime;

}

printf("\n total waiting time :%d", totwttime );

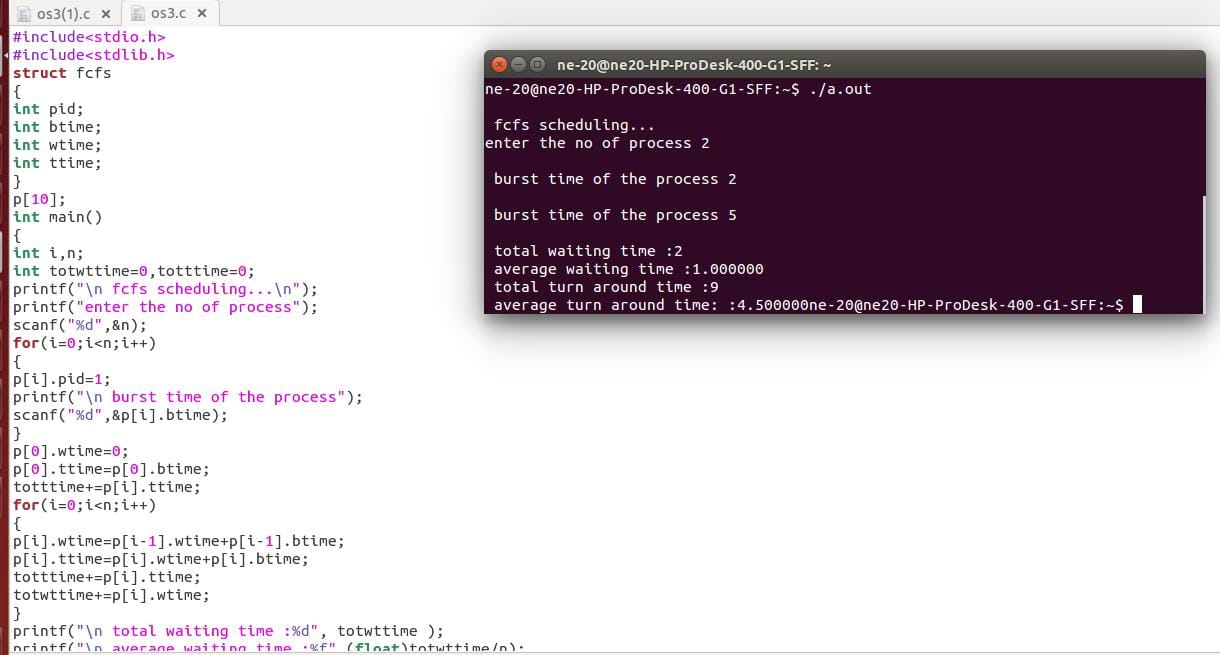
printf("\n average waiting time :%f",(float)totwttime/n);

printf("\n total turn around time :%d",totttime);

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

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.**

**ALGORITHM:**

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time

and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of

process.

Step 6: Print total wait time and total turnaround time.

Step 7: Stop the program.

**PROGRAM:**

#include <stdio.h>

intmain()

{

intA[100][4];

inti, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of process: ");

scanf("%d", &n);

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

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

{

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

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

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

{

index = i;

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

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

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

A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("P BT WT TAT\n");

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

{

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],A[i][1], A[i][2], A[i][3]);

}

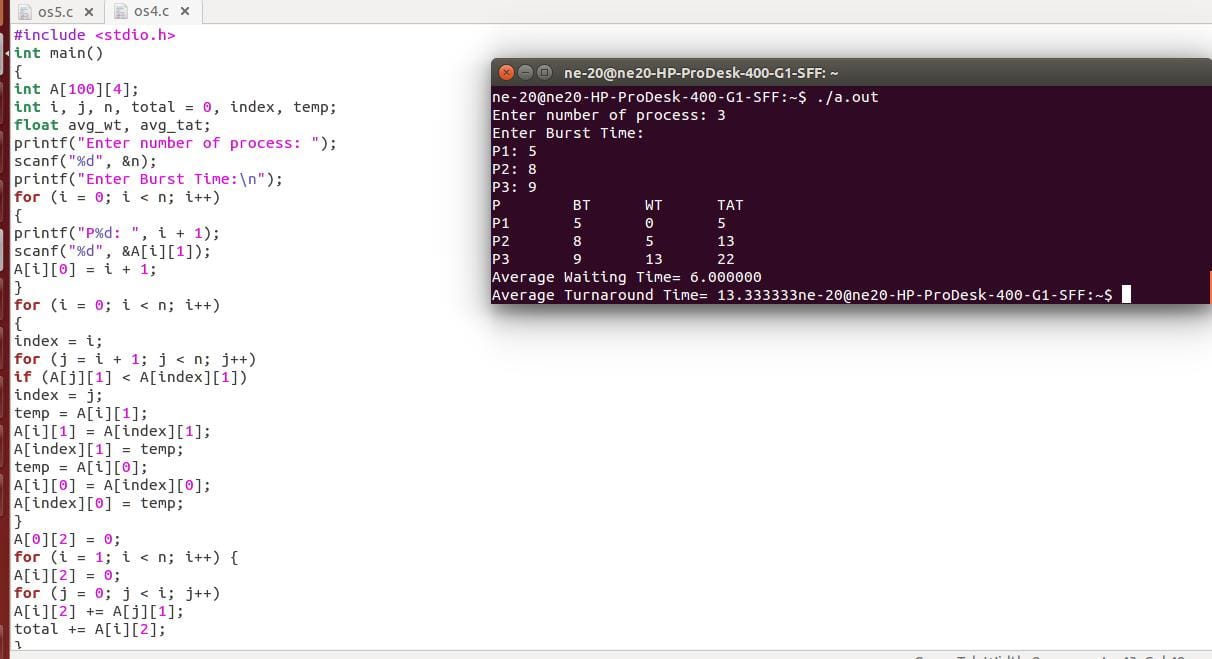
avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt);

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

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**5. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.**

**ALGORITHM:**

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: Compare each and every process using the banker’s algorithm.

Step-5: If the process is in safe state then it is a not a deadlock process otherwise it is a

deadlock process

Step-6: produce the result of state of process

Step-7: Stop the program

**PROGRAM:**

**#include <stdio.h>**

**intmain()**

**{**

**int n, m, i, j, k;**

**n = 5;**

**m = 3;**

**intalloc[5][3] = { { 0, 1, 0 }, { 2, 0, 0 },{ 3, 0, 2 }, { 2, 1, 1 }, { 0, 0, 2 } };**

**intmax[5][3] = { { 7, 5, 3 },{ 3, 2, 2 }, { 9, 0, 2 },{ 2, 2, 2 }, { 4, 3, 3 } };**

**intavail[3] = { 3, 3, 2 };**

**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(inti=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]);**

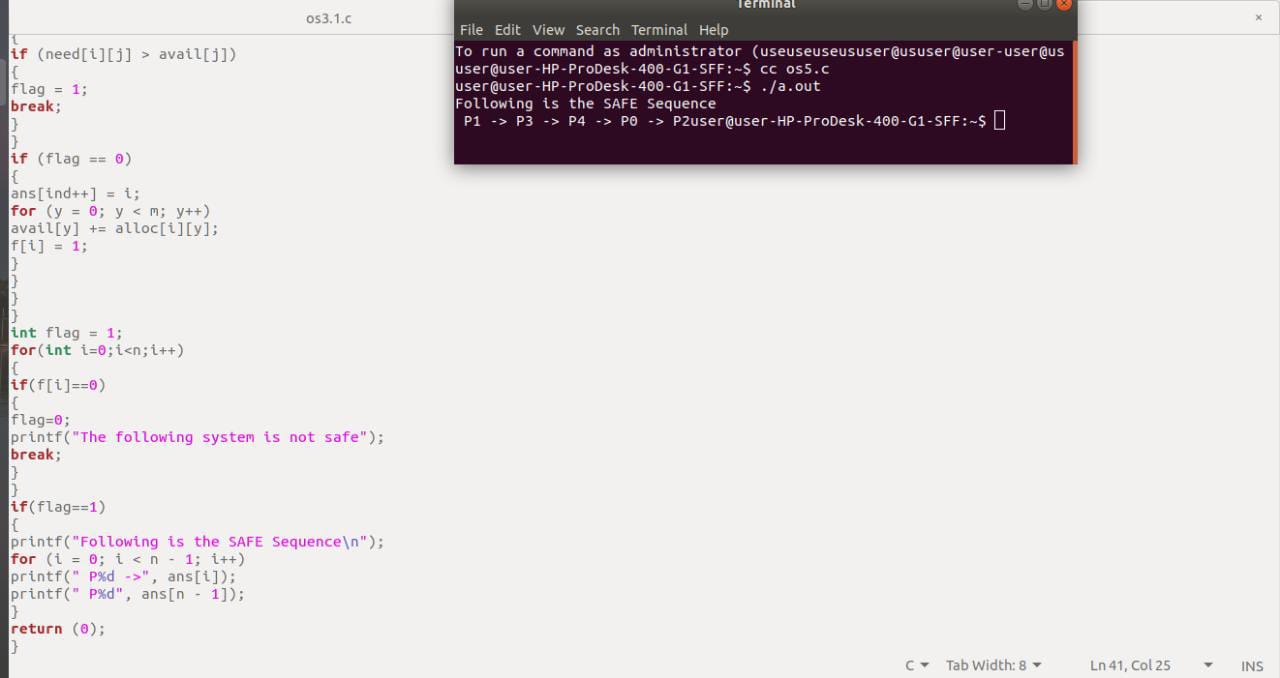
**printf(" P%d", ans[n - 1]);**

**}**

**return (0);**

**}**

**OUTPUT:**



**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**6.Construct a C program to simulate producer-consumer problem using semaphores.**

**ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Initialize the buffer size and get maximum item you want to produce.

Step 4: Get the option, which you want to do either producer, consumer or exit from the

operation.

Step 5: If you select the producer, check the buffer size if it is full the producer should not

produce the item or otherwise produce the item and increase the value buffer size.

Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not

consume the item or otherwise consume the item and decrease the value of buffer size.

Step 7: If you select exit come out of the program.

Step 8: Stop the program

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

intmutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()

{

--mutex;

++full;

--empty;

x++;

printf("\nProducerproduces""item %d",x);

++mutex;

}

void consumer()

{

--mutex;

--full;

++empty;

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

x--;

++mutex;

}

intmain()

{

int n, i;

printf("\n1. Press 1 for Producer""\n2. Press 2 for Consumer""\n3. Press 3 for Exit");

#pragma omp critical

for (i = 1; i> 0; i++)

{

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n)

{

case 1:

if ((mutex == 1)

&& (empty != 0))

{

producer();

}

else {

printf("Buffer is full!");

}

break;

case 2:

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

{

consumer();

}

else {

printf("Buffer is empty!");

}

break;

case 3:

exit(0);

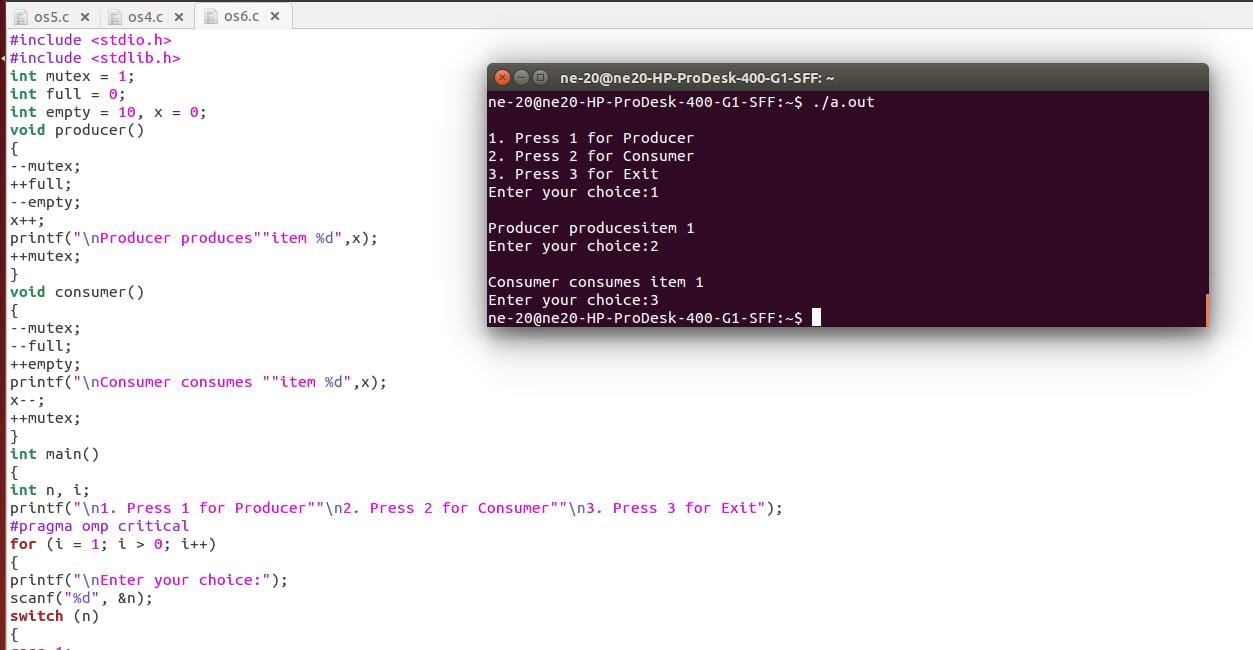
break;

}

}

}

**OUTPUT:**



**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**7. Construct a C program to simulate the First in First Out paging technique of**

**memory management.**

**ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the necessary variables.

Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and

when the page must be replaced the oldest page is chosen.

Step 6: When a page is brought into memory, it is inserted at the tail of the queue.

Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases.

Step 9: Print the total number of page faults.

Step 10: Stop the program.

**PROGRAM:**

#include<stdio.h>

intmain()

{

inti=0,j=0,k=0,i1=0,m,n,rs[30],flag=1,p[30];

system("clear");

printf("FIFO page replacement algorithm....\n");

printf("enter the no. of frames:");

scanf("%d",&n);

printf("enter the reference string:");

while(1)

{

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

if(rs[i]==0)

break;

i++;

}

m=i;

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

p[j]=0;

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

{

flag=1;

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

if(p[j]==rs[i]) {

printf("data already in page....\n");

flag=0;

break; }

if(flag==1)

{

p[i1]=rs[i];

i1++;

k++;

if(i1==n)

i1=0;

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

{

printf("\n page %d:%d",j+1,p[j]);

if(p[j]==rs[i])

printf("\*");

}

printf("\n\n");

}

}

printf("total no page faults=%d",k);

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**8. Construct a C program to simulate the Least Recently Used paging technique of**

**memory management.**

**ALGORITHM:**

Step 1: Start the process

Step 2: Declare the size

Step 3: Get the number of pages to be inserted

Step 4: Get the value

Step 5: Declare counter and stack

Step 6: Select the least recently used page by counter value

Step 7: Stack them according the selection.

Step 8: Display the values

Step 9: Stop the process

**PROGRAM:**

#include<stdio.h>

int main()

{

int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];

printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:");

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

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

printf("Enter no of frames:");

scanf("%d",&f);

q[k]=p[k];

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

c++;

k++;

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

{

c1=0;

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

{

if(p[i]!=q[j])

c1++;

}

if(c1==f)

{

c++;

if(k<f)

{

q[k]=p[i];

k++;

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

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

printf("\n");

}

else

{

for(r=0;r<f;r++)

{

c2[r]=0;

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

{

if(q[r]!=p[j])

c2[r]++;

else

break;

}

}

for(r=0;r<f;r++)

b[r]=c2[r];

for(r=0;r<f;r++)

{

for(j=r;j<f;j++)

{

if(b[r]<b[j])

{

t=b[r];

b[r]=b[j];

b[j]=t;

}

}

}

for(r=0;r<f;r++)

{

if(c2[r]==b[0])

q[r]=p[i];

printf("\t%d",q[r]);

}

printf("\n");

}

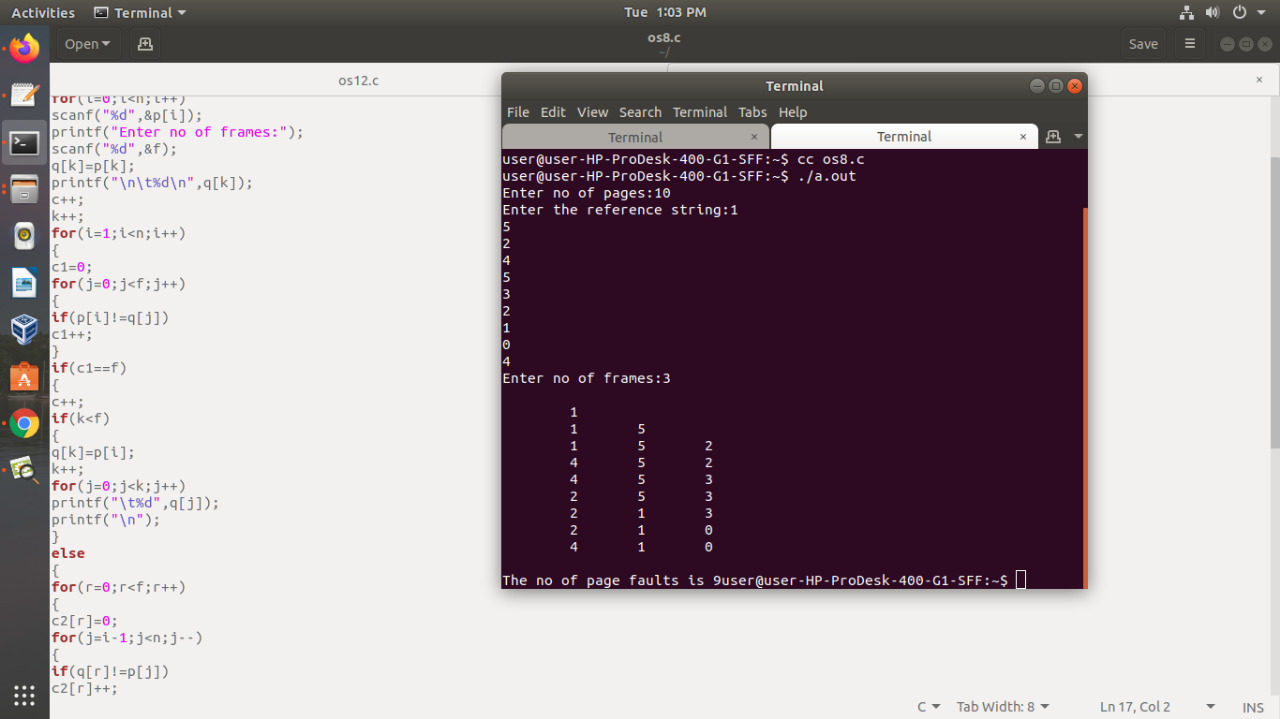
}

}

printf("\nThe no of page faults is %d",c);

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**9. Construct a C program to simulate the optimal paging technique of memory**

**management.**

**ALGORITHM:**

Step 1: Start the process

Step 2:Read the number of pages,Reference String.

Step 3: Get the number of pages to be inserted

Step 4: Get the value

Step 5: Declare counter and stack

Step 6: Select the Optimal page replacement by counter value

Step 7: Stack them according the selection.

Step 8: Display the values

Step 9: Stop the process.

**PROGRAM:**

#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]);

}

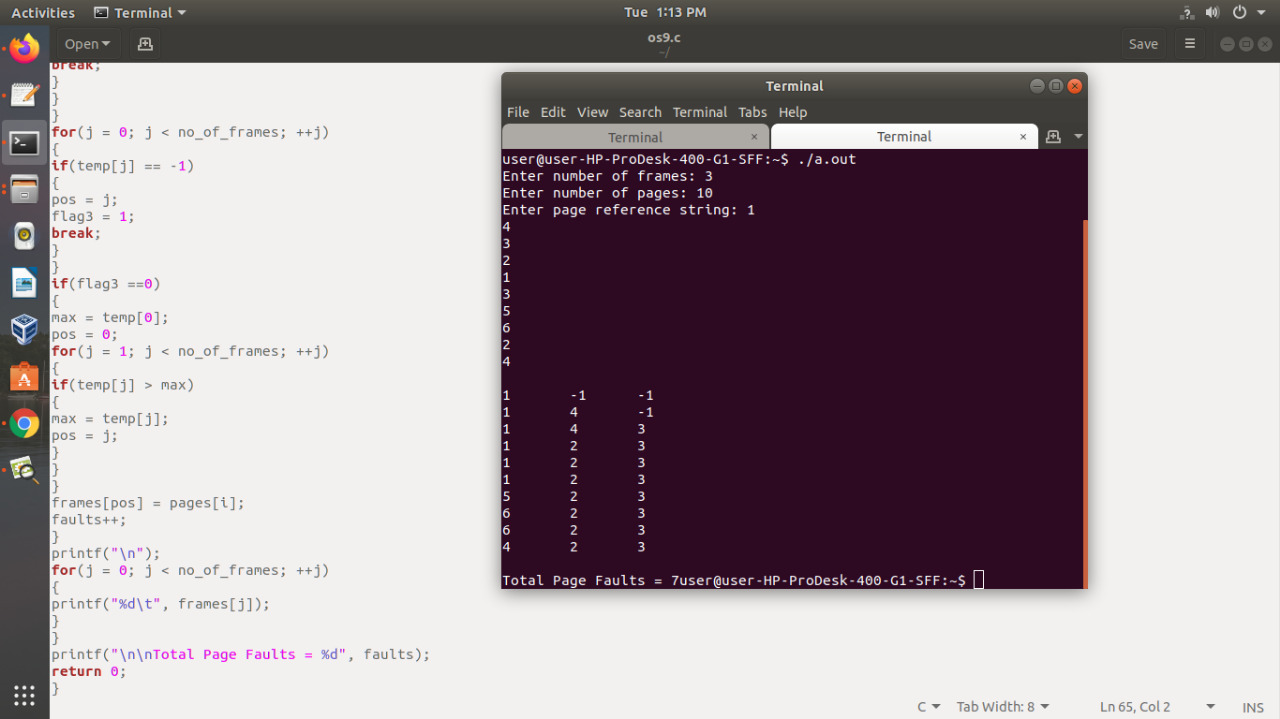
}

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

return 0;

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**10. Consider a file system where the records of the file are stored one after another both**

**physically and logically. A record of the file can only be accessed by reading all the**

**previous records. Design a C program to simulate the file allocation strategy.**

**ALGORITHM:**

Step-1: Start the program.

Step-2: Get the number of records user want to store in the system.

Step-3: Using Standard Library function open the file to write the data into the file.

Step-4: Store the entered information in the system.

Step-5: Using do..While statement and switch case to create the options such as

1-DISPLAY, 2.SEARCH, 3.EXIT.

Step-6: Close the file using fclose() function.

Step-7: Process it and display the result.

Step-8: Stop the program.

**PROGRAM:**

**#include <stdio.h>**

**typedef struct**

**{**

**int usn;**

**char name[25];**

**int m1,m2,m3;**

**}**

**STD;**

**STD s;**

**void display(FILE \*);**

**int search(FILE \*,int);**

**void main()**

**{**

**int i,n,usn\_key,opn;**

**FILE \*fp;**

**printf(" How many Records ? ");**

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

**fp=fopen("stud.dat","w");**

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

**{**

**printf("Read the Info for Student: %d (usn,name,m1,m2,m3) \n",i+1);**

**scanf("%d%s%d%d%d",&s.usn,s.name,&s.m1,&s.m2,&s.m3);**

**fwrite(&s,sizeof(s),1,fp);**

**}**

**fclose(fp);**

**fp=fopen("stud.dat","r");**

**do**

**{**

**printf("Press 1- Display\t 2- Search\t 3- Exit\t Your Option?");**

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

**switch(opn)**

**{**

**case 1: printf("\n Student Records in the File \n");**

**display(fp);**

**break;**

**case 2: printf(" Read the USN of the student to be searched ?");**

**scanf("%d",&usn\_key);**

**if(search(fp,usn\_key))**

**{**

**printf("Success ! Record found in the file\n");**

**printf("%d\t%s\t%d\t%d\t%d\n",s.usn,s.name,s.m1,s.m2,s.m3);**

**}**

**else**

**printf(" Failure!! Record with USN %d not found\n",usn\_key);**

**break;**

**case 3: printf(" Exit!! Press a key . . .");**

**break;**

**default: printf(" Invalid Option!!! Try again !!!\n");**

**break;**

**}**

**}**

**while(opn != 3);**

**fclose(fp);**

**}**

**/\* End of main() \*/**

**void display(FILE \*fp)**

**{**

**rewind(fp);**

**while(fread(&s,sizeof(s),1,fp))**

**printf("%d\t%s\t%d\t%d\t%d\n",s.usn,s.name,s.m1,s.m2,s.m3);**

**}**

**int search(FILE \*fp, int usn\_key)**

**{**

**rewind(fp);**

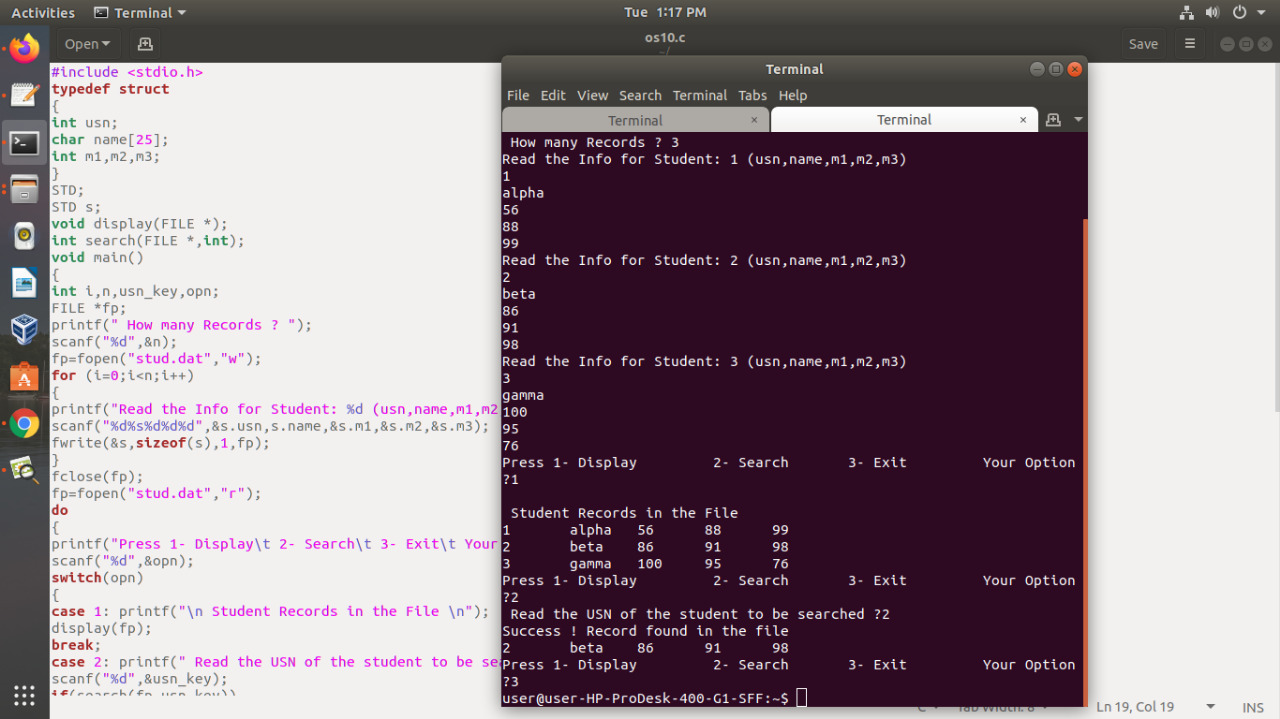
**while(fread(&s,sizeof(s),1,fp))**

**if( s.usn == usn\_key) return 1;**

**return 0;**

**}**

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**11. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.**

**ALGORITHM:**

Step-1: Start the program.

Step-2: Get the number of records user want to store in the system.

Step-3: Using Standard Library function open the file to write the data into the file.

Step-4: Store the entered information in the system.

Step-5: Using do..While statement and switch case to create the options such as

1-DISPLAY, 2.SEARCH, 3.EXIT.

Step-6: Close the file using fclose() function.

Step-7: Process it and display the result.

Step-8: Stop the program.

**PROGRAM:**

#include<stdio.h>

intmain()

{

char name[10][30];

int start[10],length[10],num;

printf("Enter the number of files to be allocated\n");

scanf("%d",&num);

int count=0,k,j;

for(inti=0;i<num;i++)

{

printf("Enter the name of the file %d\n",i+1);

scanf("%s",&name[i][0]);

printf("Enter the start block of the file %d\n",i+1);

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

printf("Enter the length of the file %d\n",i+1);

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

for(j=0,k=1;j<num&& k<num;j++,k++)

{

if(start[j+1]<=start[j] || start[j+1]>=length[j])

{

}

else

{

count++;

}

}

if(count==1)

{

printf("%s cannot be allocated disk space\n",name[i]);

}

}

printf("File Allocation Table\n");

printf("%s%40s%40s\n","FileName","StartBlock","Length");

printf("%s%50d%50d\n",name[0],start[0],length[0]);

for(inti=0,j=1;i<num&& j<num;i++,j++)

{

if(start[i+1]<=start[i] || start[i+1]>=length[i])

{

printf("%s%50d%50d\n",name[j],start[j],length[j]);

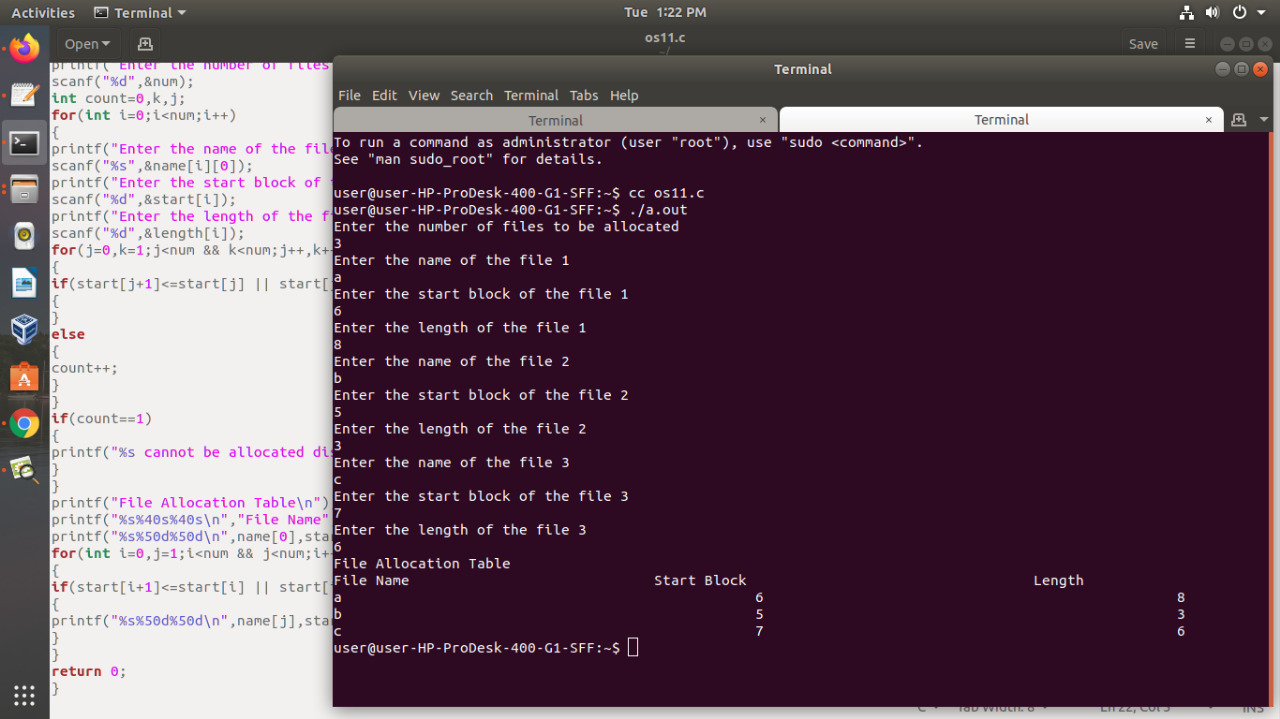
}

}

return 0;

}

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**12. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.**

**ALGORITHM:**

Step-1: Start the program.

Step-2: Get the number of records user want to store in the system.

Step-3: Using Standard Library function open the file to write the data into the file.

Step-4: Store the entered information in the system.

Step-5: Using do..While statement and switch case to create the options such as

1-DISPLAY, 2. SEARCH, 3. EXIT.

Step-6: Close the file using fclose() function.

Step-7: Process it and display the result.

Step-8: Stop the program.

**PROGRAMM:**

**#include<stdio.h>**

**#include<stdlib.h>**

**intmain()**

**{**

**intRQ[100],i,n,TotalHeadMoment=0,initial;**

**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);**

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

**{**

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

**initial=RQ[i];**

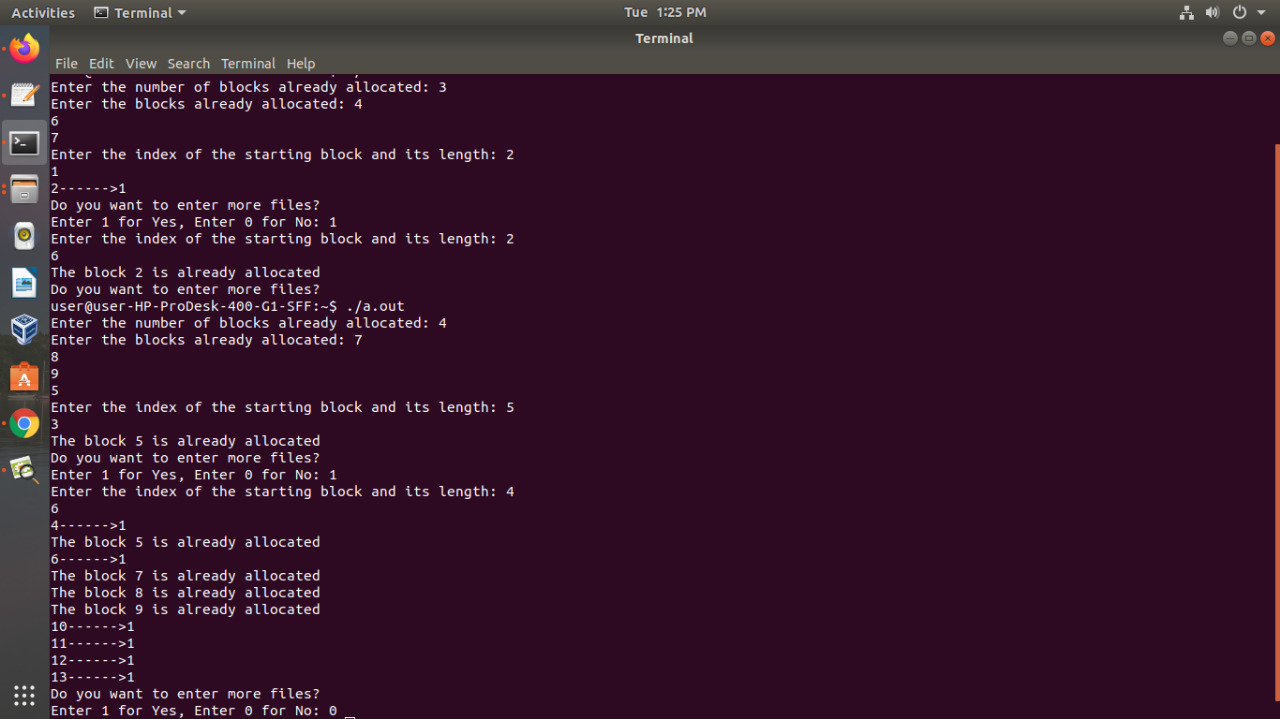
**}**

**printf("Total head moment is %d",TotalHeadMoment);**

**return 0;**

**}**

**OUTPUT:**

****

**RESULT:**

Thus, the C-Program has been verified and executed successfully.

**13. Construct a C program to simulate the First Come First Served disk scheduling algorithm.**

**ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the necessary variables.

Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and

when the page must be replaced the oldest page is chosen.

Step 6: When a page is brought into memory, it is inserted at the tail of the queue.

Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases.

Step 9: Print the total number of page faults.

Step 10: Stop the program.

**PROGRAM:**

**#include<stdio.h>**

**#include<stdlib.h>**

**intmain()**

**{**

**intRQ[100],i,n,TotalHeadMoment=0,initial;**

**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);**

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

**{**

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

**initial=RQ[i];**

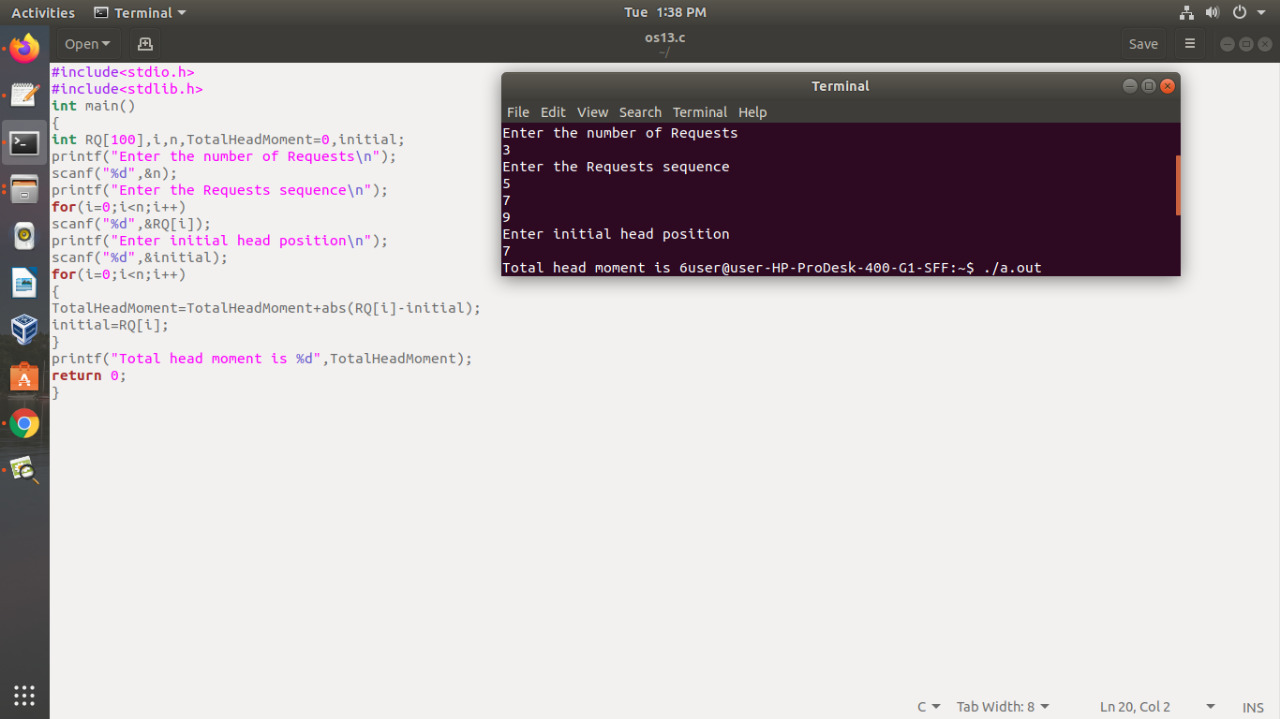
**}**

**printf("Total head moment is %d",TotalHeadMoment);**

**return 0;**

**}**

**OUTPUT:**



**RESULT:**

Thus, the C-Program has been verified and executed successfully.