**Ex No :1 BASICS OF UNIX COMMANDS**

**I File and Directory Related commands**

**1) pwd**

This command prints the current working directory

**[user@sys108 ~]$ pwd**

**/home/user**

**2) ls**

This command displays the list of files in the current working directory.

$ls –l Lists the files in the long format

$ls –t Lists in the order of last modification time

$ls –d Lists directory instead of contents

$ls -u Lists in order of last access time

**[user@sys108 ~]$ ls**

**hello.txt first.c perl1 sample hai.c**

**3) cd**

This command is used to change from the working directory to any other directory specified.

$cd directoryname

**[user@sys108 ~]$ cd jean**

**[user@sys108 jean]$**

**4) cd ..**

This command is used to come out of the current working directory.

$cd ..

**[user@sys108 jean]$ cd ..**

**[user@sys108 ~]$**

**5) mkdir**

This command helps us to make a directory.

$mkdir directoryname

**[user@sys108 ~]$ mkdir jean**

**[user@sys108 ~]$ ls**

**hello.txt first.c perl1 sample hai.c jean**

**6) rmdir**

This command is used to remove a directory specified in the command line. It requires the

specified directory to be empty before removing it.

$rmdir directoryname

**[user@sys108 ~]$ rmdir jean**

**hello.txt first.c perl1 sample hai.c**

**7) cat**

This command helps us to list the contents of a file we specify.

$cat [option][file]

cat > filename – This is used to create a new file.

cat >>filename – This is used to append the contents of the file

**[user@sys108 ~]$ cat > jean1**

**Have a Good Day**

**[user@sys108 ~]$ cat >> jean1**

**Welcome**

**[user@sys108 ~]$ cat jean1**

**Have a Good Day**

**Welcome**

**8) cp**

This command helps us to create duplicate copies of ordinary files.

$cp source destination

**[user@sys108 ~]$ cp jean1 jean2**

**[user@sys108 ~]$ cat jean2**

**Have a Good Day**

**Welcome**

**9) mv**

This command is used to move or rename files.

$mv source destination

**[user@sys108 ~]$ mv jean1 jean3**

**[user@sys108 ~]$ cat jean3**

**Have a Good Day**

**Welcome**

**10) ln**

This command is to establish an additional filename for the same ordinary file.

$ln firstname secondname

**user@sys108 ~]$ ln jean2 cse1**

**11) rm**

This command is used to delete one or more files from the directory.

$rm [option] filename

$rm –i Asks the user if he wants to delete the file mentioned.

$rm –r Recursively delete the entire contents of the directory as well as the

directory itself.

**[user@sys108 ~]$ rm jean2**

**II) Process and status information commands**

**1) who**

This command gives the details of who all have logged in to the UNIX system currently.

$ who

**[user@sys108 ~]$ who**

**user tty1 2014-11-11 19:16 (:0)**

**user pts/0 2014-11-11 19:19 (:0.0)**

**user pts/1 2014-11-11 19:19 (:0.0)**

**2) who am i**

This command tells us as to when we had logged in and the system’s name for the

connection being used. $who am i

**[user@sys108 ~]$ who am i**

**user pts/0 2014-11-11 19:19 (:0.0)**

**3) date**

This command displays the current date in different formats.

+%D mm/dd/yy +%w Day of the week

+%H Hr-00 to 23 +%a Abbr.Weekday

+%M Min-00 to 59 +%h Abbr.Month

+%S Sec-00 to 59 +%r Time in AM/PM

+%T HH:MM:SS +%y Last two digits of the year

**[user@sys108 ~]$ date**

**Tue Nov 11 19:39:14 IST 2014**

**4) echo**

This command will display the text typed from the keyboard.

$echo

**[user@sys108 ~]$ echo welcome to 2015**

**welcome to 2015**

**III Text related commands**

**1. head**

This command displays the initial part of the file. By default it displays first ten lines of the

file.

$head [-count] [filename]

**[user@sys108 ~]$ head -1 jean3**

**Have a Good Day**

**2. tail**

This command displays the later part of the file. By default it displays last ten lines of the

file.

$tail [-count] [filename]

**[user@sys108 ~]$ tail -1 jean3**

**Welcome**

**3. wc**

This command is used to count the number of lines, words or characters in a file.

$wc [-lwc] filename

**[user@sys108 ~]$ wc -lwc jean3**

**2 4 24 jean3**

**4. find**

The find command is used to locate files in a directory and in a subdirectory.

The –name option

This lists out the specific files in all directories beginning from the named

directory. Wild cards can be used.

The –type option

This option is used to identify whether the name of files specified are

ordinary files or directory files. If the name is a directory then use “-type d

“and if it is a file then use “-type f”.

The –mtime option

This option will allow us to find that file which has been modified before

or after a specified time. The various options available are –mtime n(on a

particular day),-mtime +n(before a particular day),-mtime –n(after a particular

day)

The –exec option

This option is used to execute some commands on the files that are found

by the find command.

**[user@sys108 ~]$ find fc.c**

**fc.c**

**IV File Permission commands**

**1) chmod**

Changes the file/directory permission mode: $ chmod 777 file1

Gives full permission to owner, group and others

$ chmod o-w file1

Removes write permission for others.

**[user@sys108 ~]$ chmod 777 welcome.txt**

**V Useful Commands:**

**1) exit** - Ends your work on the UNIX system.

**[user@sys108 ~]$ exit**

**exit**

**There are stopped jobs.**

**2) Ctrl-l or clear**

Clears the screen.

**[user@sys108 ~]$ clear**

**[user@sys108 ~]$**

**3) Ctrl-c**

Stops the program currently running.

**4) Ctrl-z**

Pauses the currently running program.

**5) man COMMAND**

Looks up the UNIX command COMMAND in the online manual pages.

**[user@sys108 ~]$ man**

**What manual page do you want?**

**[user@sys108 ~]$ man ls**

**6) history**

List all commands typed so far.

**[user@sys108 ~]$ history**

**109 man**

**110 man help**

**111 man ls**

**112 history**

**7) more FILE**

Display the contents of FILE, pausing after each screenful.

There are several keys which control the output once a screenful has been printed.

<enter> Will advance the output one line at a time.

<space bar> Will advance the output by another full screenful.

"q" Will quit and return you to the UNIX prompt.

**[user@sys108 ~]$ more jean3**

**Have a Good Day**

**Welcome**

**8) less FILE**

"less" is a program similar to "more", but which allows backward movement

in the file as well as forward movement.

**[user@sys108 ~]$ less jean3**

**Have a Good Day**

**Welcome**

**jean3 (END)**

**9) lpr FILE**

To print a UNIX text or PostScript file, type the following command at the system

prompt:

**[user@sys108 ~]$ lpr jean3**

**lpr: Error - no default destination available.**

**EXP NO : 2(i) COMPARISON OF TWO STRINGS USING SHELL PROGRAM**

**AIM:**

To write a shell program to compare the two strings.

**ALGORITHM :**

1. Enter into the vi editor and go to the insert mode for entering the code
2. Read the first string.
3. Read the second string
4. Compare the two strings using the if loop
5. If the condition satisfies then print that two strings are equal else print two

strings are not equal.

1. Enter into the escape mode for the execution of the result and verify the output

**PROGRAM :**

#!/bin/bash  **//instruct the operating system to use bash as a command interpreter**

echo "Enter the first String : "

read str1 **//read the value of first string**

echo "Enter the second String : "

read str2 **//read the value of second string**

if [ $str1 = = $str2 ] **//$ is used before variable name to access its value. Space should be properly given**

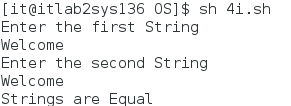
then

echo "Strings are Equal"

else

echo "Strings are Unequal"

fi **//to close if block**

**OUTPUT :**

**RESULT :**

Thus the shell program to compare two strings is executed successfully and the output is verified.

**EXP NO : 2(ii) MAXIMUM OF THREE NUMBERS USING SHELL PROGRAM**

**AIM:**

To write a shell program to find greatest of three numbers.

**ALGORITHM:**

1. Declare the three variables.
2. Check if A is greater than B and C.
3. If so print A is greater.
4. Else check if B is greater than C.
5. If so print B is greater.
6. Else print C is greater.

**PROGRAM :**

#!/bin/bash

echo "Enter A : "

read a

echo "Enter B : "

read b

echo "Enter C : "

read c

if [ $a -gt $b -a $a -gt $c ] **//-gt means "greater than" and and –a is used to combine two conditions**

then

echo "A is greater"

elif [ $b -gt $a -a $b -gt $c ]

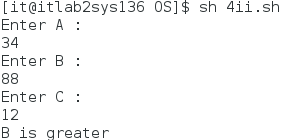
then

echo "B is greater"

else

echo "C is greater"

fi

**OUTPUT :**

**RESULT :**

Thus the shell program to find the maximum of three numbers is executed successfully and the output is verified.

**EXP NO : 2(iii) MULTIPLICATION TABLE USING SHELL PROGRAM**

**AIM:**

To write the shell program to check whether the number is odd or even.

**ALGORITHM :**

Step 1 : Start the program

Step 2 : Get the input, Use a for loop to read numbers in the range from 1 to 10

Step 3 : Calculate the result

Step 4 : Increment the variable and repeat

Step 5 : Display the result.

Step 6 : Terminate the program.

**PROGRAM :**

#!/bin/bash

echo "which number to generate multiplication table"

read number

i=1

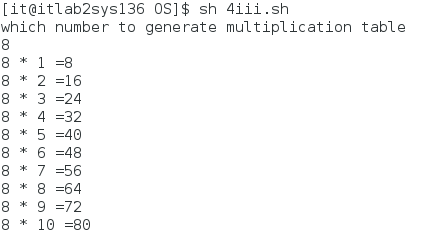
while [ $i -le 10 ] **// -le means less than**

do

echo "$number \* $i =`expr $number \\* $i ` "

i=`expr $i + 1` **//expr is a key to evaluate the expression and it should be placed inside back-tick symbol**

done **//ends while loop**

**OUTPUT :**

**RESULT :**

Thus the shell program to generate multiplication table is executed successfully and the output is verified.

**EXP NO : 2(iv) BASIC ARITHMETIC OPERATION USING SHELL PROGAMMING**

**AIM:**

To write the shell program to perform the basic arithmetic operations.

**ALGORITHM :**

Step 1 : Start the program.

Step 2 : Read the inputs

Step 3 : Perform the arithmetic calculation.

Step 4 : Print the result.

Step 5 :Stop the execution.

**PROGRAM :**

#!/bin/bash

echo "Enter value A :"

read a

echo "enter value B : "

read b

c=`expr $a + $b`

echo "sum:"$c

c=`expr $a - $b`

echo "sub:"$c

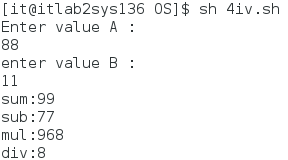
c=`expr $a \\* $b`

echo "mul:"$c

c=`expr $a / $b`

echo "div:"$c

**OUTPUT :**



**RESULT :**

Thus the shell program to perform basic arithmetic operations is executed successfully and the output is verified.

**Ex No:3(i) Implementation of Unix system calls : OPENDIR, READDIR.**

**AIM**

To write a program for implementing the following directory system calls of UNIX operating system: opendir, readdir.

**ALGORITHM**

1.Open the directory specified in command line input.

1. Display the directory contents.

**PROGRAM**

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

#include<sys/types.h>

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

{

DIR \*dname; **//to create a variable of type Directory / Folder**

struct dirent \*dptr**; // points to a string that gives the name of a file in the directory**

printf("\nListening the directory content");

if(argc!=2) **//only one folder name should be given as argument while executing (argc value will be 2) . If argument is not given or more than one argument (argc value will not be 2) is given error will be thrown using this block**

{

printf("Usage :./a.out <dirname>\n");

exit(-1);

}

if((dname=opendir(argv[1]))==NULL){ **//If given argument is not a folder name, error will be thrown**

perror(argv[1]);

exit(-1);

}

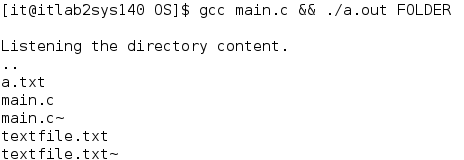
while(dptr=readdir(dname)){ **//reads the file one by one and displays the file names**

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

}

closedir(dname);

}

**OUTPUT :**

**RESULT**

Thus the program for implementing directory system calls: open and read was successfully executed.

**Ex No:3(ii) Implementation of Unix system calls : STAT.**

**AIM:**

To write the C program to implement the system call stat( ).

**ALGORITHM :**

Step 1 : Include the necessary header files to execute the system call stat( ).

Step 2 : Input file path of source and destination file.

Step 3 : Open source file in r (read) and destination file in w (write) mode.

Step 4 : [Read character from source file](https://codeforwin.org/2018/01/c-program-read-and-display-file-contents.html) and [write it to destination file](https://codeforwin.org/2018/01/c-program-create-file-write-contents.html) .Repeat step 3 till source file has reached end.

Step 5 :Close both source and destination file.

**SYSTEM CALLS USED :**

**1. creat()**

The creat() function either creates a new file or prepares to rewrite an existing file named by the path name pointed to by pathname.

**2. open( )**

The open() system call opens the file specified by pathname. The return value of open() is a file descriptor, a small nonnegative integer that is an index to an entry in the process’s table of open file descriptors.

**3. fstat( )**

fstat() is a system call that is used to determine information about a file based on its file descriptor.

**4. stat()**

The stat() function obtains information about the file pointed to by path. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

**PROGRAM CODING**:

#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<unistd.h>

#include<fcntl.h>

main()

{

int fd1,fd2,n;

char source[30],ch[5];

struct stat s,t,w; // struct stat is used to retrieve information about files such as file size, permissions, timestamps, and more. ‘s’ variable represents information about the current file or directory. It contains details like file size, permissions, and timestamp. ‘t’ variable could be used to store information about another file or directory.

fd1=creat("text.txt",0644); // creat does create a new file or rewrite an existing one. Creat is defined in fcntl.h. creat(path,mode) is the syntax. 0644 represents that the file owner can read and write

printf("Enter the file to be copied\n");

scanf("%s",source);

fd2=open(source,0); //opening source file in read only mode

if(fd2==-1) //if file doesnot exist, open returns -1

{

perror("file doesnot exist");

exit(0);

}

while((n=read(fd2,ch,1))>0) //while loop executes until all characters are copied from fd2 to fd1. Reads single byte from fd2 and store it in ch

write(fd1,ch,n); //writes n bytes of ch to fd1

close(fd2);

stat(source,&s); //to get the status information of source file in s

printf("Source file size=%d\n",s.st\_size); //st\_size gives the size of the file in bytes

fstat(fd1,&t); //fstat gets information about files. Stat applies on file name while fstat applies on file descriptor

printf("Destination file size =%d\n",t.st\_size);

close(fd1);

}

**OUTPUT:**

[2cse@localhost ~] vi ex.c

[2cse@localhost ~] cc ex.c

[2cse@localhost ~] ./a.out

Enter the file to be copied

ex.c

Source file size=554

Destination file size =554

[2cse@localhost ~]

**RESULT:**

Thus the C program to implement the system call stat( ) is successfully executed and output is verified.

**Ex No: 4 SIMULATION OF UNIX COMMANDS LS,CP,GREP**

**AIM**

To write a C program to simulate UNIX commands like ls,cp,grep.

**ALGORITHM**

1. Create a file using cat command.
2. Simulate ls command to list all the files in current working directory
3. Open the file in read mode and copy the contents of the file.
4. Input the word to be searched in the file.
5. Display the word if searching is successful.

**PROGRAM**

**i)ls**

#include<stdio.h>

#include<dirent.h>

int main()

{

struct dirent \*\*namelist; //dirent is used to get information about files and subdirectories inside the directory and store it in the name ‘namelist’

int n,i;

char pathname[100];

printf("ENTER THE PATH : ");

scanf("%s",pathname);

n=scandir(pathname,&namelist,0,alphasort); // scans the directory given by the path name, sorts the file in alphabetical order and store it in namelist. n represents number of files inside directory

if(n<0){

printf("No such file or directory");

}else{

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

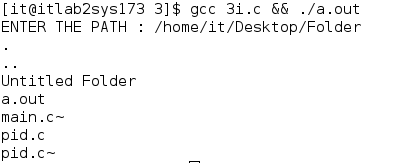
printf("%s\n",namelist[i]->d\_name);

}

}

}

**OUTPUT :**



**ii)grep**

**PROGRAM :**

#include <stdio.h>

#include <string.h>

int main()

{

char fn[30], pat[30], temp[200];

FILE \*fp;

printf("Enter file name : ");

scanf("%s", fn);

printf("Enter pattern to be searched : ");

scanf("%s", pat);

fp = fopen(fn, "r");

if(fp==NULL){

printf("File not fount");

return 0;

}

while (!feof(fp)) //until end of file is reached

{

fgets(temp, 1000, fp); //The fgets() reads a line from fp and stores it into the string pointed to by temp. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

if (strstr(temp, pat) != NULL) //strstr checks whether pat is present in temp

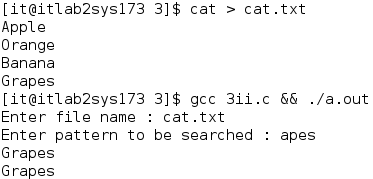
printf("%s", temp);

}

fclose(fp);

}

**OUTPUT :**

****

**iii)cp**

**PROGRAM :**

#include <sys/stat.h>

#include <sys/types.h>

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main(int argc, char \*argv[]) //source and destination file name is given as argument during runtime which is stored in argv

{

int i, fd1, fd2;

char \*file1, \*file2, buf[2];

file1 = argv[1]; //represents source name given during runtime

file2 = argv[2]; //represents destination name given during runtime

fd1 = open(file1, O\_RDONLY); //opens file1 in read only mode

if (fd1 == -1) //open returns -1 if we cannot open file

{

printf("%s File not found",file1);

return 0;

}

fd2 = creat(file2, 0777); //create file2 . 0777 gives full permission such as read, write

while (i = read(fd1, buf, 1) > 0) // reads a single byte from the file descriptor fd1 into the buffer buf.

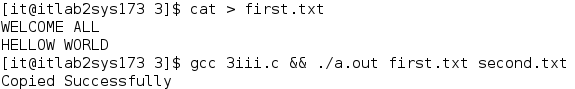
write(fd2, buf, 1); //writes a single byte to the file descriptor fd2 from the buffer buf.

printf("Copied Successfully\n");

close(fd1);

close(fd2);

}

**OUTPUT :** 

**RESULT**

Thus the C program to simulate UNIX commands like ls,cp,grep was executed successfully.

**Ex.No: 5** **IMPLEMENTATION OF SHARED MEMORY AND IPC**

**AIM**

To write a C program to illustrate interprocess communication using Shared Memory system calls.

**ALGORITHM**

1. Create shared memory using shmget( ) system call

2. If successfull it returns positive value

3. Attach the created shared memory using shmat( ) systemcall.

4. Write to shared memory using shmsnd( ) system call

5. Read the contents from shared memory using shmrcv( ) systemcall

**PROGRAM**

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <stdio.h>

#include <stdlib.h>

int main()

{

int shmid;

key\_t key = 0 \* 10; // Unix requires a key of type key\_t defined in file sys/types.h for requesting resources such as shared memory segments, message queues and semaphores. A key is simply an integer of type key\_t;

shmid = shmget(key, 100, IPC\_CREAT | 0666); //shmget is used to create a new shared memory segment or access an existing one.The key is an identifier for the shared memory segment.The second argument specifies the size of the shared memory segment in bytes (in this case, 100 bytes).IPC\_CREAT | 0666 flags indicate that a new shared memory segment should be created if it doesn’t exist, and the permissions for the segment are set to read and write for all users (0666).

if (shmid < 0)

printf("First SHMID failed\n\n");

else

printf("First SHMID succeded id=%d\n\n", shmid);

shmid = shmget(key, 101, IPC\_CREAT | 0666);

if (shmid < 0)

printf("Second SHMID failed\n\n");

else

printf("Secondt SHMID succeded id=%d\n\n", shmid);

shmid = shmget(key, 90, IPC\_CREAT | 0666);

if (shmid < 0)

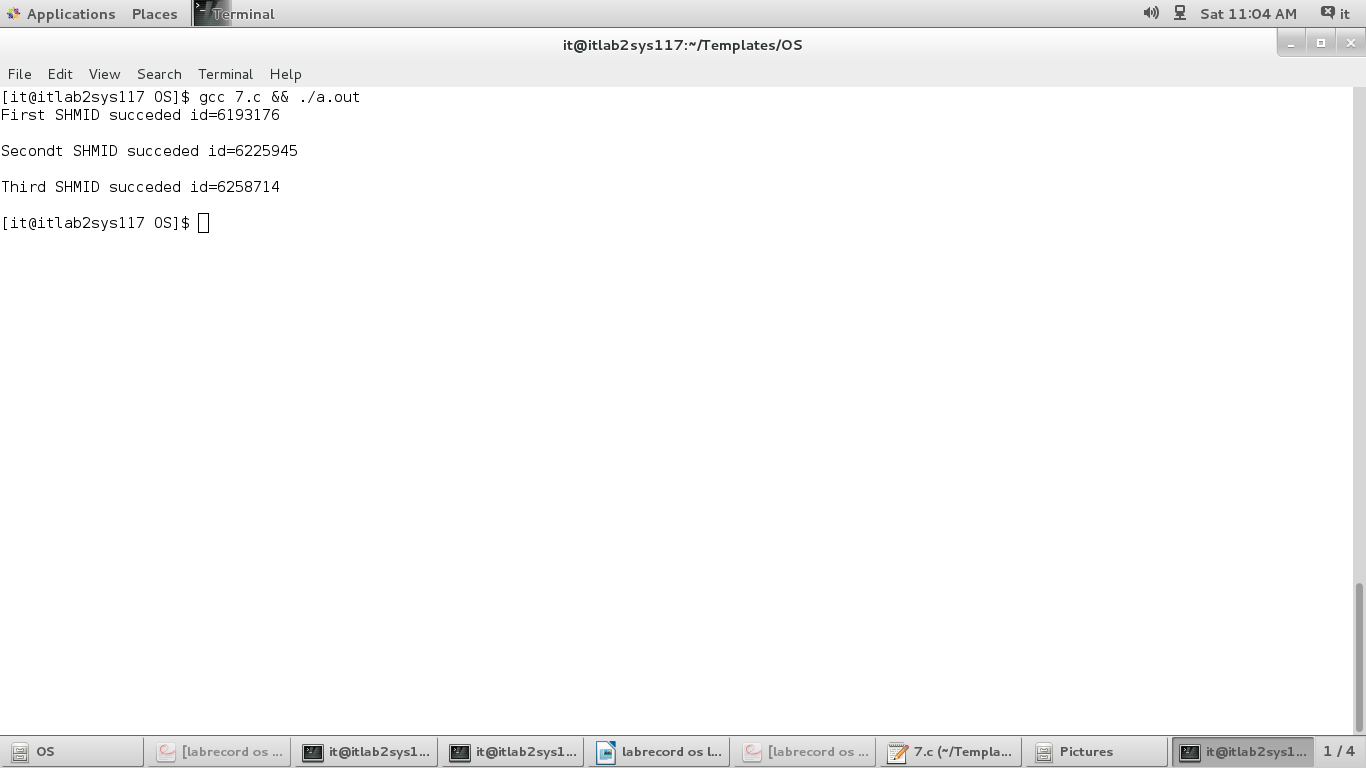
printf("Third SHMID failed\n\n");

else

printf("Third SHMID succeded id=%d\n\n", shmid);

}

**OUTPUT :**



**RESULT**

Thus the program for Interprocess communication using Shared Memory system calls was executed successfully

**Ex.No : 6 IMPLEMENTATION OF SEMAPHORE**

**AIM**

To implement Producer Consumer problem using Semaphore.

**ALGORITHM**

1. Union a variable “mysemun” as integer type to implement semaphore.
2. A buffer “sembuf” variable for producer consumer is written
3. In producer, buffer “sembuf” full condition is checked and if it is full then it is

displayed.

1. In consumer, buffer “sembuf” empty condition is checked and a message displayed.
2. Producer produces an element sequentially from main().and a message displayed.
3. Consumer consumes an element sequentially from main().and a message displayed.
4. The producer consumer produces messages are displayed till the user defines in the

program.

**PROGRAM :**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1, full = 0, empty = 3, x = 0; //mutex is a lock. There are 3 empty buffers and 0 full buffers

main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");

while (n <= 3)

{

printf("\nENTER YOUR CHOICE : ");

scanf("%d", &n);

switch (n)

{

case 1:

if ((mutex == 1) && (empty != 0)) //if lock is available and empty buffers are available

producer();

else

printf("BUFFER IS FULL\n");

break;

case 2:

if ((mutex == 1) && (full != 0)) //if lock is available and full buffers are available

consumer();

else

printf("BUFFER IS EMPTY\n");

break;

case 3:

exit(1);

break;

}

}

}

int wait(int s)//decrement the value

{

return (--s);

}

int signal(int s) //increment the value

{

return (++s);

}

void producer()

{

mutex = wait(mutex);//get the lock

full = signal(full); //increase the number of full buffers

empty = wait(empty); //decrease the number of empty buffers

x++;

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

mutex = signal(mutex);//release the lock

}

void consumer()

{

mutex = wait(mutex);//get lock

full = wait(full); //decrease the number of full buffers

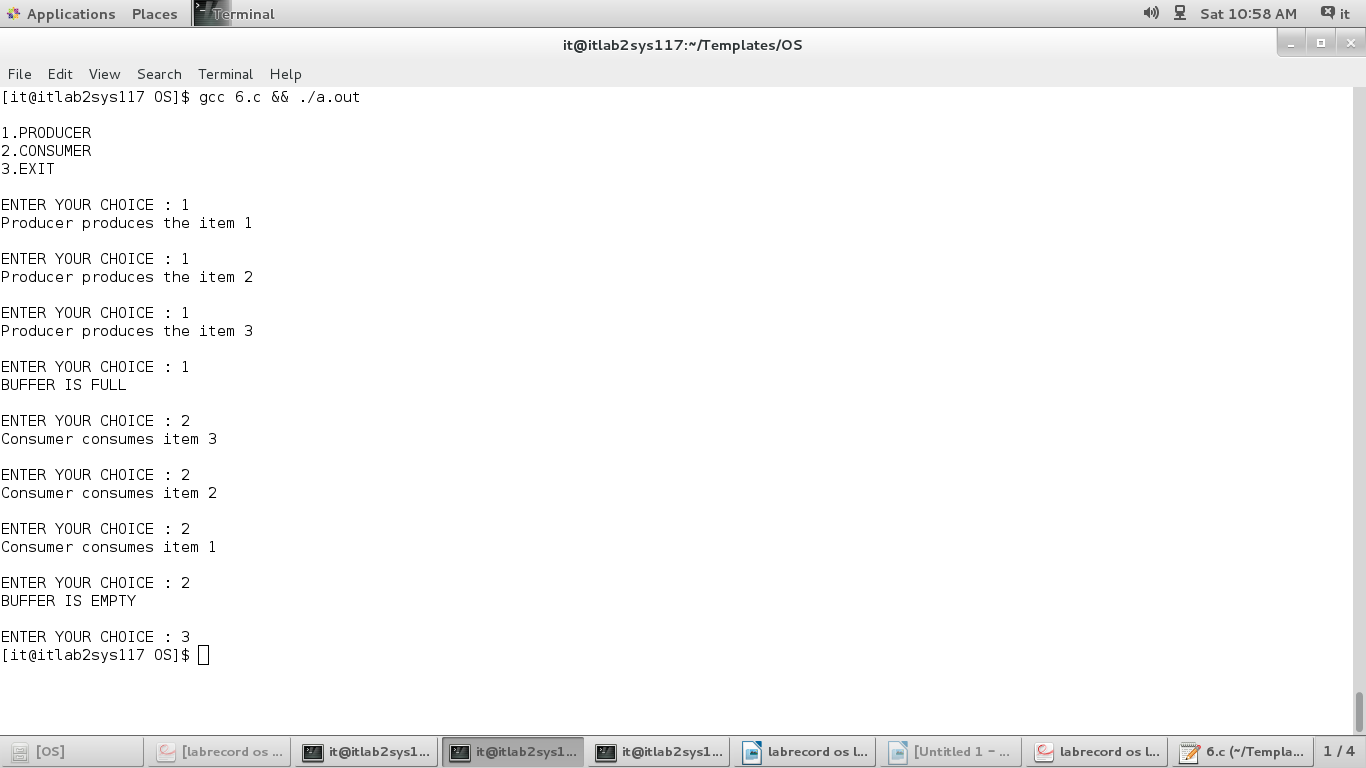
empty = signal(empty); //increase the number of empty buffer

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

x--;

mutex = signal(mutex);//release lock

}

**OUTPUT :**

**RESULT**

Thus the Producer Consumer problem using Semaphore is written and executed successfully.

**Ex: No : 7(i) FCFS SCHEDULING**

**AIM**

To write a C program to implement FCFS scheduling algorithm.

**ALGORITHM**

|  |
| --- |
| 1. Get the number of processes and their burst time. |
| 1. Initialize the waiting time for process 1 and 0. |
| 1. The waiting time for the other processes are calculated as follows: |
| for(i=2;i<=n;i++),wt.p[i]=p[i-1]+bt.p[i-1]. |
| 1. The waiting time of all the processes is summed then average value time is   calculated. |
| 1. The waiting time of each process and average times are displayed. |
|  |

**PROGRAM**

#include <stdio.h>

int main()

{

int c = 0, i, n, bt[10], at[10], wt[10], ft[10]; //bt-burst time; at-arrival time; wt-waiting time;ft-finishing time

int st[10], tat[10]; //st-starting time;tat-turn around time

float awt = 0, atat = 0, rr[10]; //rr-response ratio

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

scanf("%d", &n);

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

{

printf("Enter the arrival time and burst time for the process %d : ", i);

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

}

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

{

st[i] = c;

c = c + bt[i];

wt[i] = st[i] - at[i]; //waiting time of process i=starting time of i – arrival time of i

ft[i] = st[i] + bt[i]; //finishing time of process i= starting time of i + burst time of i

tat[i] = wt[i] + bt[i]; //turnaround time of i =waiting time of i + burst time of i

rr[i] = tat[i] / bt[i];

}

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

{

awt = awt + wt[i];

atat = atat + tat[i];

}

awt = awt / n;

atat = atat / n;

printf("\n\t\t CPU SCHEDULING\n\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\t\t FIRST COME FIRST SERVE\n\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n--------------------------------------------------------------\n");

printf("proc\t at\t bt\t st\t ft\t wt\t tat\t rr\t\n");

printf("--------------------------------------------------------------");

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

{

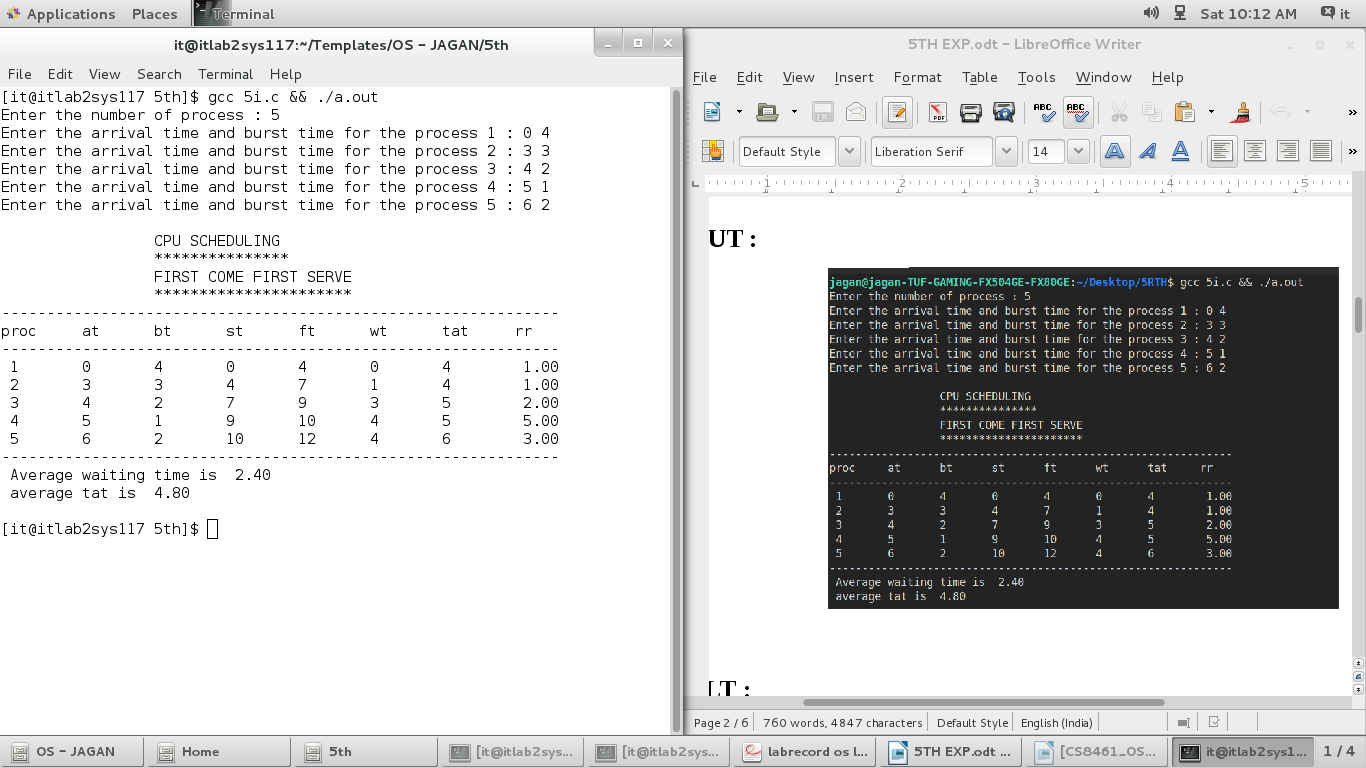
printf("\n %d\t %d\t %d\t %d\t %d\t %d\t %d\t %5.2f", i, at[i], bt[i], st[i], ft[i], wt[i], tat[i], rr[i]);

}

printf("\n--------------------------------------------------------------");

printf("\n Average waiting time is %5.2f\n average tat is %5.2f\n\n", awt, atat);

}



**OUTPUT :**

**RESULT**

Thus the FCFS program for scheduling was executed and verified successfully.

**Ex: No : 7(b) SJF SCHEDULING**

**AIM**

To write a C program to implement Shortest Job First scheduling algorithm.

**ALGORITHM**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Initialize the waiting time for process 1 as 0. | | | |
| 1. The processes are stored according to their burst time. | |  | |
| 1. The waiting time for the processes are calculated as follows:   for(i=2;i<=n;i++).wt.p[i]=p[i=1]+bt.p[i-1]. | |  | |
| 1. The waiting time of all the processes summed and then the average time is   calculated. | |  | |
| 1. The waiting time of each processes and average time are displayed. | |  | |
| **PROGRAM**  #include <stdio.h>  #include <stdlib.h>  int main()  {  int i, n, p[10], st[10], at[10], bt[10], ft[10], wt[10], tt[10];  int nextst, count, minsrt, minpos;  static int iscompleted[10];  float rr[10], awt = 0, att = 0;  printf("\n\t SHORTEST JOB FIRST\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printf("\nEnter the no. of process to be executed : ");  scanf("%d", &n);  printf("\nEnter the process,arrival time and burst time\n");  for (i = 0; i < n; i++)  scanf("%d %d %d", &p[i], &at[i], &bt[i]);  nextst = 0; //next process starting time  for (count = 0; count < n; count++) //Find the process with the minimum burst time among the processes that have arrived and are not completed.   Initialize "shortest\_job" position (minpos) to 0 and "shortest\_time"(minsrt) to a large initial value. Iterate through the processes, checking if they have arrived and if their burst time is smaller than the current "shortest\_time." If so, update "shortest\_job" to the current process.  {  minsrt = 100;  minpos = 0;  for (i = 0; i < n; i++){  if (at[i] <= nextst && iscompleted[i] == 0)  {  if (minsrt > bt[i])  {  minsrt = bt[i];  minpos = i;  }  }  }  //Update the start time, finish time, waiting time, and turnaround time for the selected process. Calculate the response ratio (rr) for the selected process (rr = turnaround time / burst time). Mark the selected process as completed. e. Update the nextst to the finish time of the selected process  i = minpos;  st[i] = nextst;  ft[i] = st[i] + bt[i];  wt[i] = st[i] - at[i];  tt[i] = wt[i] + bt[i];  rr[i] = tt[i] / bt[i];  iscompleted[i] = 1;  nextst = ft[i];  }  printf("\n---------------------------------------");  printf("\nPRO AT bT ST FT WT TT RR \n");  printf("---------------------------------------\n");  for (i = 0; i < n; i++){  printf("%3d %2d %2d", p[i], at[i], bt[i]);  printf(" %3d %2d %2d %2d %4.2f\n", st[i], ft[i], wt[i], tt[i], rr[i]);  }  printf("---------------------------------------");  //Calculate the average waiting time (AWT) and average turnaround time (ATT) by summing up the individual waiting times and turnaround times and dividing by the total number of processes  for (i = 0; i < n; i++){  awt = awt + wt[i];  att = att + tt[i];  }  awt = awt / n;  att = att / n;  printf("\nAverage waiting time is %5.2f", awt);  printf("\nAverage turn around time is %5.2f\n", att);  }  **OUTPUT :** | |  | |
| **RESULT**  Thus the SJF program was executed and verified successfully. | |  | |

**Ex. No : 7(c) PRIORITY SCHEDULING**

**AIM**

To write a C program to implement priority scheduling algorithm.

**ALGORITHM**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Read burst time, waiting time, turnaround time and priority. | | | |
| 1. Initialize the waiting time for process 1 and 0 | |  | |
| 1. Based up on the priority process are arranged. | |  | |
| 1. The waiting time for other processes are calculated based on priority. | |  | |
| 1. The waiting time of all the processes is summed and then the average waiting time is calculated. | |  | |
| 1. The waiting time of each process and average waiting time are displayed based on the priority. | |  | |
|  |  | |  |

**PROGRAM (PRIORITY NON-PREEMPTIVE) :**

#include <stdio.h>

main()

{

int i, n, p[10], st[10], at[10], bt[10], ft[10], wt[10], tt[10], pri[10];

float awt = 0.0, att = 0.0;

int nextst, count, minsrt, minpos;

static iscompleted[10];

float rr[10];

printf("\n\t PRIORITY NON-PREEMPTIVE\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nEnter the no. of process to be executed: ");

scanf("%d", &n);

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

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

{

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

}

nextst = 0;

for (count = 0; count < n; count++)//Find the process with the highest priority (lowest number as highest priority) among the processes that have arrived and are not completed.   Initialize "highest priority job" position (minpos) to 0 and "highest priority"(minsrt) to a large initial value. Iterate through the processes, checking if they have arrived and if their priority is higher than the current "minsrt." If so, update "minsrt" to the current process.

{

minsrt = 100;

minpos = 0;

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

{

if (at[i] <= nextst && iscompleted[i] == 0)

{

if (minsrt > pri[i])

{

minsrt = pri[i];

minpos = i;

}

}

}

i = minpos;

st[i] = nextst;

ft[i] = st[i] + bt[i];

wt[i] = st[i] - at[i];

tt[i] = wt[i] + bt[i];

rr[i] = tt[i] / bt[i];

iscompleted[i] = 1;

nextst = ft[i];

}

printf("\n\t\t CPU SCHEDULING\n\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\t\t PRIORITY NON-PREEMPTIVE\n\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n-----------------------------------------");

printf("\nPRO AT bT PRIORITY ST FT WT TT RR \n");

printf("-----------------------------------------\n");

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

{

printf("%3d %2d %2d %5d", p[i], at[i], bt[i], pri[i]);

printf(" %5d %2d %2d %2d %4.2f\n", st[i], ft[i], wt[i], tt[i], rr[i]);

}

printf("-----------------------------------------");

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

{

awt = awt + wt[i];

att = att + tt[i];

}

awt = awt / n;

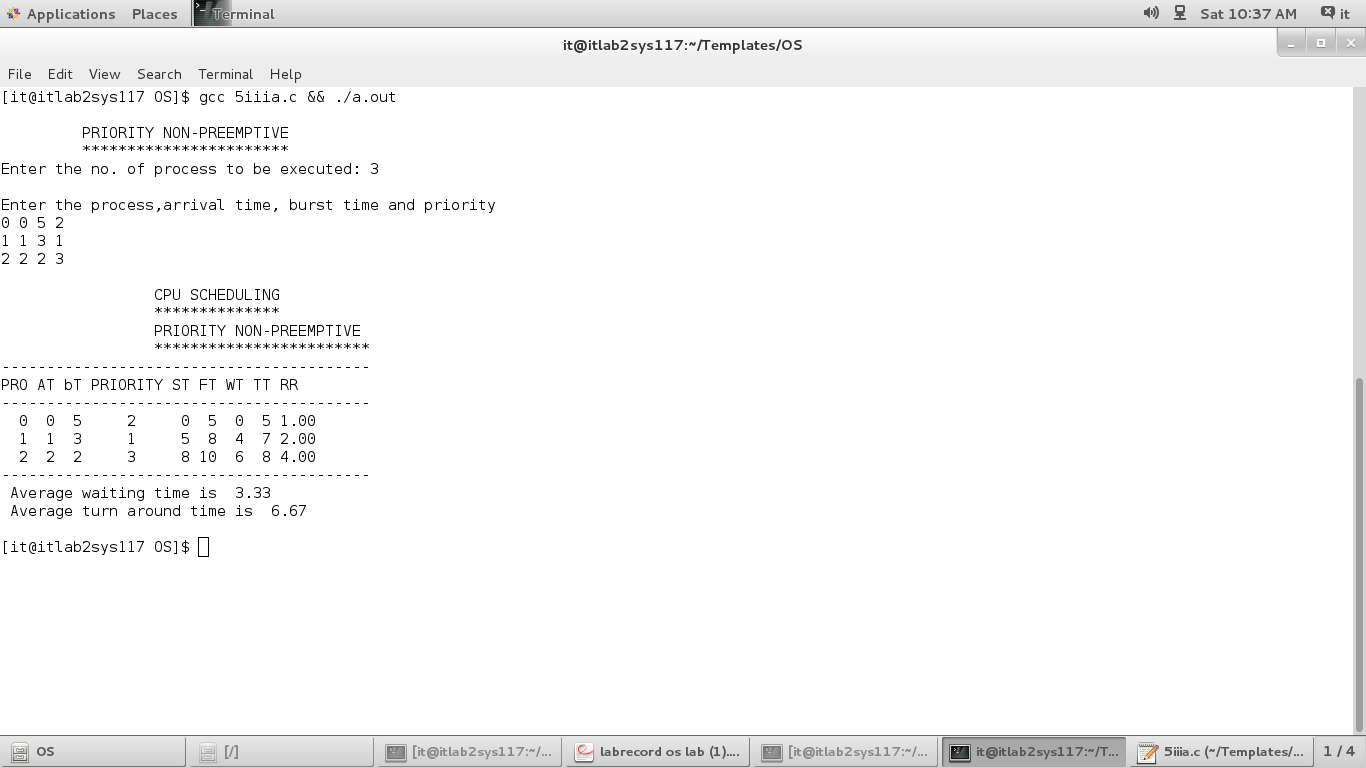
att = att / n;

printf("\n Average waiting time is %5.2f", awt);

printf("\n Average turn around time is %5.2f\n\n", att);

}

**OUTPUT :**



**PROGRAM (PRIORITY PRE-EMPTIVE) :**

#include <stdio.h>

int main()

{

int i, n, pid[10], at[10], srt[10], st[10], ft[10], wt[10], tt[10], pri[10];

int timer, totalsrt, tempsrt[10], minsrt, minpos;

static int iscompleted[10], isstarted[10];

float rr[10], awt, atat;

printf("\n\t PRIORITY PRE-EMPTIVE\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nENTER THE NO.OF PROCESSES TO BE EXECUTED : ");

scanf("%d", &n);

printf("ENTER THE PROCESSES ID,ARRIVAL TIME,BURST TIME AND PRIORITY\n");

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

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

totalsrt = 0;

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

{

totalsrt = totalsrt + srt[i];

tempsrt[i] = srt[i];

}

timer = 0;

while (timer < totalsrt)

{

minsrt = 100;

minpos = 0;

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

{

if (at[i] <= timer && iscompleted[i] == 0)

{

if (minsrt > pri[i])

{

minsrt = pri[i];

minpos = i;

}

}

}

i = minpos;

if (isstarted[i] == 0)

{

st[i] = timer;

wt[i] = st[i] - at[i];

isstarted[i] = 1;

}

srt[i] = srt[i] - 1;

timer = timer + 1;

if (srt[i] == 0)

{

ft[i] = timer;

wt[i] = wt[i] + (ft[i] - (st[i] + tempsrt[i]));

tt[i] = wt[i] + tempsrt[i];

rr[i] = tt[i] / tempsrt[i];

iscompleted[i] = 1;

}

}

printf("\n\t CPU SCHEDULING ALGORITHM\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\t PRIORITY PRE-EMPTIVE\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n--------------------------------------------------");

printf("\nPID AT SRT ST FT WT TT RR PRIORITY\n");

printf("--------------------------------------------------\n");

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

{

printf("%2d %2d %2d ", pid[i], at[i], tempsrt[i]);

printf("%2d %2d %2d %2d %2.2f %3d\n", st[i], ft[i], wt[i], tt[i], rr[i], pri[i]);

}

printf("--------------------------------------------------\n");

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

{

awt = awt + wt[i];

atat = atat + tt[i];

}

atat = atat / n;

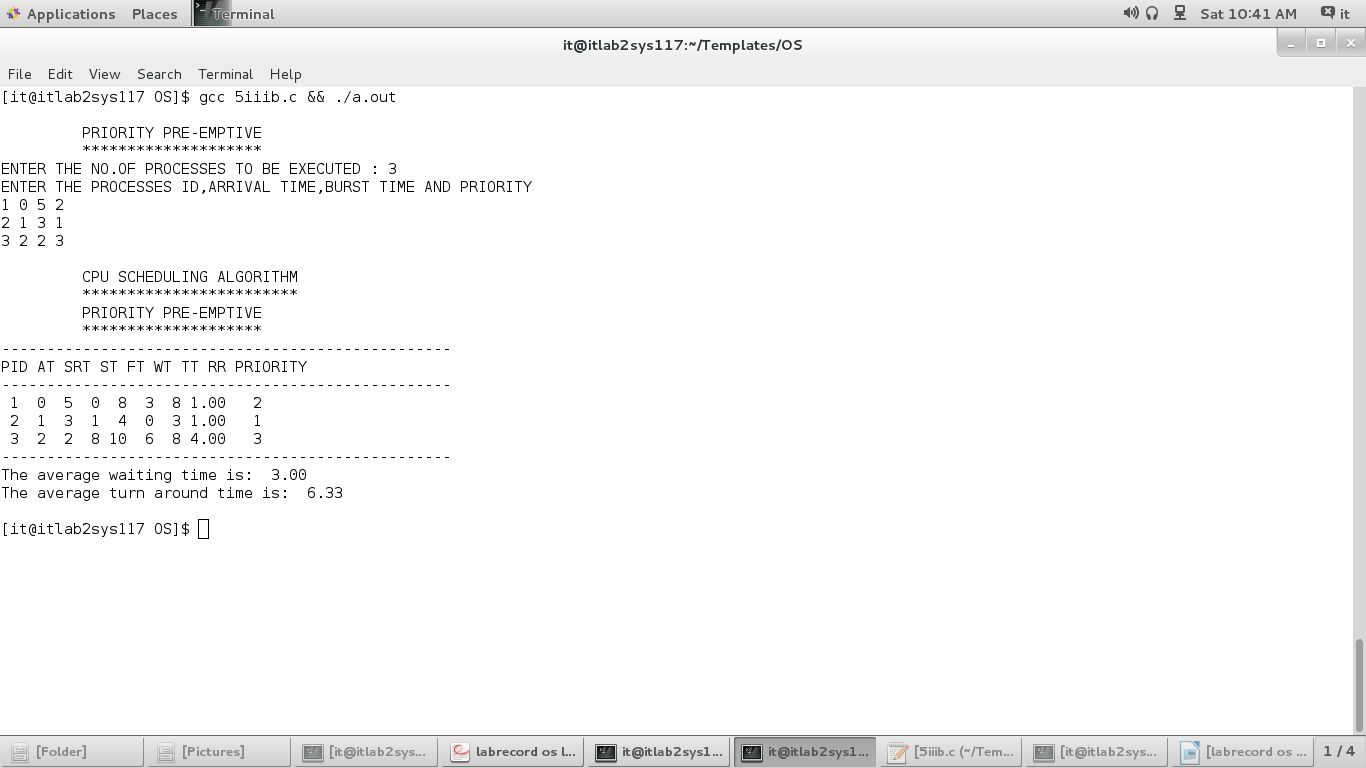
awt = awt / n;

printf("The average waiting time is: %5.2f\n", awt);

printf("The average turn around time is: %5.2f\n\n", atat);

}

**OUTPUT :**

****

**RESULT**

Thus the priority scheduling program was executed and verified successfully.

**Ex: No : 7(d) ROUND ROBIN SCHEDULING**

**AIM**

To write a C program to implement Round Robin Scheduling algorithm.

**ALGORITHM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Initialize the array for Round Robin circular queue as ‘0’. | | | |  |
| 1. The burst time of each process is divided and the quotients are stored on the round robin array. | |  | |  |
| 1. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling. | |  | |  |
| 1. The waiting time for each process and average times are displayed. | |  | |  |
|  | **Logic:**   * Read the number of processes (n) and the time quantum (tq) from the user. * Input process information for each process, including process ID (PRO), arrival time (AT), and burst time (BUT). * Also, calculate the total burst time (totalsrt) and create a temporary array (tempsrt) to store the initial burst times. * Initialize the queue (queue), front (f), rear (r), and count variables for the ready queue. Also, initialize the timer to 0. Enqueue processes that arrive at time 0 into the ready queue and mark them as entered (isentered) with count increments. * Enter the main scheduling loop, which continues until the timer reaches the total remaining burst time (totalsrt). Dequeue a process from the front of the ready queue (queue[f]) and select it for execution.If the selected process is starting for the first time (isstarted is 0), record its start time (ST) and calculate its waiting time (WT). * Execute the process for a time quantum (tq) or until its burst time (BUT) is less than tq, whichever comes first. Update the timer accordingly. If the process has completed its burst time (BUT becomes 0), record its finish time (FT), calculate its waiting time (WT), turnaround time (TT), and response ratio (RR), and mark it as completed (iscompleted).Check for processes that have arrived and have not entered the ready queue (isentered is 0). * Enqueue them into the ready queue with count increments.If the selected process is not completed, enqueue it back into the ready queue Repeat the scheduling loop until the timer reaches totalsrt.   **PROGRAM :**  #include <stdio.h>  main()  {  int n, i, pro[10], at[10], srt[10], st[10], ft[10], wt[10], tt[10];  static int iscompleted[10], isstarted[10], isentered[10];  int queue[10], f, r, count, tq, j, timer, totalsrt, tempsrt[10];  float rr[10], awt = 0, atat = 0;  printf("\n\t ROUND ROBIN");  printf("\nEnter the no. of process :");  scanf("%d", &n);  printf("\nEnter the value for Time Quantum :");  scanf("%d", &tq);  printf("\nEnter the process id for n process:\n");  for (i = 0; i < n; i++)  {  scanf("%d", &pro[i]);  }  printf("\nEnter the arrival time for n process :\n");  for (i = 0; i < n; i++)  {  scanf("%d", &at[i]);  }  printf("\nEnter the Burst time for n process:\n");  for (i = 0; i < n; i++)  {  scanf("%d", &srt[i]);  }  totalsrt = 0; //totalsrt represents the total burst time of all processes  for (i = 0; i < n; i++)  {  totalsrt = totalsrt + srt[i];  tempsrt[i] = srt[i];  }  f = 0; //front  r = -1; //rear  count = 0;  timer = 0;  for (i = 0; i < n; i++)  {  if (at[i] == 0)  {  r = (r + 1) % n;  queue[r] = i;  isentered[i] = 1;  count = count + 1;  }  }  while (timer < totalsrt)  {  j = queue[f];  f = (f + 1) % n;  if (isstarted[j] == 0)  {  st[j] = timer;  wt[j] = st[j] - at[j];  isstarted[j] = 1;  }  if (srt[j] >= tq)  {  timer = timer + tq;  srt[j] = srt[j] - tq;  }  else  {  timer = timer + srt[j];  srt[j] = srt[j] - srt[j];  }  if (srt[j] == 0)  {  ft[j] = timer;  wt[j] = wt[j] + (ft[j] - (st[j] + tempsrt[j]));  tt[j] = wt[j] + tempsrt[j];  rr[j] = (float)tt[j] / tempsrt[j];  iscompleted[j] = 1;  }  for (i = 0; i < n && count < n; i++)  {  if (at[i] <= timer && isentered[i] == 0)  {  r = (r + 1) % n;  queue[r] = i;  isentered[i] = 1;  count = count + 1;  }  }  if (iscompleted[j] == 0)  {  r = (r + 1) % n;  queue[r] = j;  }  }  printf("\n\t CPU SCHEDULING\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printf("\n\t ROUND ROBIN\n\t \*\*\*\*\*\*\*\*\*\*\*\n");  printf(" ------------------------------------------- \n");  printf("PRO AT BUT ST FT WT TT RR");  printf("\n------------------------------------------- \n");  for (i = 0; i < n; i++)  {  printf("%3d %2d %2d", pro[i], at[i], tempsrt[i]);  printf("%3d %3d %2d", st[i], ft[i], wt[i]);  printf("%3d %4.2f\n", tt[i], rr[i]);  }  printf(" -------------------------------------------");  for (i = 0; i < n; i++)  {  awt = awt + wt[i];  atat = atat + tt[i];  }  awt = awt / n;  atat = atat / n;  printf("\nAvg waiting time is %5.2f ", awt);  printf("\nAvg turn around time is %5.2f\n", atat);  }  **OUTPUT :** | |  | |

**RESULT**

Thus the Round Robin scheduling program was executed and verified successfully.

**Ex.No : 8 BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE**

**AIM**

To implement Deadlock Avoidance by using Banker’s Algorithm in C.

**ALGORITHM**

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether it’s possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.
9. Stop the program.

// Banker's Algorithm

#include <stdio.h>

**int** main()

{

    // P0, P1, P2, P3, P4 are the Process names here

**int** n, m, i, j, k;

    n = 5;                         // Number of processes

    m = 3;                         // Number of resources

**int** alloc[5][3] = {{0, 1, 0},  // P0 // Allocation Matrix

                       {2, 0, 0},  // P1

                       {3, 0, 2},  // P2

                       {2, 1, 1},  // P3

                       {0, 0, 2}}; // P4

**int** max[5][3] = {{7, 5, 3},  // P0 // MAX Matrix

                     {3, 2, 2},  // P1

                     {9, 0, 2},  // P2

                     {2, 2, 2},  // P3

                     {4, 3, 3}}; // P4

**int** avail[3] = {3, 3, 2}; // Available Resources

**int** f[n], ans[n], ind = 0;

**for** (k = 0; k < n; k++)  //initialise f for all process to 0

    {

        f[k] = 0;

    }

**int** need[n][m];

**for** (i = 0; i < n; i++)  //calculate need matrix

    {

**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)  //if process is not executed

            {

**int** flag = 0;

**for** (j = 0; j < m; j++)

{

                    if (need[i][j] > avail[j]) //if need is greater than available, process cannot execute & flag will be reset to 1

{

                        flag = 1;

                         break;

                    }

                }

                if (flag == 0) //flag=0 indicates that need is less than available & process can execute

{

                    ans[ind++] = i; //ans array will contain the process number that completes its execution

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

                        avail[y] += alloc[i][y]; //the allocated resources of executed process will be released and added with available resources

                    f[i] = 1;

                }

            }

        }

    }

      int flag = 1;

      for(int i=0;i<n;i++)

    {

      if(f[i]==0)

      {

        flag=0;

         printf("The following system is not safe and deadlock is possible");

        break;

      }

    }

      if(flag==1)

    {

      printf("Following is the SAFE Sequence and deadlock is not possible\n");

      for (i = 0; i < n - 1; i++)

        printf(" P%d ->", ans[i]);

      printf(" P%d", ans[n - 1]);

    }

    return (0);

}

**Ex. No: 9 DEADLOCK DETECTION ALGORITHM**

**AIM**

To write a C program to implement DeadLock Detection using Banker’s Algorithm.

**ALGORITHM**

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.

**PROGRAM :**

#include <stdio.h>

#include <stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n, r;

void input();

void show();

void cal();

int main() {

int i, j;

printf("\*\*\*\*\*\*\*\*\*\*Deadlock Detection Algo \*\*\*\*\*\*\*\*\*\*\*\*\n");

input();

show();

cal();

return 0;

}

void input() {

int i, j;

printf("Enter the no of Processes : ");

scanf("%d", &n);

printf("Enter the no of resource instances : ");

scanf("%d", &r);

printf("Enter the Max Matrix\n");

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

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

{

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

}

}

printf("Enter the Allocation Matrix\n");

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

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

{

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

}

}

printf("Enter the available Resources\n");

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

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

}

}

void show()

{

int i, j;

printf("Process\t Allocation\t Max\t Available\t");

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

printf("\nP%d\t ", i + 1);

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

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

printf("\t");

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

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

}

printf("\t");

if (i == 0) {

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

printf("%d ", avail[j]);

}

}

}

void cal() {

int finish[100], temp, need[100][100], flag = 1, k, c1 = 0;

int dead[100];

int safe[100];

int i, j;

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

finish[i] = 0;

//find need matrix

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

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

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

}

}

while (flag) {

flag = 0;

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

int c = 0;

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

if ((finish[i] == 0) && (need[i][j] <= avail[j])) {

c++;

if (c == r) {

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

avail[k] += alloc[i][j];

finish[i] = 1;

flag = 1;

}

//printf("\nP%d",i);

if (finish[i] == 1) {

i = n;

}

}

}

}

}

}

j = 0;

flag = 0;

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

if (finish[i] == 0) {

dead[j] = i;

j++;

flag = 1;

}

}

if (flag == 1) {

printf("\n\nSystem is in Deadlock and the Deadlock process are\n");

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

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

}

}

else {

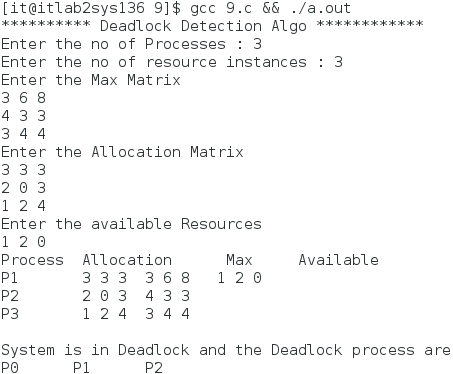
printf("\nNo Deadlock Occur");

}

printf("\n");

}

**OUTPUT :**



**RESULT**

Thus the C program for Dead Lock Detection was executed successfully.

**Ex. No : 10(a) FIRST FIT MEMORY ALLOCATION**

**AIM**

To write a C program to implement First fit Memory allocation technique

**ALGORITHM**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Enter the number of blocks and number of files. | | | |
| 1. Enter the size of the blocks. | |  | |
| 1. Enter the size of the files. | |  | |
| 1. First fit chooses the first available block that is large enough. | |  | |
| 1. Display File number, File size, Occupied block number and size. | |  | |
|  |  | |  |

**PROGRAM: (FIRST FIT MEMORY MANAGEMENT) :**

#include <stdio.h>

void implementFirstFit(int blockSize[], int blocks, int processSize[], int processes)

{

// This will store the block id of the allocated block to a process

int allocate[processes];

int occupied[blocks];

// initially assigning -1 to all allocation indexes means nothing is allocated currently

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

{

allocate[i] = -1;

}

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

occupied[i] = 0; //initially all blocks will be free

}

// take each process one by one and find first block that can accomodate it

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

{

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

{

if (!occupied[j] && blockSize[j] >= processSize[i])

{

// allocate block j to p[i] process

allocate[i] = j;

occupied[j] = 1;

break;

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocate[i] != -1)

printf("%d\n",allocate[i] + 1);

else

printf("Not Allocated\n");

}

}

void main()

{

int blockSize[] = {30, 5, 10};

int processSize[] = {10, 6, 9};

int m = sizeof(blockSize)/sizeof(blockSize[0]);

int n = sizeof(processSize)/sizeof(processSize[0]);

implementFirstFit(blockSize, m, processSize, n);

}

Output

Process No. Process Size Block no.

1 10 1

2 6 3

3 9 Not Allocated

**RESULT**

Thus First fit Memory allocation technique was executed successfully.

**Ex. No : 10(b) WORST FIT MEMORY ALLOCATION**

**AIM**

To write a C program to implement Worst fit Memory allocation technique

**ALGORITHM**

|  |
| --- |
| 1. Enter the number of blocks and number of files. |
| 1. Enter the size of the blocks. |
| 1. Enter the size of the files. |
| 1. Worst fit occupies the largest available block. |
| 1. Display File number, File size, occupied block number and size. |

**PROGRAM: (WORST FIT MEMORY MANAGEMENT) :**

#include <stdio.h>

void implementWorstFit(int blockSize[], int blocks, int processSize[], int processes)

{

// This will store the block id of the allocated block to a process

int allocation[processes];

int occupied[blocks];

// initially assigning -1 to all allocation indexes means nothing is allocated currently

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

allocation[i] = -1;

}

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

occupied[i] = 0;

}

// pick each process and find suitable block according to its size ad assign to it

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

{

int indexPlaced = -1;

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

{

// if not occupied and block size is large enough

if(blockSize[j] >= processSize[i] && !occupied[j])

{

// place it at the first block fit to accomodate process

if (indexPlaced == -1)

indexPlaced = j;

// if any future block is larger than the current block where process is placed, change the block and thus indexPlaced

else if (blockSize[indexPlaced] < blockSize[j])

indexPlaced = j;

}

}

// If we were successfully able to find block for the process

if (indexPlaced != -1)

{

// allocate this block j to process p[i]

allocation[i] = indexPlaced;

// make the status of the block as occupied

occupied[indexPlaced] = 1;

// Reduce available memory for the block

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

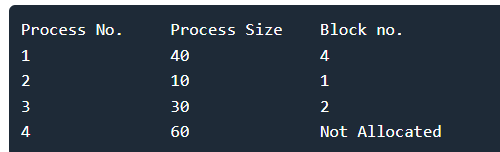
int processes = sizeof(processSize)/sizeof(processSize[0]);

implementWorstFit(blockSize, blocks, processSize, processes);

return 0;

}

**OUTPUT :**



**RESULT**

Thus Worst fit Memory allocation technique was executed and verified successfully.

**Ex. No : 10(c) BEST FIT MEMORY ALLOCATION**

**AIM**

To write a C program to implement Best fit Memory allocation technique

**ALGORITHM**

|  |  |
| --- | --- |
| 1. Enter the number of blocks and number of files. | |
| 1. Enter the size of the blocks. |  |
| 1. Enter the size of the files. |  |
| 1. Best fit chooses the block that is closest in size to the request. |  |
| 1. Display File number, File size, occupied block number and size. |  |

**PROGRAM: (BEST FIT MEMORY MANAGEMENT)**

#include <stdio.h>

void implementBestFit(int blockSize[], int blocks, int processSize[], int proccesses)

{

// This will store the block id of the allocated block to a process

int allocation[proccesses];

int occupied[blocks];

// initially assigning -1 to all allocation indexes means nothing is allocated currently

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

allocation[i] = -1;

}

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

occupied[i] = 0;

}

// pick each process and find suitable blocks according to its size ad assign to it

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

{

int indexPlaced = -1;

for (int j = 0; j < blocks; j++) {

if (blockSize[j] >= processSize[i] && !occupied[j])

{

// place it at the first block fit to accomodate process

if (indexPlaced == -1)

indexPlaced = j;

// if any future block is smalller than the current block where process is placed, change the block and thus indexPlaced this reduces the wastage thus best fit

else if (blockSize[j] < blockSize[indexPlaced])

indexPlaced = j;

}

}

// If we were successfully able to find block for the process

if (indexPlaced != -1)

{

// allocate this block j to process p[i]

allocation[i] = indexPlaced;

// make the status of the block as occupied

occupied[indexPlaced] = 1;

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

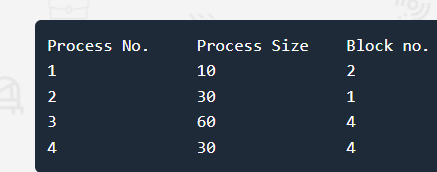
int proccesses = sizeof(processSize)/sizeof(processSize[0]);

implementBestFit(blockSize, blocks, processSize, proccesses);

return 0 ;

}

**OUTPUT :**



**RESULT**

Thus Best fit Memory allocation technique was executed and verified successfully.

**Ex.No: 11 PAGING TECHNIQUE OF MEMORY MANAGEMENT**

**AIM**

To implement the Memory management policy- Paging using C.

**ALGORITHM**

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages - Logical memory is broken into fixed - sized blocks.

Step 3: Frames – Physical memory is broken into fixed – sized blocks.

Step 4: Calculate the physical address using the following

Physical address = ( Frame number \* Frame size ) + offset

Step 5: Display the physical address.

**PROGRAM :**

#include<stdio.h>

#include<conio.h>

main()

{

 int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

 int s[10], fno[10][20];

printf("\nEnter the logical memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);

nop = ms/ps; //nop – number of pages

printf("\nThe no. of pages available in memory are -- %d ",nop);

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

 scanf("%d",&np);

rempages = nop; //initialize remaining pages to number of pages

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

{

printf("\nEnter no. of pages required for p[%d]-- ",i);

 scanf("%d",&s[i]);

if(s[i] >rempages) //if required pages for a process is greater than remaining pages

{

printf("\nMemory is Full");

break;

}

rempages = rempages - s[i]; //update remaining process by subtracting the pages allocated to a particular process i

printf("\nEnter pagetable for p[%d] --- ",i);

 for(j=0;j<s[i];j++)

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

}

printf("\nEnter Logical Address to find Physical Address ");

printf("\nEnter process no. and pagenumber and offset -- ");

scanf("%d %d %d",&x,&y, &offset);

if(x>np || y>=s[i] || offset>=ps) //x-process no, y- page number, if process no > no of process (or) page number >= allocated pages (or) offset >= pagesize

printf("\nInvalid Process or Page Number or offset");

else

{

pa=fno[x][y]\*ps+offset; //calculation of physical address

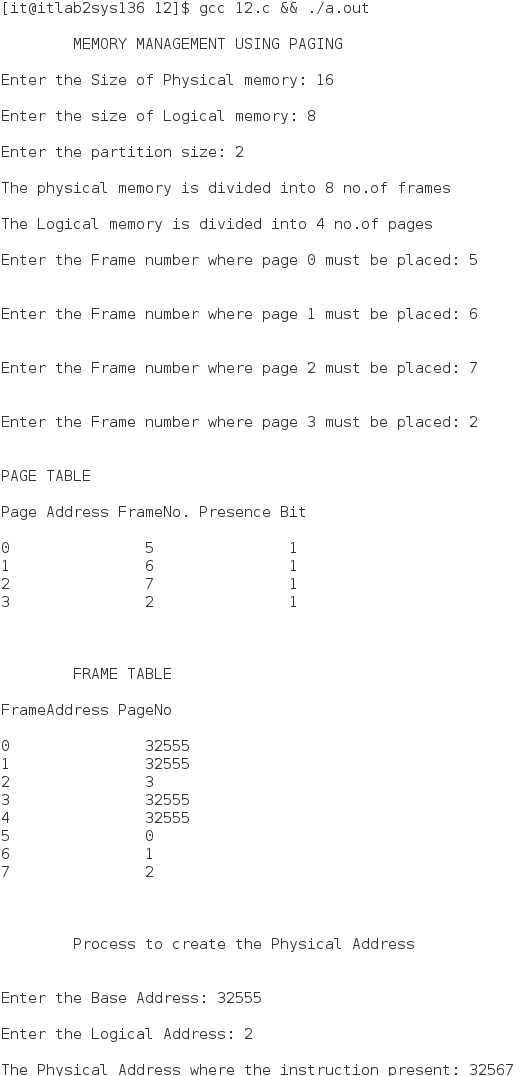
printf("\nThe Physical Address is -- %d",pa);

}

getch();

}

**OUTPUT** :



**RESULT**

Thus the Memory management policy- Paging was executed successfully.

**Ex. No: 12(a) FIFO PAGE REPLACEMENT ALGORITHM**

A**IM**

To write a C program to implement FIFO page replacement algorithm

**ALGORITHM**

1. Declare the size with respect to page length

3. Check the need of replacement from the page to memory

4. Check the need of replacement from old page to new page in memory

5. Form a queue to hold all pages

6. Insert the page require memory into the queue

7. Check for bad replacement and page fault

8. Get the number of processes to be inserted

9. Display the values.

**PROGRAM :**

#include <stdio.h>

int main()

{

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

printf("ENTER THE NUMBER OF PAGES : ");

scanf("%d", &n);

printf("\nENTER THE REF STRING : ");

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++) //Initially all the frames will be empty and the value will be -1

frame[i] = -1;

j = 0;

printf("\tRef string\t Page 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]) //checks whether the reference string is already in frame; if already present, avail will be 1

avail = 1;

if (avail == 0) //if reference string is not present, we have to find a empty space or replace existing page in FIFO order

{

frame[j] = a[i];

j = (j + 1) % no; //to find the page for replacement according to FIFO

count++; //count of page fault

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

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

}

printf("\n");

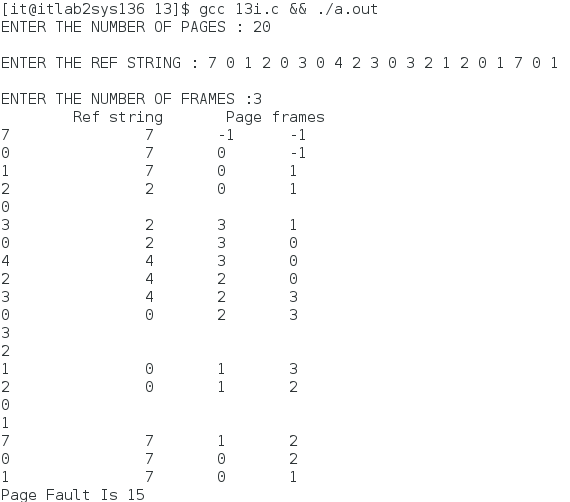
}

printf("Page Fault Is %d\n", count);

return 0;

}

**OUTPUT :**



**RESULT**

Thus the program for Page Replacement Algorithm using FIFO is written and executed successfully.

**Ex. No: 12(b) LRU PAGE REPLACEMENT ALGORITHM**

**AIM**

To implement Least Recently Used Page Replacement algorithm in C.

**ALGORITHM**

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queue

Step 3: Now the new page is inserted at the tail of the queue

Step 4: Create a stack

Step 5: When the page fault occurs replace page present at the bottom of the stack

**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");

}

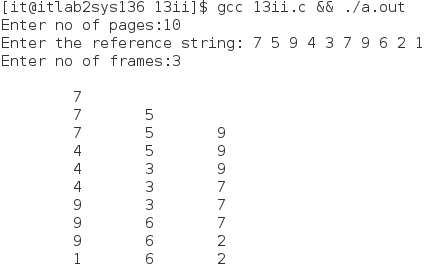
}

}

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

}

**OUTPUT :**



**RESULT**

Thus the program for LRU Page Replacement Algorithm is written and executed successfully.

**Ex No: 12 (c)**  **LFU PAGE REPLACEMENT ALGORITHM**

**AIM**

To implement LFU page replacement technique.

**ALGORITHM**

1. Read Number Of Pages And Frames

2. Read Each Page Value

3. Search For Page In The Frames

4. If Not Available Allocate Free Frame

5. If No Frames Is Free Repalce The Page With The Page That Is Least frequently used.

6. Print Page Number Of Page Faults

**PROGRAM :**

#include <stdio.h>

int i, j, nof, nor, flag = 0, ref[50], frm[50], pf = 0, victim = -1;

int recent[10], optcal[50], count = 0;

int optvictim();

int main()

{

printf("................................");

printf("\nOPTIMAL PAGE REPLACEMENT ALGORITHN");

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

printf("\nEnter the no.of frames : ");

scanf("%d", &nof);

printf("Enter the no.of reference string : ");

scanf("%d", &nor);

printf("Enter the reference string : ");

for (i = 0; i < nor; i++)

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

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

printf("\nOPTIMAL PAGE REPLACEMENT ALGORITHM");

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

printf("\n\nThe Given string");

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

for (i = 0; i < nor; i++)

printf("%3d", ref[i]);

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

{

frm[i] = -1;

optcal[i] = 0;

}

for (i = 0; i < 10; i++)

recent[i] = 0;

printf("\n");

for (i = 0; i < nor; i++)

{

flag = 0;

printf("\nref no %d ->\t", ref[i]);

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

{

if (frm[j] == ref[i])

{

flag = 1;

break;

}

}

if (flag == 0)

{

count++;

if (count <= nof)

victim++;

else

victim = optvictim(i);

pf++;

frm[victim] = ref[i];

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

printf("%4d", frm[j]);

}

}

printf("\nNumber of page faults : %d\n\n", pf);

}

int optvictim(int index)

{

int i, j, temp, notfound;

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

{

notfound = 1;

for (j = index; j < nor; j++)

if (frm[i] == ref[j])

{

notfound = 0;

optcal[i] = j;

break;

}

if (notfound == 1)

return i;

}

temp = optcal[0];

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

if (temp < optcal[i])

temp = optcal[i];

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

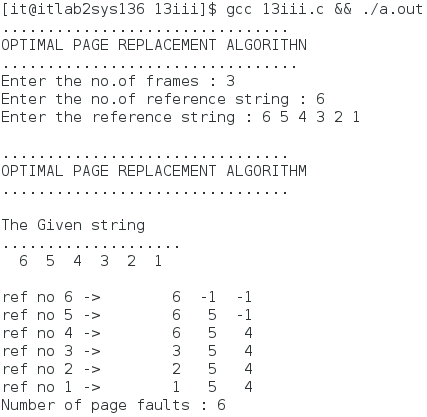
if (frm[temp] == frm[i])

return i;

return 0;

}

**OUTPUT :**



**RESULT**

Thus the program for LFU was executed successfully.

**FILE ORGANIZATION TECHNIQUES**

**Ex No: 13 (a)**  **SINGLE LEVEL DIRECTORY**

**AIM**

To write a c program to simulate Single level directory structure.

**ALGORITHM**

Start the program.

Get the number of main directories to be created.

Get the name of the directory and size of the directory.

Simulate the directory level and display the structure.

Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int master,s[20];

char f[20][20][20];

char d[20][20];

int i,j;

//get the number of directories (Folders) and its names and sizes

printf("Enter number of directorios:");

scanf("%d",&master);

printf("Enter names of directories:");

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

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

printf("Enter size of directories:");

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

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

//for each directories, get the name of the files stored inside it

printf("Enter the file names :");

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

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

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

printf("\n");

printf(" directory\tsize\tfilenames\n");

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

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

{

printf("%s\t\t%2d\t",d[i],s[i]);

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

printf("%s\n\t\t\t",f[i][j]);

printf("\n"); }

printf("\t\n");

}

**OUTPUT**

[user@sys108 ~]$ vi singlelevel.c

[user@sys108 ~]$ cc singlelevel.c

[user@sys108 ~]$ ./a.out

Enter number of directorios:2

Enter names of directories: hai hello

Enter size of directories:2 2

Enter the file names :sample test fibonacci fact

directory size filenames

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hai 2 sample

test

hello 2 fibonacci

fact

**RESULT**

Thus the program to simulate Single level directory structure is executed successfully.

**Ex.No: 14(b)** **TWO LEVEL DIRECTORY**

**AIM**

To write a C program to simulate Two Level directory structures.

**ALGORITHM**

1. Start the program.
2. Get the number of main directories to be created, name of the directory and size of the

directory.

1. Get the number of sub directories to be created, name of the directory and size of the

directory.

1. Simulate the directory level and display the structure.
2. Stop the program.

**PROGRAM**

#include<stdio.h>

struct st

{

char dname[10];

char sdname[10][10];

char fname[10][10][10];

int ds,sds[10];

}dir[10];

void main()

{

int i,j,k,n;

//get the number of directories (Folders) and its names and sizes

printf("Enter number of directories:");

scanf("%d",&n);

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

{

printf("Enter directory %d names:",i+1);

scanf("%s",&dir[i].dname);

printf("Enter size of directories:");

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

//for each directories, get the name of the sub-directories stored inside it

for(j=0;j<dir[i].ds;j++)

{

printf("Enter subdirectory name and size:");

scanf("%s",&dir[i].sdname[j]);

scanf("%d",&dir[i].sds[j]);

//for each sub- directories, get the name of the files stored inside it

for(k=0;k<dir[i].sds[j];k++)

{

printf("Enter file name:");

scanf("%s",&dir[i].fname[j][k]);

}

}

}

printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");

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

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

{

printf("%s\t\t%d",dir[i].dname,dir[i].ds);

for(j=0;j<dir[i].ds;j++)

{

printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);

for(k=0;k<dir[i].sds[j];k++)

printf("%s\t",dir[i].fname[j][k]);

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

}

printf("\n"); }

}

**OUTPUT**

[user@sys108 ~]$ vi twolevel.c

[user@sys108 ~]$ cc twolevel.c

[user@sys108 ~]$ ./a.out

Enter number of directories:2

Enter directory 1 names:colleges

Enter size of directories:1

Enter subdirectory name and size:deemed 1

Enter file name:stjosephs

Enter directory 2 names:companies

Enter size of directories:1

Enter subdirectory name and size:MNC 1

Enter file name:CTS

dirname size subdirname size files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

colleges 1 affiliated 1 stjosephs

companies 1 MNC 1 CTS

**RESULT**

Thus the program to simulate Two level directory structure is executed successfully.

**Ex.No:14(c) HIERARCHICAL & DAG**

**AIM**

To write a C program to implement Hierarchical and DAG.

**ALGORITHM**

1. Start the program.
2. Get the number of main directories to be created, name of the directory and size of the directory.
3. Get the number of sub directories to be created, name of the directory and size of the directory.
4. Get the name of the file to be shared between directories.
5. Simulate the directed acyclic graph and display the structure.
6. Stop the program.

**PROGRAM**

#include <stdio.h>#include <stdlib.h>#include <time.h>#define MIN\_PER\_RANK 1 #define MAX\_PER\_RANK 5#define MIN\_RANKS 3 #define MAX\_RANKS 5#define PERCENT 30 int main (void){ int i, j, k,nodes = 0; srand (time (NULL)); int ranks = MIN\_RANKS+ (rand () % (MAX\_RANKS - MIN\_RANKS + 1)); printf ("digraph {\n");for (i = 0; i < ranks; i++) { int new\_nodes = MIN\_PER\_RANK + (rand () % (MAX\_PER\_RANK - MIN\_PER\_RANK + 1)); for (j = 0; j < nodes; j++) for (k = 0; k < new\_nodes; k++) if ( (rand () % 100) < PERCENT) printf (" %d -> %d;\n", j, k + nodes); nodes += new\_nodes; } printf ("}\n"); return 0;}

**OUTPUT**

[user@sys108 ~]$ cc dgraph.c

[user@sys108 ~]$ ./a.out

digraph {

0 -> 5;

1 -> 6;

2 -> 5;

2 -> 8;

3 -> 5;

3 -> 8;

4 -> 6;

4 -> 7;

1 -> 9;

3 -> 9;

6 -> 9;

**RESULT**

Thus the program to simulate Directed Acyclic Graph and hierarchial structure is executed successfully.

**Ex.No:14(a) FCFS DISK SCHEDULING**

**AIM**  
 To write a c program to simulate FCFS disk scheduling.

**ALGORITHM**

1. Enter current position.
2. Enter number of requests
3. Enter the request order.
4. Calculate the absolute value.
5. Calculate and display total head movement.

**PROGRAM**

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,n,req[50],mov=0,cp;

printf("enter the current position\n"); //current disk head position

scanf("%d",&cp);

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

scanf("%d",&n);

printf("enter the request order\n");

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

{

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

}

mov=mov+abs(cp-req[0]); // calculate the number of movements from initial position to first request

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

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

{

mov=mov+abs(req[i]-req[i-1]); //calculate the total number of movements by servicing all requests

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

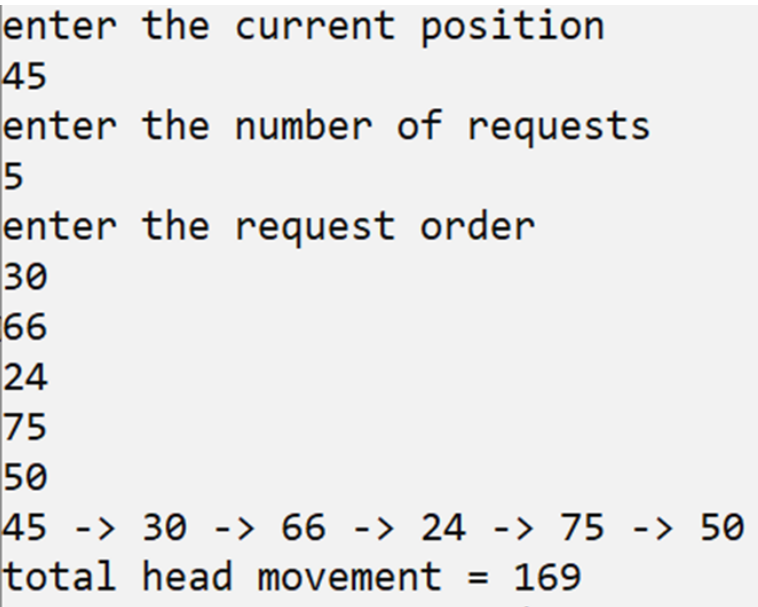
}

printf("\n");

printf("total head movement = %d\n",mov);

}

**OUTPUT**



**RESULT**

Thus a C program to simulate FCFS strategy is executed successfully.

**Ex.No: 14(b) SSTF DISK SCHEDULING**

**AIM**  
 To write a C program to simulate SSTF disk scheduling.

**ALGORITHM**

1. Get the index block number and number of files in the index block as input from user.
2. Get the file numbers (i.e referred block numbers holding file) as input .
3. Check whether that the input block number is already allocated if so print block allocated.
4. Else increase the count and allocate the file.
5. Continue the loop to enter another index block.

**PROGRAM**

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;

printf("enter the current position\n");

scanf("%d",&cp);

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

scanf("%d",&n);

cp1=cp;

printf("enter the request order\n");

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

{

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

}

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

{

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

{

index[i]=abs(cp-req[i]); // calculate distance of each request from current position

}

// to find the nearest request

min=index[0];

mini=0;

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

{

if(min>index[i])

{

min=index[i];

mini=i;

}

}

a[j]=req[mini];

j++;

cp=req[mini]; // change the current position value to next request

req[mini]=999;

} // the request that is processed its value is changed so that it is not processed again printf("Sequence is : ");

printf("%d",cp1);

mov=mov+abs(cp1-a[0]); // head movement from initial position to first request

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

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

{

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

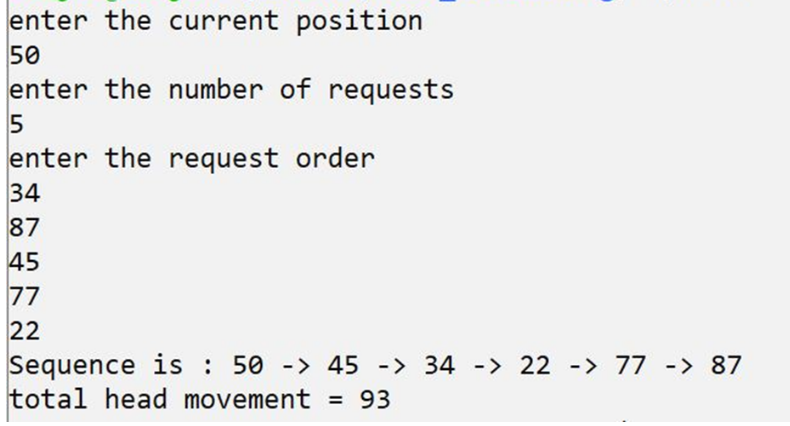
printf(" -> %d",a[i]);

} printf("\n");

printf("total head movement = %d\n",mov);

}

**OUTPUT**



**RESULT**

Thus a C program to simulate SSTF disk scheduling is executed successfully.