|  |  |
| --- | --- |
| **EX:NO:4A** | **USE OF PROCESS OF SYSTEM CALLS** |
| **DATE:** |

**AIM:**

To illustrate the use of process of system calls using C program

**ALGORITHM:**

1. Start
2. Declare pid
3. Create a new process using fork()
4. Perform if pid<0

Display fork cannot be created

else if pid==0

Display parent by getppid and child by getpid

Else

Display parent by getpid and grandparent getppid

1. Stop

**INFERENCE:**

Processes use the fork() system call to create a program that is a copy of themselves.

This is one of the major methods of process creation in operating systems.

When a parent process creates a child process and the execution of the parent process is suspended until the child process executes.

The process which is called fork() call is the parent process and the process which is created newly is the child process.

The child process will be exactly the same as the parent .

**PROGRAM:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

void main()

{

int pid;

pid=fork();

if (pid < 0)

{

printf("The fork cannot be created");

exit(0);

}

else

if (pid==0)

{

execlp("/bin/ps","ps");

printf("\n The process id of the child: %d", getpid());

printf("\n The process id of the parent: %d", getppid());

}

else{

printf("\n The process id of the parent: %d", getpid());

printf("\n The process id of the grandparent: %d", getppid());

}

}

**OUTPUT:**



**RESULT:**

Thus the use of process system call using c program has been illustrated and executed

successfully.

|  |  |
| --- | --- |
| **EX:NO:4B** | **USE OF GETPID** |
| **DATE:** |

**AIM:**

To illustrate the use of process of system calls using C program

**ALGORITHM:**

1. Start
2. Declare pid
3. Create a new process using fork()
4. Perform if pid<0

Display fork cannot be created

else if pid==0

Display parent by getppid and child by getpid

Else

Display parent by getpid and grandparent getppid

1. Stop

**INFERENCE:**

Processes use the fork() system call to create a program that is a copy of themselves.

This is one of the major methods of process creation in operating systems.

When a parent process creates a child process and the execution of the parent process is suspended until the child process executes.

The process which is called fork() call is the parent process and the process which is created newly is the child process.

The child process will be exactly the same as the parent .

**PROGRAM:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

void main()

{

int pid;

pid=fork();

if (pid < 0)

{

printf("The fork cannot be created");

exit(0);

}

else

if (pid==0)

{

execlp("/bin/ps","ps");

printf("\n The process id of the child: %d", getpid());

printf("\n The process id of the parent: %d", getppid());

}

else{

printf("\n The process id of the parent: %d", getpid());

printf("\n The process id of the grandparent: %d", getppid());

}

}

**OUTPUT:**



**RESULT:**

Thus the use of process system call using c program has been illustrated and executed

successfully.

|  |  |
| --- | --- |
| **EX NO: 5** | **FCFS CPU SCHEDULING** |
| **DATE:** |

**AIM:**

To illustrate FCFS CPU Scheduling using C Program.

**ALGORITHM:**

1. Start
2. Declare the variables
3. Input the number of process from user
4. Using for loop input the arrival time and burst time for each process
5. Using for loop, for each process

Calculate turn around time by

TAT = completion - arrival

Calculate waiting time by

WT = Turn around time – burst time

6. Now calculate average turn around time and waiting time

7. Display every calculated values

8. Stop.

**PROGRAM:**

#include<stdio.h>

int main(){

 int bt[10]={0},wt[10]={0},ct[10]={0};

float at[10]={0},tat[10]={0};

int n,sum=0;

float totalTAT=0,totalWT=0;

 printf("\_\_\_FCFS CPU SCHEDULING\_\_\_");

printf("\n\nEnter number of processes ");

scanf("%d",&n);

printf("Enter arrival time and burst time for each process\n\n");

 for(int i=0;i<n;i++)

{

printf("Arrival time of process[%d] ",i+1);

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

printf("Burst time of process[%d] ",i+1);

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

printf("\n");

}

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

{

sum+=bt[j];

ct[j]+=sum;

}

 for(int k=0;k<n;k++)

{

tat[k]=ct[k]-at[k];

totalTAT+=tat[k];

}

 for(int k=0;k<n;k++)

{

wt[k]=tat[k]-bt[k];

totalWT+=wt[k];

}

printf("Solution: \n\n");

printf("P\t AT\t BT\t CT\t TAT\t WT\t\n\n");

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

{

printf("P%d\t %.2f\t %d\t %d\t %.2f\t %d\n",

i+1,at[i],bt[i],ct[i],tat[i],wt[i]);

}

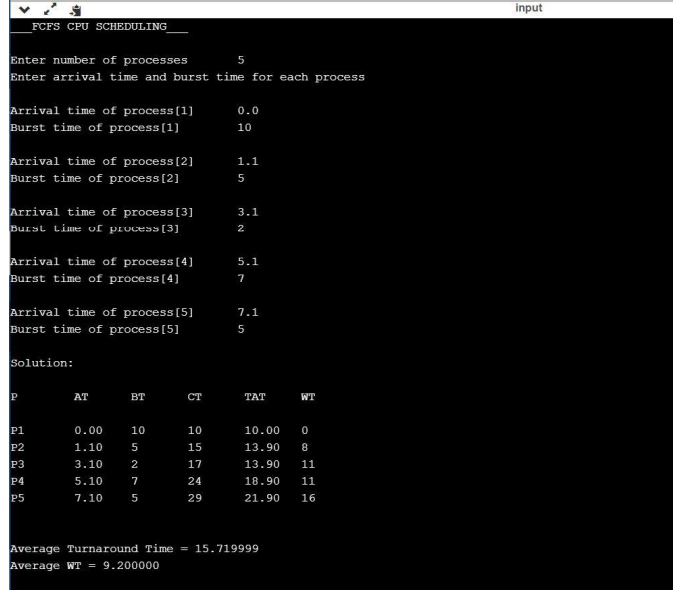
printf("\n\nAverage Turnaround Time = %f\n",totalTAT/n);

printf("Average WT = %f\n\n",totalWT/n);

return 0;

}

**OUTPUT:**



**RESULT:**

Thus the FCFS CPU scheduling using c program has been illustrated and executed successfully.

|  |  |
| --- | --- |
| **EX.NO:6** | **PRODUCER CONSUMER PROBLEM** |
| **DATE:** |

**AIM:**

To illustrate interprocess communication producer consumer problem using c program.

**ALGORITHM:**

Start

Declare the variables

Using switch case get the choice from user

Case 1:

Call the producer function

Get the data from the user

Add it to buffer front

Front =(front+1)%5

Increment count

If (Consumersleep==1 and count==1)

Display consumer is now ready

Else

Display Buffer is full

Producersleep is one

Case 2:

Call the consumer function

Get the item from user

Buffer [rear]= “ “

Now display the consumed items

Tear + (Tear+1) %5

Decrement count

If producersleep ==1 and count==4

Display Producer is now ready

Else

Display Buffer is empty

Consumer sleep is,

Case 3:

Call view function

Using for loop

Display buffer data

Case 4:

Exit

Stop.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#define N 5;

int count=0;

int front=0;int rear=0;

char buffer[7];

int prodsleep=0;int consleep=0;

void producer(void){

char item;

if (count<5){

printf("Enter data :");

scanf(" %c",&item);

buffer [front]=item;

front = (front+1)%5;

count++;

if(consleep==1 && count==1){

printf("\n Consumer is now ready ");

}

}

else{

printf("\n Buffer is full...");

prodsleep=1;

}

}

void consumer(void){

char item;

if (count>0){

item = buffer[rear];

buffer[rear]=' ';

printf("\n C: %c",item);

rear=(rear+1)%5;

count--;

if(prodsleep==1 && count==4)

{

printf("\n Producer is now ready");

}

}

else{

printf("\n Buffer is empty...");

consleep=1;

}

}

void view(void)

{

int i;

printf("\n Data of buffer: ");

for(i=0;i<5;i++){

printf("- %c ",buffer[i]);

}

}

void main(){

int i,choice,flag=0;

printf("\_\_\_PRODUCER CONSUMER\_\_\_\n");

printf("\n 1: Produce item ");

printf("\n 2: Consume item ");

printf("\n 3: To view buffer ");

printf("\n 4: Exit");

do{

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

scanf("%d",&choice);

switch(choice){

case 1:producer();

break;

case 2:consumer();

break;

case 3:view();

break;

case 4:flag=1;

break;

default:printf("\n Enter correct choice");

break;

}

}

while(flag==0);

}

**OUTPUT:**



**RESULT:**

Thus the inter process communication producer consumer problem has been illustrated and executed successfullu

|  |  |
| --- | --- |
| **EX.NO:7**  **DATE:** | **C-SIMULATION OF VI,CAT AND CP** |

**AIM:**

To illustrate the simulation of vi,cat and cp using c program.

**ALGORITHM:**

1. START
2. Declare the required variables.
3. Input the choice from the user.
4. Perform the operation using switch case.
5. Case 1(vi)

Get the file from the user

Using while loop

While(a!=’\*’){

Fput c(a,file)

a=getchar()

}

Fclose(f1)

1. Case 2(cat)

Get the file name

File must be in read mode

if (file 1==’\0’)

DISPLAY FILE IS EMPTY

Else

a=fgetc(file 1)

using while loop perform

while(a!=EOF){

DSPLAY a

a=fgetc(file 1)

}

fclose(f1)

1. Case 3(cp)

Get the file name for the source and destination

Source file should be in read mode and destination in write mode

if (file1 ==’\0’ && file2==’\0’)

DISPLAY File is empty

else

b=fgetc(file 1)

using while loop perform

while (b!=EOF){

fputc(b,file 1);

b=getc(file 1)

}

fclose (f1)

1. By default if no cases matches

DISPLAY Enter valid option

1. STOP

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

 int ch;

 char a,b,file1[10],file2[10];

 FILE \*f1,\*f2;

 printf("MENU");

 printf("\n1.Press 1 for vi \n2.Press 2 for cat \n3.Press 3 for cp ");

 printf("\nEnter your choice: ");

 scanf("%d",&ch);

 switch(ch)

 {

 case 1:

 printf("\n\_\_\_vi command\_\_\_");

 printf("\nEnter the file name: ");

 scanf("%s",file1);

 f1=fopen(file1,"w");

 a=getchar();

 while(a!='\*')

 {

 fputc(a,f1);

 a=getchar();

 }

 fclose(f1);

 break;

 case 2:

 printf("\n\_\_\_\_cat command\_\_\_");

 printf("\nEnter the file name: ");

 scanf("%s",file1);

 f1=fopen(file1,"r");

 if(f1=='\0')

 {

 printf("\n File is empty");

 exit(0);

 }

 else

 {

 a=fgetc(f1);

 while(a!=EOF)

 {

 printf("%c",a);

 a=fgetc(f1);

 }

 }

 fclose(f1);

 break;

 case 3:

 printf("\n\_\_\_cp command\_\_\_");

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

 scanf("%s",file1);

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

 scanf("%s",file2);

 f1=fopen(file1,"r");

 f2=fopen(file2,"w");

 if(f1=='\0' && f2=='\0')

 {

 printf("\nFile is empty");

 }

 else

 {

 b=fgetc(f1);

 while(b!=EOF)

 {

 fputc(b,f2);

 b=getc(f1);

 }

 }

 fclose(f1);

 fclose(f2);

 printf("\n File is copied successfully");

 break;

 default:

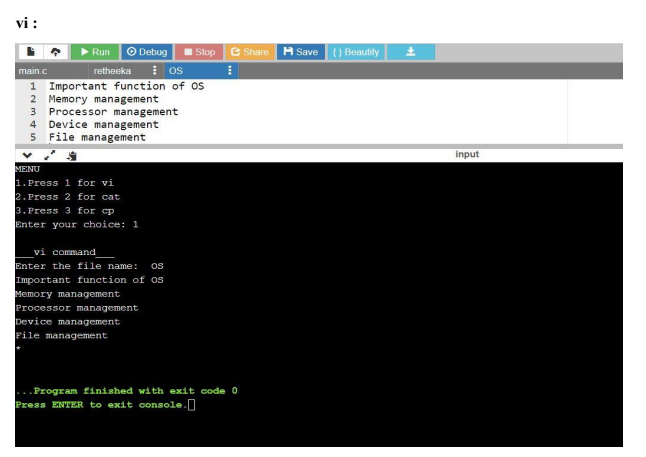
 printf("\nEnter a valid option");

 }

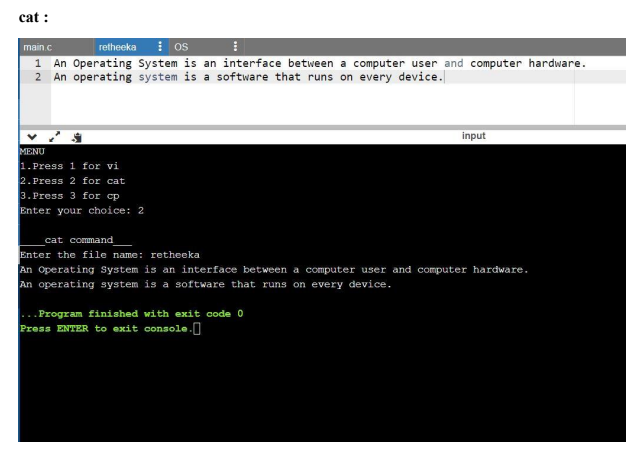
}

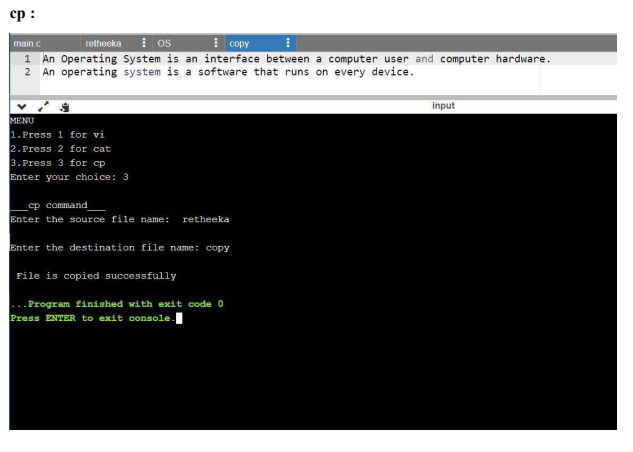
**OUTPUT:**

**vi:**



**cat:**





**RESULT:**

Thus the simulation of vi,cat and cp using c program has been illustrated and executed  successfully.

|  |  |
| --- | --- |
| **EX.NO:08-A** | **USE OF FILE SYSTEM CALLS** |
| **DATE:** |

**AIM:**

To write a program to establish the concept of file system call and its uses.

**ALGORITHM:**

1. Start
2. Import the necessary reader file
3. Declare necessary variables
4. Declare a static char message
5. Assign a string ‘HELLOWORLD’ to message
6. Declare char buffer
7. Open the file
8. if fd=-1
9. print file is opened for read/write access
10. write the message into the file
11. lseek(fd,ol,o)
12. if
13. print message written to file
14. else
15. print error and close the file
16. else print file exists
17. stop

**PROGRAM:**

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdlib.h>

static char message [] = "Hello, world";

int main()

{

int fd;

char buffer [80];

fd = open("df.dat",O\_RDWR | O\_CREAT | O\_EXCL, S\_IREAD |

S\_IWRITE);

if (fd != -1)

{

printf("datafile df.dat opened for read/write access\n");

write(fd, message, sizeof(message));

lseek(fd, 0L, 0);

if (read(fd, buffer, sizeof(message)) == sizeof (message))

printf("\"%s\" was written to datafile.dat\n", buffer);

else

printf("\*\*\* error reading datafile.dat \*\*\*\n");

close (fd);

}

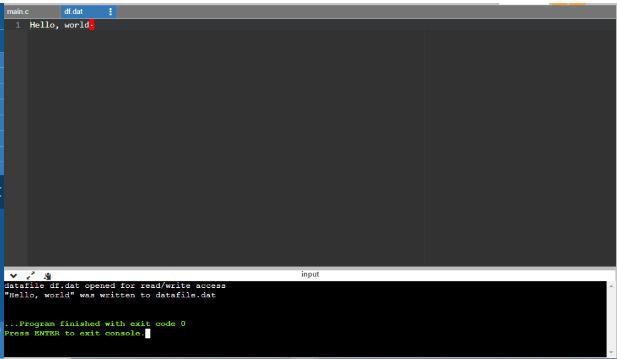
else

printf("\*\*\* datafile.dat already exists \*\*\*\n");

exit (0);

}

**OUTPUT:**



|  |  |
| --- | --- |
| **EX.NO:08-B** | **USE OF STAT SYSTEM CALLS** |
| **DATE:** |

**AIM:**

To wite a program to exhibit the concept of stat system calls and its uses.

**ALGORITHM:**

1. Start
2. Declare s structure and a variable s
3. Declare necessary variables
4. If(stat(“test’,&s)==-1)
5. Show a perror
6. Exit
7. Compute size of input files
8. Print the file sizes

**PROGRAM :**

#include <stdio.h>

#include <sys/stat.h>

#include <stdlib.h>

int main()

{

struct stat s;

int a; int b;

if(stat("HEMA",&s)==(-1))

{

perror("Error: cannot stat file");

exit(0);

}

a=s.st\_blksize;

b=s.st\_size;

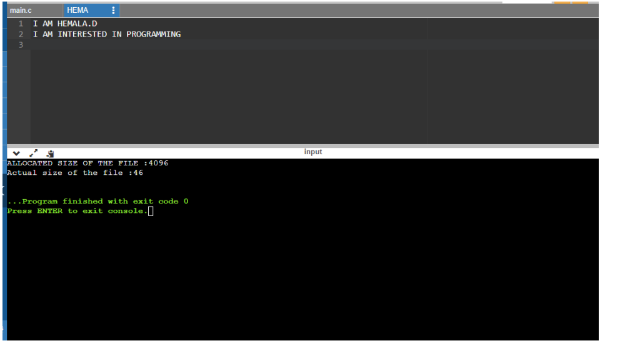
printf("ALLOCATED SIZE OF THE FILE :%d\nActual size of the file

:%d\n",a,b);

return 0;

}

**OUTPUT:**



|  |  |
| --- | --- |
| **EX.NO:08-C** | **USE OF DIRECTORY SYSTEM CALLS** |
| **DATE:** |

**AIM:**

To write a program to illustrate the concept of directory system calls.

**ALGORITHM:**

1. Start
2. Im[port necessary header files and variables
3. Search( )
4. Create a pointer for DIR
5. If file is empty or directory is empty
6. Print unable to open directory
7. Get the name of file to be searched
8. While dir is empty or NULL
9. Check for the file
10. Add 1 to flag
11. If flag is 1
12. Print file is found
13. Else
14. Print file not found
15. Main( )
16. Get name of directory
17. Search (name)
18. Stop

**PROGRAM :**

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

#include<string.h>

void sea(char \*dname)

{

DIR \*dir;

struct dirent \*ent;

int flag = 0;

char a[15];

if ((dir= opendir(dname))==NULL)

{

printf("\n unable to open directory ");

exit(1);

}

printf("\n Enter the name of the file to be searched :");

scanf("%s",a);

while((ent=readdir(dir))!=NULL)

{

if(!strcmp(a,ent->d\_name))

{

printf("%s",ent->d\_name);

flag++;

}

}

if(flag==1)

printf("\n the given file is found\n\n");

else

printf("\nfile not found");

if(closedir(dir)!=0)

printf("unable to close directory");

}

void main()

{

char dirname[25];

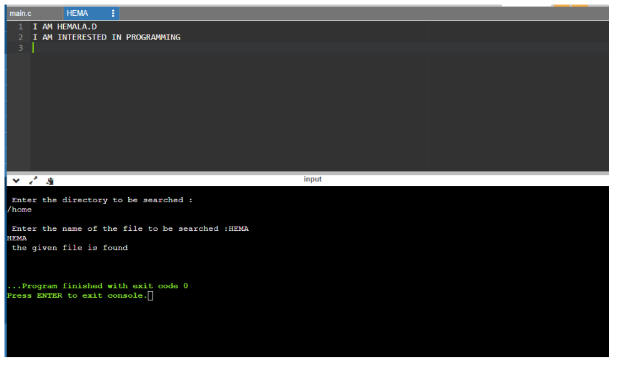
printf("\n Enter the directory to be searched :\n");

scanf("%s",dirname);

sea(dirname);

}

**OUTPUT:**



|  |  |
| --- | --- |
| **EX.NO:10** | **BANKER’S ALGORITHM** |
| **DATE:** |

**AIM:**

To illustrate Banker’s algorithm using C Program.

**ALGORITHM:**

1. Start
2. Declare the variables
3. Create various method to calculate
4. Print method,

        Using for loop,

               DISPLAY the required values to be printed

1. Safety method,

        In this method predict whether the resources can be allocated or not and ensure safety of the process.

1. Resource requested method,

        Create a resources request method is the user needs additional request then this method will be called

         And the process are checked for the new request and safety is also ensured.

1. Mainly used method is Banker’s method
2. Here in Banker’s method

          Calculate need matrix by

                              Maximum - Allocation

1. Create a accept method

          To input total no: of process and resources

          Also input the available resources from the user.

1. DISPLAY Allocation , Maximum requirement and Need matrix
2. From the main method call every method to perform the task

          Ask whether there is an resource request from the user,

                  if yes then use that request method

1. DISPLAY output
2. Stop

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

void print(int x[][10],int n,int m){

int i,j;

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

printf("\n");

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

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

}

}

}

void res\_request(int A[10][10],int N[10][10],int AV[10][10],int pid,int m)

{

int reqmat[1][10];

int i;

printf("\n\_\_\_FOR ADDITINAL REQUEST:\_\_\_");

printf("\n Enter additional request :- \n");

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

printf(" Request for resource %d : ",i+1);

scanf("%d",&reqmat[0][i]);

}

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

if(reqmat[0][i] > N[pid][i]){

printf("\n The request can be granted.\n");

exit(0);

}

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

if(reqmat[0][i] > AV[0][i]){

printf("\n Resources unavailable.\n");

exit(0);

}

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

AV[0][i]-=reqmat[0][i];

A[pid][i]+=reqmat[0][i];

N[pid][i]-=reqmat[0][i];

}

}

int safety(int A[][10],int N[][10],int AV[1][10],int n,int m,int a[]){

int i,j,k,x=0;

int F[10],W[1][10];

int pflag=0,flag=0;

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

F[i]=0;

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

W[0][i]=AV[0][i];

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

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

if(F[i] == 0){

flag=0;

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

if(N[i][j] > W[0][j])

flag=1;

}

if(flag == 0 && F[i] == 0){

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

W[0][j]+=A[i][j];

F[i]=1;

pflag++;

a[x++]=i;

}

}

}

if(pflag == n)

return 1;

}

return 0;

}

void accept(int A[][10],int N[][10],int M[10][10],int W[1][10],int \*n,int \*m){

int i,j;

printf("\n Enter total no. of processes : ");

scanf("%d",n);

printf("Enter total no. of resources : ");

scanf("%d",m);

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

printf("\n\tProcess %d\n",i+1);

for(j=0;j<\*m;j++){

printf(" Allocation for resource %d : ",j+1);

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

printf(" Maximum for resource %d : ",j+1);

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

}

}

printf("\n Available resources : \n");

for(i=0;i<\*m;i++){

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

scanf("%d",&W[0][i]);

}

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

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

N[i][j]=M[i][j]-A[i][j];

printf("\n Allocation Matrix");

print(A,\*n,\*m);

printf("\n Maximum Requirement Matrix");

print(M,\*n,\*m);

printf("\n Need Matrix");

print(N,\*n,\*m);

}

int banker(int A[][10],int N[][10],int W[1][10],int n,int m){

int j,i,a[10];

j=safety(A,N,W,n,m,a);

if(j != 0 ){

printf("\n\n");

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

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

printf("\n Hence the process sequence is safe...\n");

return 1;

}else{

printf("\n The process sequence is not safe...\n");

return 0;

}

}

int main(){

int ret;

int A[10][10];

int M[10][10];

int N[10][10];

int W[1][10];

int n,m,pid,ch;

printf("\n\_\_\_BANKER'S ALGORITHM\_\_\_\n");

accept(A,N,M,W,&n,&m);

ret=banker(A,N,W,n,m);

if(ret !=0 ){

printf("\n Want make an additional request ? (1=Yes|0=No)");

scanf("%d",&ch);

if(ch == 1){

printf("\n Enter process no. : ");

scanf("%d",&pid);

res\_request(A,N,W,pid-1,m);

ret=banker(A,N,W,n,m);

if(ret == 0 )

exit(0);

}

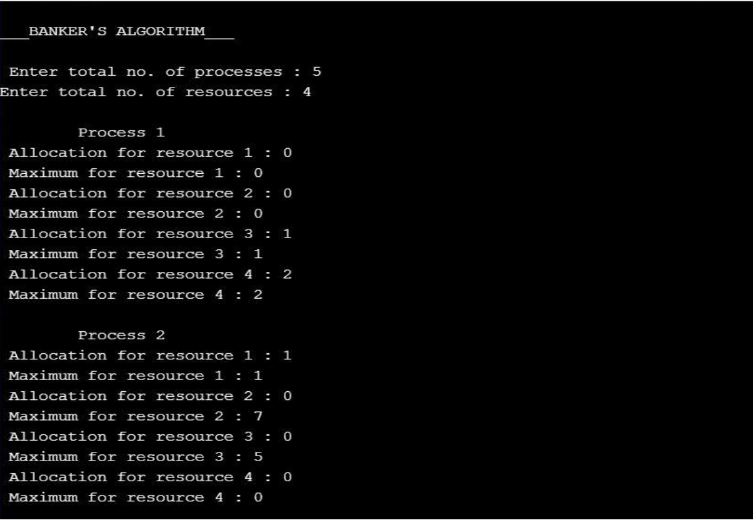
}else

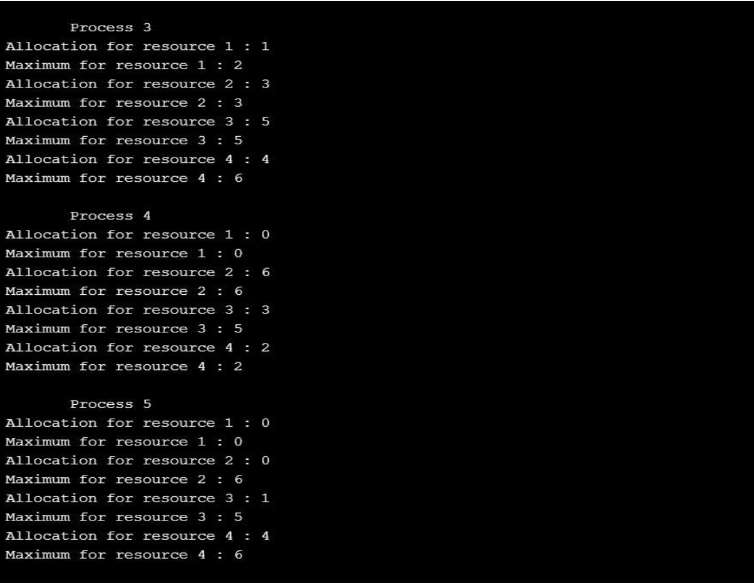
exit(0);

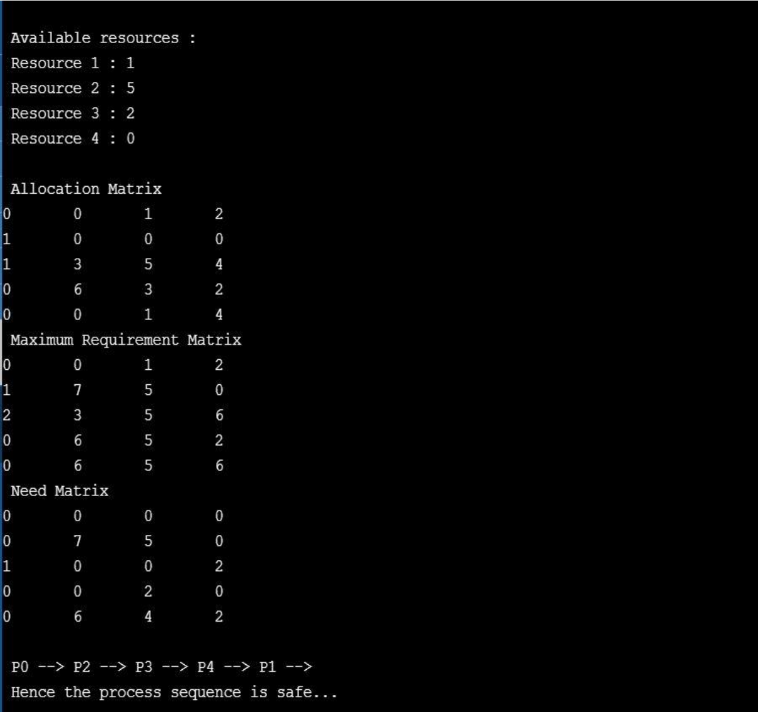
return 0;

}

**OUTPUT:**







  
  
**RESULT:**

Thus the Bankers algorithm using c program has been illustrated and executed

successfully.

|  |  |
| --- | --- |
| **EX NO : 11** | **PAGE REPLACEMENT ALGORITHM** |
| **DATE:** |

**FIFO PAGE REPLACEMENT ALGORITHM:**

**AIM:**

To illustrate the page replacement algorithm using C program

**ALGORITHM:**

1. START
2. Declare the variables required
3. Input the page numbers and page frames from the user using for loop
4. Check the need of replacement from old page to new page in memory using for loop

If frame[k]==a[i]

Initialize avail =1

If (avail==0)

Assign frame[j]=a[i]

j = (j+1) % num

Increment count by one

1. Form a queue to hold the pages
2. Get the page numbers and insert into the queue
3. Check for the page fault
4. Display the page numbers
5. Display the total numbers of page fault
6. STOP

**PROGRAM:**

#include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

printf("\_\_\_FIFO PAGE REPLACEMENT ALGORITHM:\_\_\_");

printf("\n\nENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\nENTER THE PAGE NUMBER :\n");

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

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

printf("\nENTER THE NUMBER OF FRAMES :");

scanf("%d",&no);

for(i=0;i<no;i++)

frame[i]= -1;

j=0;

printf("Reg page\t Frames\n");

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

{

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

avail=0;

for(k=0;k<no;k++)

if(frame[k]==a[i])

avail=1;

if (avail==0)

{

frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

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

}

printf("\n");

}

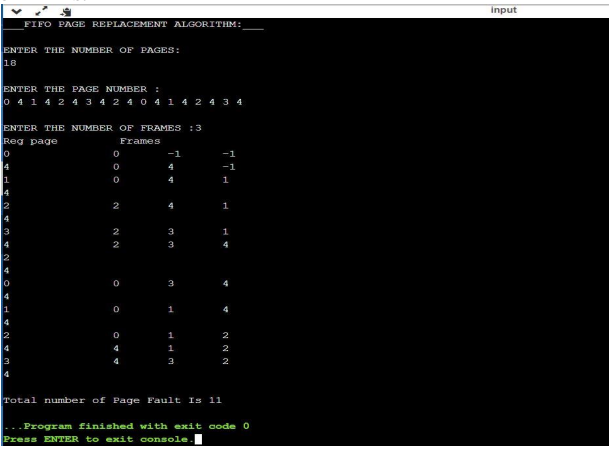
printf("\nTotal number of Page Fault Is %d",count);

return 0;

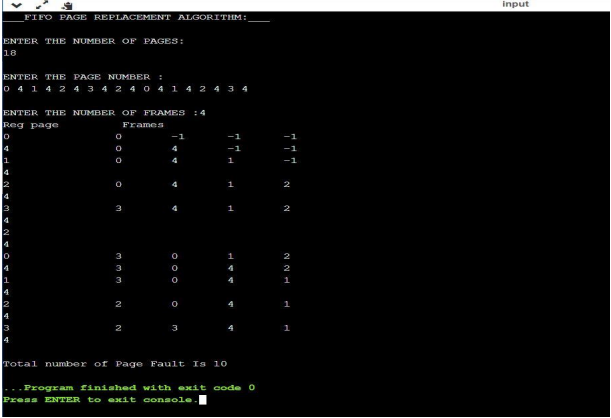
}

**OUTPUT:**

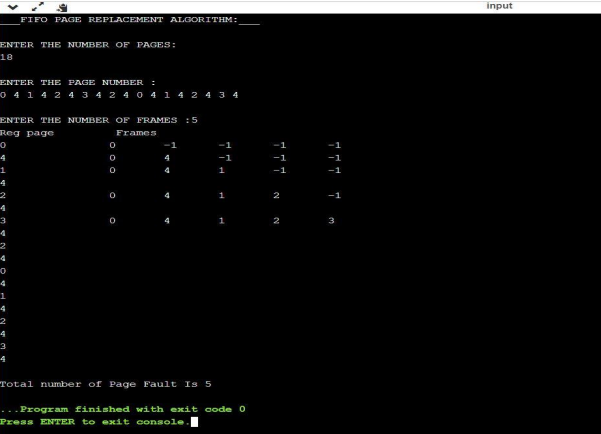
**3 FRAMES:**



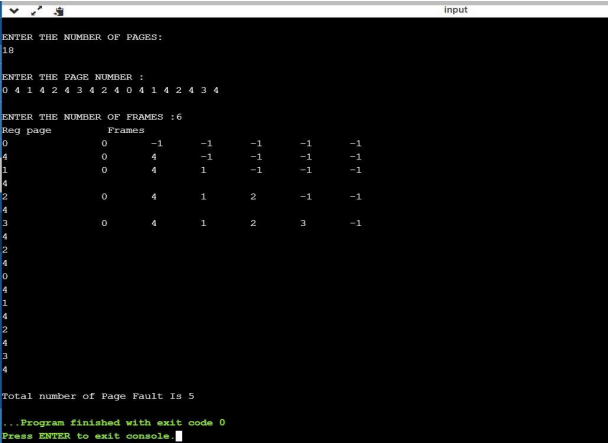
**4 FRAMES:**



**5 FRAMES:**



**6 FRAMES:**



**RESULT:**

Thus the page replacement algorithm using c program has been completed and executed successfully.

|  |  |
| --- | --- |
| **EX.NO: 12** | **DISK SCHEDULING** |
| **DATE:** |

**AIM:**

To illustrate the given disk scheduling using c programme.

**ALGORITHM:**

**LOOK:**

1. START
2. Declare the required variables
3. Get the current head position from user
4. Input the number of requests
5. Get all the request and store it and also the upper bound
6. To check whether the head is moving towards upper bound or lower bound get the info from user
7. Using for loop perform

The head continue in moving in same direction until all the request in the direction are not finished

While moving in this direction calculate the absolute distance of track of the head

Increment total seek count with this distance

1. Display the need output
2. STOP

**SCAN:**

1. START
2. Declare the required variables
3. Input the head position and the total requests
4. Let direction represents the head is moving towards right or left
5. Using for loop perform

Calculate the absolute distance of the track from the head

Increment the total seek count with this distance

Currently serviced track will become the head now

1. Perform the operation until one end of distance is reached
2. If one end is reached then reverse the direction continue the looping process until are the tracks are serviced
3. Display the required output
4. STOP

**PROGRAM:**

**LOOK:**

#include<math.h>

#include<stdio.h>

int main()

{

int i,n,j=0,k=0,x=0,l,req[50],mov=0,cp,ub,end, lower[50],upper[50], temp,a[50];

printf("\_\_\_DISK SCHEDULING\_\_\_(LOOK)\n\n");

printf("Enter the current head position: ");

scanf("%d",&cp);

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

scanf("%d",&n);

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

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

{

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

}

printf("Enter the upper bound: ");

scanf("%d",&ub);

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

{

if(req[i]<cp)

{

lower[j]=req[i];

j++;

}

if(req[i]>cp)

{

upper[k]=req[i];

k++;

}

}

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

{

for(l=0;l<j

-1;l++)

{

if(lower[l]<lower[l+1])

{

temp=lower[l];

lower[l]=lower[l+1];

lower[l+1]=temp;

}

}

}

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

{

for(l=0;l<k

-1;l++)

{

if(upper[l]>upper[l+1])

{

temp=upper[l];

upper[l]=upper[l+1];

upper[l+1]=temp;

}

}

}

printf("Enter the end to which the head is moving 0 - for lower end and 1 - for upper

end\n");

scanf("%d",&end);

printf("--------------------------------------------------------------------------------------------------

\n");

printf("Solution:");

printf("\n\n Movement:\n");

switch(end)

{

case 0:

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

{

a[x]=lower[i];

x++;

}

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

{

a[x]=upper[i];

x++;

}

break;

case 1:

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

{

a[x]=upper[i];

x++;

}

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

{

a[x]=lower[i];

x++;

}

break;

}

mov=mov+abs(cp-a[0]);

printf("%d -> %d",cp,a[0]);

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

{

mov=mov+abs(a[i]-a[i-1]);

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

}

printf("\n");

printf("Total distance in cylinders = %d cylinders\n",mov);

}

**SCAN:**

#include <stdio.h>

#include <stdlib.h>

#define LOW 0

#define HIGH 4299

int main(){

int queue[20];

int head, max, q\_size, temp, sum;

int dloc;

printf("\n\_\_\_DISK SCHEDULING\_\_\_(SCAN)\n\n");

printf("Enter head position:");

scanf("%d", &head);

printf("Enter no.of Disk Requests:");

scanf("%d", &q\_size);

printf("Enter the elements into disk queue:\n");

for(int i=0; i<q\_size; i++){

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

}

queue[q\_size] = head;

q\_size++;

for(int i=0; i<q\_size;i++){

for(int j=i; j<q\_size; j++){

if(queue[i]>queue[j]){

temp = queue[i];

queue[i] = queue[j];

queue[j] = temp;

}

}

}

max = queue[q\_size-1];

for(int i=0; i<q\_size; i++){

if(head == queue[i]){

dloc = i;

break;

}

}

if(abs(head-LOW) <= abs(head-HIGH)){

for(int j=dloc; j>=0; j--){

}

for(int j=dloc+1; j<q\_size; j++){

}

} else {

for(int j=dloc+1; j<q\_size; j++){

}

for(int j=dloc; j>=0; j--){

}

}

sum = head + max;

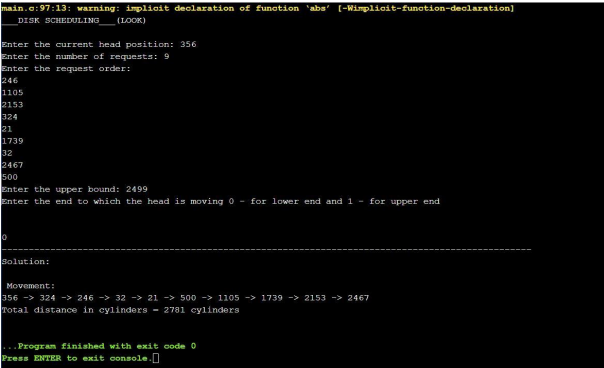
printf("\nTotal Seek Time: %d cylinders", sum);

return 0;

}

**OUTPUT:**

**LOOK:**



**SCAN:**



**RESULT:**

Thus the given disk scheduling using c program has been completed and executed successfully.

|  |  |
| --- | --- |
| **EX.NO:13** | **SEGMENTATION PROBLEM** |
| **DATE:** |

**AIM:**

To illustrate the segmentation problem using C program.

**ALGORITHM:**

1. START
2. Declare the required variables
3. Input the number of segments, base values, length values, logical address(offset).
4. Perform the operation

If the logical address (offset)is less than equal to length

Then do

Sum up offset with base address

Else

DISPLAY segmentation fault

1. DISPLAY the required output
2. STOP

**PROGRAM:**

#include <stdio.h>

int main()

{

int n,bv[10],lv[10],la[10],sum[10];

printf("\t\t\_\_\_SEGMENTATION\_\_\_\n\n");

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

scanf("%d",&n);

printf("Enter the base values:\n");

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

{

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

}

printf("\nEnter the length values:\n");

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

{

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

}

printf("\nEnter the logical addresses: \n");

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

{

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

}

printf("\nSEGMENT\t BASE\tLENGTH\tOFF SET\n");

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

{

printf(" %d\t %d\t %d\t %d\n",i+1,bv[i],lv[i],la[i]);

}

printf("\n\nPHYSICAL ADDRESS\n");

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

{

if(la[i]<=lv[i])

{

sum[i]=bv[i]+la[i];

printf("%d\n",sum[i]);

}

else{

printf("Segmentation fault\n");

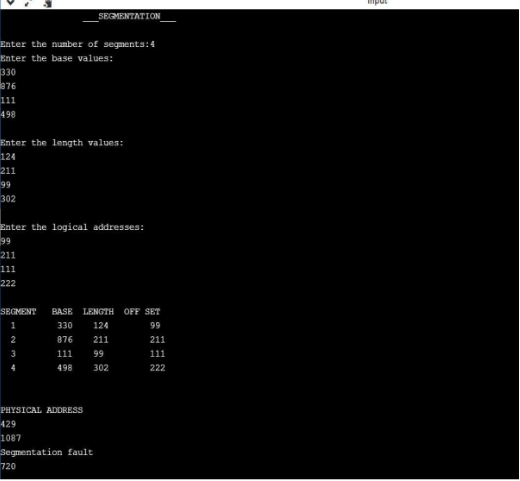
}

}

return 0;}

727721EUIT156

**OUTPUT:**



**RESULT:**

Thus, the segmentation problem using c program has been completed and executed successfully.

**RESULT:**

Thus the interprocess communication producer consumer problem has been illustrated

and executed successfully**.**