EX.NO: 1 **<u>BASICS OF UNIX COMMANDS</u>**
DATE:

<u>AIM</u>

To study various UNIX command in detail.

<u>COMMANDS</u>

<u>GENERAL COMMANDS</u>

NAME	SYNTAX	DESCRIPTION	EXAMPLE
Who	$who	Displays users who are currently logged on	[it40@syamantaka ~]$ who root pts/0 2011-03-10 20:57 (10.2.1.73) it40 pts/4 2011-03-10 23:17 (10.2.1.204)
Who am i	$who am i	Displays our own login terminal and other details	[it40@syamantaka ~]$ who am i it40 pts/4 2011-03-10 23:17 (10.2.1.204)
finger [user]	$ finger it3	Displays system biography on user `[user]'.	[it3@syamantaka ~]$ finger it3 Login: it3 Name: (null) Directory: /home/it3 Shell: /bin/bash On since Tue Mar 15 02:30 (IST) on pts/0 from 10.2.1.204 No mail. No Plan.
Date	$date	Displays the date which is stored in particular format	[it40@syamantaka ~]$ date Thu Mar 10 23:19:17 IST 2011
Calendar	$cal<month><y	of specified month and year	[it40@syamantaka ~]$ cal 1 1983 January 1983 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Clear	$clear	Clears the screen and displays the new screen	[it40@syamantaka ~]$ clear [it40@syamantaka ~]$

Uname	$uname	Displays the OS which is used.	[it40@syamantaka ~]$ uname Linux #1 SMP Fri Jan 26 14:42:21 EST 2007 i686 i6 86 i386 GNU/Linux
	$uname –a	Displays all machine details	[it40@syamantaka ~]$ uname -a Linux syamantaka.vec 2.6.18-8.el5xen
	$uname –s	Displays the OS name	[it40@syamantaka ~]$ uname -s Linux
	$uname –v	Displays the version of OS	[it40@syamantaka ~]$ uname -v #1 SMP Fri Jan 26 14:42:21 EST 2007
	$uname –r	Displays the release of OS	[it40@syamantaka ~]$ uname -r 2.6.18-8.el5xen
	$uname –n	Displays the name of network mode	[it40@syamantaka ~]$ uname -n syamantaka.vec
	$uname –m	Displays the type of OS	[it40@syamantaka ~]$ uname -m i686
Binary calculato r	$bc	Used for calculation	[it40@syamantaka ~]$ bc bc 1.06 Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc. This is free software with ABSOLUTELY NO WARRANTY. For details type `warranty'. 15 + 2 17
Identifier	$id	Displays userid and groupid	[it40@syamantaka ~]$ id uid=575(it40) gid=575(it40) groups=575(it40) context=user_u:system_r:unconfined_t
Present working Directory	$pwd	Displays the currently working directory	[it40@syamantaka ~]$ pwd /home/it40
Echo	$echo<text>	Displays the text given by the user	[it40@syamantaka ~]$ echo os os

Man	$man <command>	Displays the details of specified command	$man clear clear(1) clear(1) NAME clear - clear the terminal screen SYNOPSIS clear DESCRIPTION clear clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen. clear ignores any command-line parameters that may be present.
Touch	$touch <filenam		[it40@syamantaka ~]$ touch f1
Tty	$tty	Displays the terminal number of system	[it40@syamantaka ~]$ tty /dev/pts/4

LISTING OPTIONS

	DESCRIPTION	EXAMPLE
$ls	Lists all files in the present working directory	[it40@syamantaka ~]$ ls a.out fact.c ffs.c hi max.c one.sh rad1.c rr.c sjf. su two.c
$ls –a	Displays all hidden files of the user	[it40@syamantaka ~]$ ls -a . .bash_logout fact ffs.c lg.c one.c rad .ragr.c.swp six.c ss two.c a.out .bashrc fc.c four.c max.c .one.sh.swp rad.c .sf.c.swp sjf.c sum.c .viminfo
$ls –c	Lists all subdirectories of the file in columnwise fashion	[it40@syamantaka ~]$ ls -c f1 ss fcfs.c nandy.sh lg.c five.c three.c fc.c sjf. a.out sjf.c mat.c six.c rad two.c fact.c
$ls –d	Displays the root directory of the present directory	[it40@syamantaka ~]$ ls -d .
$ls –r	Reverses the order in which files and sundirectories are displayed	[it40@syamantaka ~]$ ls -r two.sh sum.c sjf.c shivani rad.c priority.c nandy.sh lg.c five.c fc.c f1

$ls –R	Lists the files and subdirectories in hierarchial order	[it40@syamantaka ~]$ ls -R .: a.out fact.c ffs.c hi max.c one.sh rad1.c rr.c sjf. su two.c f1 fc.c five.c lg.c nandy.sh priority.c rad.c shivani sjf.c sum.c two.sh
$ls –t	Displays the files in the order of modification	[it40@syamantaka ~]$ ls -t f1 ss fcfs.c nandy.sh lg.c five.c three.c fc.c
$ls –p	Displays files and sundirectories by a slashmark	[it40@syamantaka ~]$ ls -p a.out fact.c ffs.c hi/ max.c one.sh rad1.c rr.c sjf. su two.c
$ls –i	Displays the node number of each file	[it40@syamantaka ~]$ ls -i 26543903 a.out 26545681 ffs.c 26545241 max.c 26544524 rad1.c 26546141 sjf. 26544865 two.c
$ls –l	Lists the permission given to each file	[it40@syamantaka ~]$ ls -l total 248 -rwxrwxr-x 1 it40 it40 6356 Feb 24 21:34 a.out -rw-rw-r-- 1 it40 it40 0 Mar 10 23:31 f1 -rw-rw-r-- 1 it40 it40 193 Jan 6 21:58 fact

PATTERN SEARCHING COMMAND

	SYNTAX	DESCRIPTION	EXAMPLE
Grep	$grep<pattern> <filename>	Displays the line in which the given pattern is seen	[it3@syamantaka ~]$ grep include io.c #include<sys/types.h> #include<sys/stat.h> #include<fcntl.h> #include<stdlib.h> #include<stdio.h>

FILE MANIPULATION COMMANDS

	SYNTAX	DESCRIPTION	EXAMPLE
	$cat>><filename>	Edit contents of existing file	[it3@syamantaka ~]$ cat>>os os is an interface
Cat	$cat <filename>	View the contents of the file	[it3@syamantaka ~]$ cat os os is an interface

Cat	$cat –n filename	Displays the file with line numbers	[it3@syamantaka ~]$ cat -n os os is an interface
More	$more <filename>	Displays the file page by page	[it3@syamantaka ~]$ more os os is an interface
		e+10 <filename> Files will be from the 10th page onwards	[it3@syamantaka ~]$ more +10 fibo.sh a=$b b=$c i=`expr $i + 1` echo $c done
Wc	$wc	Counts the number of words,characters and lines for a given file	[it3@syamantaka ~]$ wc os 2 4 20 os
Wc –l	$wc –l	Displays the number of lines	[it3@syamantaka ~]$ wc -l os 2 os
	$wc –w	Displays the number of words	[it3@syamantaka ~]$ wc -w os 4 os
Wc –		Displays only the number of characters	[it3@syamantaka ~]$ wc -c os 20 os
Cp	$cp <filename1><filename e	Copy a file	[it3@syamantaka ~]$ cp os so [it3@syamantaka ~]$ cat so

	2>		os is an interface
Mv	$mv<filename1><filen ame2>	Move a file from a directory to another directory	[it3@syamantaka ~]$ mv os op [it3@syamantaka ~]$ cat op os is an interface
rm	rm <filename>	removes a file	[it3@syamantaka ~]$ rm f1
	rm –i filename	ask you for confirmation before actually deleting anything	[it3@syamantaka ~]$ rm -i f2 rm: remove regular file `f2'? y

Diff	diff <filename1> <filename2 >	compares files, and shows where they differ	[it3@syamantaka ~]$ cat>>check1 anand paul rajesh [it3@syamantaka ~]$ cat>>check2 anand rajesh [it3@syamantaka ~]$ diff check1 check2 2d1 < paul
Ln –s	$ln –s <source filename><new filename>	Creates a soft link between files. Here contents are copied to a new file but the memory addresses of both files are same.[After creating the link (i.e.,) naming a file with 2 different names,if the original file is deleted ,the newly created file is automatically deleted]	[it3@syamantaka ~]$ cat>>os os is resident monitor [it3@syamantaka ~]$ ln -s os os1 [it3@syamantaka ~]$ cat os os is resident monitor [it3@syamantaka ~]$ cat os1 os is resident monitor [it3@syamantaka ~]$ rm os [it3@syamantaka ~]$ cat os1 cat: os1: No such file or directory [it3@syamantaka ~]$
Ln	$ln <source filename><new filename>	Creates a hard link between files .Here contents is copied to a new file which is saved in a new address. .[After creating the link (i.e.,) naming a file with 2 different names, if the original file is deleted ,the newly created file is not deleted]	[it3@syamantaka ~]$ cat>>os os is an interface [it3@syamantaka ~]$ ln os os1 [it3@syamantaka ~]$ cat os os is an interface [it3@syamantaka ~]$ cat os1 os is an interface [it3@syamantaka ~]$ rm os [it3@syamantaka ~]$ cat os1 os is an interface

FILTER COMANDS

	SYNTAX	DES
Head	$head <filename	Dis the
Head -5	$head -5 <filename>	Displays the top file
Tail	$tail <filename>	Displays the last the file
	$tail -5 <filename	Di
Paste	$paste <filename1><filename2>	Paste two files in manner

Sort	$sort <filename>	Sorts the contents of the file in alphabetical order	[it3@syamantaka ~]$ sort alpha a b c d e f g h i j k l m n o p q r s t u v w x y z
	$sort –r<filename>	Sorts the contents of the file in reversed alphabetical order	[it3@syamantaka ~]$ sort -r alpha z y x w v u t s r q p o n m l k

			j i h g f e d c b a

DIRECTORY COMMANDS

NAME	SYNTAX	DESCRIPTION	EXAMPLE
Present working Directory	$pwd	Displays the currently working directory	[it40@syamantaka ~]$ pwd /home/it40
Make directory	$ mkdir subdir	**mkdir** creates a new subdirectory inside of the directory where you are currently working	[it3@syamantaka ~]$ mkdir book
Change directory	$ cd Misc	**cd** moves you to another directory.	it3@syamantaka ~]$ cd book [it3@syamantaka book]$
		To change back to your home directory: To change back to your home directory:	[it3@syamantaka book]$ cd .. [it3@syamantaka ~]$
Remove directory	$rmdir filename	To remove a directory. If the directory contains subdirectory or files, then remove it first and then remove the directory.	[it3@syamantaka ~]$ rmdir book

RESULT

Thus the various UNIX commands are executed successfully.

PROGRAMS USING SYSTEM CALLS

EX.NO: 2a FORK SYSTEM CALL
DATE:

AIM

To write a UNIX C program to create a child process from parent process using fork() system call.

ALGORITHM

Step 1: Start the program.
Step 2: Invoke a fork() system call to create a child process that is a duplicate of parent process. Step 3: Display a message.
Step 4: Stop the program.

PROGRAM 1

```
#include<stdio.h>
#include<unistd.h>
main()
 {
fork();
printf("Hello World\n");
 }
```

OUTPUT

```
[it1@localhost ~]$ vi ex1a.c
[it1@localhost ~]$ cc ex1a.c
[it1@localhost ~]$ ./a.out
Hello World
Hello World
```

PROGRAM 2

```
#include<stdio.h>
#include<unistd.h>
main()
 {
fork();
fork();
fork();
printf("Hello World\n");
 }
```

OUTPUT

```
[it1@localhost ~]$ cc ex1aa.c
[it1@localhost ~]$ ./a.out
Hello World
Hello World
Hello World
Hello World
```

Hello World
Hello World
Hello World
Hello World

RESULT

Thus a UNIX C program to create a child process from parent process using fork() system call is executed successfully.

EX.NO: 2b **SIMULATION OF FORK, GETPID AND WAIT SYSTEM CALLS**
DATE:

AIM

To write a UNIX C program simulate fork(),getpid() and wait() system call.

ALGORITHM

Step 1: Invoke a fork() system call to create a child process that is a duplicate of parent process. Step 2: Retrieve the process id of parent and child process.
Step 3: If return value of the fork system call=0(i.e.,child process),generate the fibonacii series.
Step 4: If return value of the fork system call>0(i.e.,parent process),wait until the child completes.

PROGRAM

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<conio.h>
main()
 {
 int a=-1,b=1,i,c=a+b,n,pid,cpid;
 printf("\nenter no. of terms ");
 scanf("%d",&n);
 pid=getpid();
```

```
printf("\nparent process id is %d",pid);
pid=fork();
cpid=getpid();
printf("\nchild process:%d",cpid);
if(pid==0)
 {
printf("\nchild is producing fibonacci series ");
for(i=0;i<n;i++)
{
c=a+b;
printf("\n%d",c);
a=b;
b=c;
}
printf("\nchild ends");
}
else
{
printf("\nparent is waiting for child to complete");
wait(NULL);
printf("\nparent ends");
}
 }
```

OUTPUT

```
[it27@mm4 ~]$ cc fork.c
[it27@mm4 ~]$ a.out

enter no. of terms 5

parent process id is 8723
child process:8723
parent process id is 8723
child process:8732
child is producing fibonacci series
0
1
1
2
3
child endsparent is waiting for child to complete
```

RESULT

Thus a UNIX C program to simulate fork(),getpid() and wait() system call is executed successfully.

EX.NO:2c **EXECUTION OF EXECLP SYSTEM CALL**

DATE :

AIM

To write a UNIX C program to implement execlp() system call.

ALGORITHM

Step 1:Use fork() system call to create a child process and assigns its id to pid variable. Step 2: If the pid<0,an error message is displayed.

Step 3:If the pid is equal to zero,execlp() system call is invoked and the child process is overwritten by the files in the directory.

Step 4:If the pid is greater than zero,display a message.

PROGRAM

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
main()
{
int pid;
pid=fork();
if(pid<0)
{
fprintf(stderr,"fork failed\n");
exit(-1);
}
else if(pid==0)
{
execlp("/bin/ls","ls",NULL);
}
else
{
```

```
wait(NULL);
printf("Child Complete");
exit(0);
 }
```

OUTPUT

```
 [it1@localhost ~]$ cc ex1c.c
 [it1@localhost ~]$ ./a.out
a1 ara.c arunv ex1aa.c facts itbgals kp matrix.v pre.c rad2.sh rad.c sk  wooow
a2 arain.c baabaaa ex1a.c fib.c k1 lee obu priya rad3.sh rad.sh ss xxx a.c arav.c cat ex1b.c fio
k2 lyffactz os rad1.c rad4.c sat sum.c
```

RESULT

Thus a UNIX C program to implement execlp() system call is executed successfully.

EX.NO: 2d SIMULATION OF SLEEP SYSTEM CALL/ZOOMBIE PROCESS

DATE :

AIM

To write a UNIX C program to simulate sleep system call.

ALGORITHM

Step 1: Invoke fork() system call and assign its return value to the variable p. Step 2:It the value is equal to zero,display the processid of child process. Step 3:Invoke sleep() system call,which allows the process to sleep for the specified seconds. Step 4:If the return value is greater than zero,display the process id of parent process.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
main()
 {
int pid;
pid=getpid();
printf("\n the current process id is %d",pid);
pid=fork();
if(pid==0)
 {
printf("\n child starts");
printf("\n child completed");
 }
else
 {
sleep(10);
printf("\n parent process running");
printf("\n i am in zoombie state");
 }
 }
```

OUTPUT

```
[it27@mm4 ~]$ cc zoom.c
[it27@mm4 ~]$ a.out
 the current process id is 9284
```

```
child starts
child completed
....(delay)
the current process id is 9284
parent process running
i am in zoombie state
```

RESULT

Thus a UNIX C program to simulate sleep system call was executed successfully.

EX.NO: 2e SIMULATION OF OPENDIR, READDIR SYSTEM CALLS

DATE :

AIM

To write a UNIX C program to simulate opendir and readdir system calls.

ALGORITHM

Step 1: Invoke opendir() system call to open a directory by passing command line argument as parameter.

Step 2: Invoke readdir() system call to read the opened directory.

Step 3:Until the end of directory is encountered, read all the files in directory and display its directory name,inode number and length of the record.

PROGRAM

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<dirent.h>
main(int argc,char *argv[])
{
DIR *dirname;
struct dirent *dir;
dirname=opendir(argv[1]);
dir=readdir(dirname);
while(dir!=NULL)
{
printf("Entry found :%s\n",dir->d_name);
printf("Inode number of entry:%d\n",dir->d_ino);
printf("Length of this record:%d\n",dir->d_reclen);
getchar();
}
}
```

OUTPUT

```
[it1@localhost ~]$ cc ex1f.c
[it1@localhost ~]$ mkdir anu
[it1@localhost ~]$ cd
[it1@localhost ~]$ cd anu
[it1@localhost anu]$ cat>>new
os is an interface
[it1@localhost anu]$ cat>>new1
os is resident monitor
[it1@localhost anu]$ cd
[it1@localhost ~]$ ./a.out anu
Entry found :..
Inode number of entry:26542353
```

Length of this record:16

RESULT

Thus a UNIX C program to simulate opendir and readdir system calls is executed successfully.

EX.NO: 2f **SIMULATION OF LS SYSTEM CALLS**

DATE :

AIM

To write a UNIX C program to simulate ls system call.

ALGORITHM

Step 1: Invoke opendir() system call to open a directory by passing command line argument as parameter.

Step 2: Invoke readdir() system call to read the opened directory.

Step 3:Until the end of directory is encountered, read all the files in directory and display its directory name.

PROGRAM

```
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
#include<string.h>
#include<dirent.h>
main(int argc,char *argv[])
 {
DIR *dp;
 struct dirent *link;
dp=opendir(argv[1]);
printf("contents of directory %s are \n",argv[1]);
while((link=readdir(dp))!=0)
printf("%s",link->d_name);
closedir(dp);
 }
```

OUTPUT

```
[it27@mm4 aaa]$ cat>>aa
os[it27@mm4 aaa]$ cat>>bb
system[it27@mm4 aaa]$ cd ..
[it27@mm4 ~]$ cc ls.c
[it27@mm4 ~]$ ./a.out aaa
contents of directory aaa are
.bb..aa
```

RESULT

Thus a UNIX C program to simulate ls system calls is executed successfully.

EX.NO: 2g **SIMULATION OF GREP SYSTEM CALLS** **DATE :**

AIM

To write a UNIX C program to simulate grep system call.

ALGORITHM

Step 1: Read the file and pattern to be searched.
Step 2:Open the file in read mode,search the pattern in the file using strstr function.
Step 3: If match of the pattern is found in the file ,display the entire line.

PROGRAM

```
#include<unistd.h>
#include<string.h>
main()
 {
char fn[10],pat[10],temp[1000];
FILE *fp;
printf(" enter file name:");
 scanf("%s",fn);
printf("\n enter the pattern::");
 scanf("%s",pat);
fp=fopen(fn,"r");
while(!feof(fp))
 {
fgets(temp,1000,fp);
if(strstr(temp,pat))
 {
printf("%s",temp);
 }
 }
fclose(fp);
 }
```

OUTPUT

```
[it27@mm4 ~]$ a.out
enter file name:fork.c
enter the pattern::include
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
```

RESULT

Thus a UNIX C program to simulate grep system calls is executed successfully.

EX.NO: 2h SIMULATION OF I/O SYSTEM CALLS
DATE:

AIM

To write a UNIX C program to simulate I/O system calls such as open, read and write.

ALGORITHM

Step 1: Start the program.

Step 2: Enter the filename to be opened.
Step 3: Using open () system call, open the file in read only mode.
Step 4: Using read () system call, read the content through the file descriptors and put it in a buffer.
Step 5: Display the content of the buffer.
Step 6: Stop the program.

PROGRAM

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
#include<stdio.h>
int main()
 {
int fd;
char buf1[100],fname[30];
printf("Enter the filename:");
 scanf("%s",fname);
fd=open(fname,O_RDONLY);
read(fd,buf1,30);
printf("\n The content is %s:",buf1);
close(fd);
 }
```

OUTPUT

```
[it3@localhost ~]$ cc io.c
[it3@localhost ~]$ ./a.out
Enter the filename:f1
 The content is : hi
Hello
```

RESULT

Thus a UNIX C program to simulate I/O system calls such as open read and write is executed successfully.

SHELL PROGRAMS

EX.NO: 3a PALINDROME OR NOT
DATE:

AIM

To write a shell program to check whether the string is palindrome or not.

ALGORITHM

Step 1: Read the string.
Step 2: Calculate the string length.
Step 3: Compare the first string with the last string, second string with previous character from the last until half the length is searched.

Step 4: If match is found, the given string is palindrome.
Step 5: Otherwise, the string is not a palindrome.

PROGRAM

```
echo enter the string
read str
len=`echo $str|wc -c`
len=`expr $len - 1`
i=1
j=`expr $len / 2`
while [ $i -le $j ]
do
k=`echo $str|cut -c $i`
l=`echo $str|cut -c $len`
if [ $k != $l ]
then
echo string is not palindrome
exit
fi
i=`expr $i + 1`
len=`expr $len - 1`
done
echo string is palindrome
```

OUTPUT

```
[it26@mm4 ~]$ sh pali.sh
enter the string
madam
string is palindrome
[it26@mm4 ~]$ sh pali.sh
enter the string
madem
string is not palindrome
[it26@mm4 ~]$
[it26@mm4 ~]$
```

RESULT

Thus a shell program to check if the string is palindrome or not is written and executed successfully.

EX.NO: 3b **SORTING**

DATE:

AIM

To write a shell program to arrange the numbers in ascending order.

ALGORITHM

Step 1: Read the number of elements to be sorted.
Step 2: Read the array of elements.
Step 3: The first element is compared with every other elements,if first number is greater that the others then the numbers are rearranged.
Step 4: Then ,second element is compared with third ,fourth upto n numbers,if it is greater rearrangement is done.
Step 5:This process is continued for the remaining numbers.

PROGRAM

```
echo " enter the total number of values "
read n
echo " enter the value "
```

```
for ((i=0;i<n;i++))
do
read a[$i]
done
for ((i=0;i<n;i++))
do
for ((j=i+1;j<n;j++))
do
if [ ${a[$i]} -gt ${a[$j]} ]
then
x=${a[$i]}
a[$i]=${a[$j]}
a[$j]=$x
fi
done
done
echo " ascending order is "
for ((i=0;i<n;i++))
do
echo " ${a[$i]} "
done
```

<u>OUTPUT</u>

```
enter the total number of values
4
 enter the value
5
2
8
1
 ascending order is
 1
 2
 5
 8
```

RESULT

Thus a shell program to arrange the numbers in ascending order is written& executed successfully.

EX.NO: 3c COUNT NUMBER OF CHARACTERS & WORDS
DATE:

AIM

To write a shell program to count number of characters and words in a file.

ALGORITHM

Step 1: Read the file.
Step 2: Count the number of characters and words in the file using wc command.
Step 3: Display the number of characters and words.

PROGRAM

```
echo enter the filename
read text
c=`echo $text | wc -c`
c=`expr $c - 1`
w=`echo $text | wc -w`
echo no of characters: $c
```

echo no of words: $w

OUTPUT

[it21@mm4 ~]$ cat>>aaa
os is an interface
[it21@mm4 ~]$ sh count.sh
enter the filename
aaa
no of characters: 3
no of words: 1

RESULT

Thus a shell program to arrange the numbers in ascending order was written and executed successfully.

EX.NO: 3d SEARCH OPERATION USING MENU DATE:

AIM

To write a shell program to implement linear and binary search using menu options.

ALGORITHM

Step 1: Create a menu choice of Binary Search and Linear search using Switch case Step 2: Implement Binary Search in Case 1

a. Read the number of elements in the array
b. Read array elements in ascending order
c. Read the element to be searched
d. Find the mid element then continue searching elements in the left half of an array. If it is not found search it in the right half of the array.
e. Continue the process until the element found

Step 3: Implement linear search in Case 2

a. Read the number of elements in the array
b. Read array elements in ascending order
c. Read the element to be searched
d. Search the element linearly in an array until it found

PROGRAM

echo "Search Using Menu"

```
z=1
while [ $z -eq 1 ]
do
echo "Select Choice from Menu"
echo "1)Binary Search 2)Linear Search"
read opt
case $opt in
 1) echo "***BINARY SEARCH***"
echo "enter the limit "
read n
echo "enter the array values"
for((i=0;i<n;i++))
do
read a[$i]
done
l=0
h=`expr $n - 1`
f=0
echo "enter the value to be searched"
read x
while [ $f -eq 0 -a $l -lt $h ]
do
m=`expr \( $l + $h \) / 2`
if [ ${a[$m]} -eq $x ]
then f=1
elif [ ${a[$m]} -lt $x ]
then l=`expr $m + 1`
elif [ ${a[$m ]} -gt $x ]
then h=`expr $m - 1`
fi
done
if [ $f -eq 1 ]
then
echo " element found in $m "
else
echo " not found "
fi
;;
 2) echo "***LINEAR
SEARCH***" echo "enter the limit"
read n
echo "Enter the elememts"
for (( i=0;i< $n;i++ ))
```

```
do
read a[$i]
done
echo "enter the element to be searched"
read x
found=0
for (( i=0;i< $n;i++ ))
do
if [ ${a[$i]} -eq $x ]
then
echo "found in $i"
found=1
break
fi
done
if [ $found -eq 0 ]
then
echo "not found"
fi
 ;;
* ) echo "INVALID OPTION"
;;
esac
z=0
echo "press 1 to cont...."
read $z
done
```

OUTPUT

```
Select Choice from Menu
 1)Binary Search 2)Linear Search
 1
***BINARY SEARCH***
enter the limit
5
enter the array values
 1
 2
 3
 4
 5
enter the value to be
 searched 3
```

element found in 2
press 1 to cont....
1
Select Choice from Menu
1)Binary Search 2)Linear Search
2
***LINEAR
SEARCH*** enter the
limit
5
Enter the elememts
10
20
30
40
15
enter the element to be searched
15
found in 4
press 1 to cont....

RESULT

Thus a shell program to perform search operations using menu was executed successfully.

EX.NO: 4a

DATE:

IMPLEMENTATION OF FIRST COME FIRST SERVED SCHEDULING ALGORITHM

AIM

To write a UNIX C program to implement first come first served scheduling algorithm.

ALGORITHM

Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on order in which it requests CPU.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time.
Step 6: Print the details about all the processes.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
struct process
{
int btime;
int wtime;
int ttime;
}p[50];
main()
{
int n,j,i;
float tot_num=0.0,tot_turn=0.0,tot_wait=0.0,avg_turn=0.0,avg_wait=0.0;
printf("\nEnter the number of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("\nEnter the burst time of each process:");
scanf("%d",&p[i].btime);}
i=1;
p[i].wtime=0;
p[i].ttime=p[i].btime;
tot_wait=p[i].wtime;
tot_turn=p[i].ttime;
for(i=2;i<=n;i++)
{
p[i].wtime=p[i-1].wtime+p[i-1].btime;
p[i].ttime=p[i].wtime+p[i].btime;
tot_wait=tot_wait+p[i].wtime;
tot_turn=tot_turn+p[i].ttime;
}
avg_wait=tot_wait/n;
avg_turn=tot_turn/n;
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n----------------------------------------------------------\n");
for(i=1;i<=n;i++)
printf("|\tp%d\t",i);
printf("|\t\n");
printf("\n----------------------------------------------------------\n");
printf("\n");
for(i=1;i<=n;i++)
printf("%d\t\t",p[i].wtime);
printf("%d",p[n].wtime+p[n].btime);
printf("\n----------------------------------------------------------\n");
printf("\n");
printf("\nProcess burst time waiting time turnaround time\n");
for(i=1;i<=n;i++)
{
```

```
printf("%5d%10d%15d%15d",i,p[i].btime,p[i].wtime,p[i].ttime);
printf("\n");
 }
printf("\nAverage wait time:%f\n",avg_wait);
printf("\nAverage turnaround time:%f\n",avg_turn);
 }
```

OUTPUT

```
[it17@localhost ~]$ vi fcfs.c
[it17@localhost ~]$ cc fcfs.c
[it17@localhost ~]$ ./a.out
[it1@linuserver ~]$ ./a.out

Enter the number of process:4
Enter the burst time of each process:5
Enter the burst time of each process:3
Enter the burst time of each process:8
Enter the burst time of each process:2



 GANTT CHART
----------------------------------------------------------
| p1 | p2 | p3 | p4 |
----------------------------------------------------------
0 5 8 16 18
----------------------------------------------------------

Process burst time waiting time turnaround time
 1 5 0 5
 2 3 5 8
 3 8 8 16
 4 2 16 18

Average wait time:7.250000

Average turnaround time:11.750000
```

RESULT

Thus a UNIX C program to implement first come first served scheduling algorithm is executed successfully.

EX.NO: 4b
DATE:

IMPLEMENTATION OF SHORTEST JOB FIRST SCHEDULING ALGORITHM

AIM

To write a UNIX C program to implement shortest job first scheduling algorithm.

ALGORITHM

Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on burst time.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time. Step 6: Print the details about all the processes.
Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
```

```
main()
 {
int p;
int n;
int i,j,tot_wt=0,tot_tt=0;
float avg_wt=0.0,avg_tu=0.0;
int stt;
int t,pno[10],bs[10],wt[10],tu[10],ft[10];
printf("enter the number of process:");
 scanf("%d",&p);
n=p;
for(i=1;i<=n;i++)
 {
pno[i]=i;
printf("enter the burst time :");
 scanf("%d",&bs[i]);
 }
for(i=1;i<=n;i++)
 {
for(j=i+1;j<=n;j++)
 {
if(bs[i]>bs[j])
 {
t=pno[i];
pno[i]=pno[j];
pno[j]=t;
t=bs[i];
bs[i]=bs[j];
bs[j]=t;
 }
 }
 }
 stt=0;
for(i=1;i<=n;i++)
 {
ft[i]=stt+bs[i];
wt[i]=stt;
tu[i]=wt[i]+bs[i];
tot_wt=tot_wt+wt[i];
tot_tt=tot_tt+tu[i];
 stt=ft[i];
 }
 avg_wt=tot_wt/n;
 avg_tu=tot_tt/n;
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n----------------------------------------------------------\n");
for(i=1;i<=n;i++)
printf("|\tP%d\t",pno[i]);
printf("|\t\n");
printf("\n----------------------------------------------------------\n");
```

```
printf("\n");
for(i=1;i<=n;i++)
printf("%d\t\t",wt[i]);
printf("%d",wt[n]+bs[n]);
printf("\n------------------------------------------------------------\n");
printf("\n");
printf("pno bursttime(msec) waitingtime(msec) turnaroundtime(msec)\n");
for(i=1;i<=n;i++)
 {
printf("%d\t\t%d\t\t%d\t\t%d",pno[i],bs[i],wt[i],tu[i]);
printf("\n");
 }
printf("Average waiting time in millisec is");
printf("\t%f",avg_wt);
printf("\nAverage turnaround time in millisec ");
printf("\t%f",avg_tu);
 }
```

OUTPUT

```
[it1@linuserver ~]$ ./a.out
enter the number of process:4
enter the burst time :2
enter the burst time :3
enter the burst time :4
enter the burst time :4

 GANTT CHART

------------------------------------------------------------
| P1 | P2 | P3 | P4 |

------------------------------------------------------------

0 2 5 9 13
------------------------------------------------------------

pno bursttime(msec) waitingtime(msec) turnaroundtime(msec)
 1 2 0 2
2 3 2 5
3 4 5 9
4 4 9 13
Average waiting time in millisec is 4.000000
Average turnaround time in millisec 7.000000
```

RESULT

Thus a UNIX C program to implement shortest job first scheduling algorithm is executed successfully..

EX.NO: 4c

DATE:

IMPLEMENTATION OF PRIORITY SCHEDULING ALGORITHM

AIM

To write a UNIX C program to implement priority scheduling algorithm.

ALGORITHM

Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on priority.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time. Step 6: Print the details about all the processes.
Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
main()
 {
int t[20],p[20],pro[20],temp,wait[20],turn[20],tot_wait=0,totturn=0,i,j,n;
float avgwait=0,avgturn=0;
printf("Enter the no. of processes:");
scanf("%d",&n);
for(i=1;i<=n;i++)
 {
printf("\nEnter burst time for process %d:",i);
scanf("%d",&t[i]);
printf("\nEnter priority for process %d:",i);
scanf("%d",&p[i]);
```

```
pro[i]=i;
 }
for(i=1;i<=n;i++)
 {
for(j=1;j<=n;j++)
 {
if(p[j]>p[i])
 {
temp=p[j];
p[j]=p[i];
p[i]=temp;
temp=t[j];
t[j]=t[i];
t[i]=temp;
temp=pro[j];
pro[j]=pro[i];
pro[i]=temp;
 }
 }
 }
for(i=1;i<=n;i++)
 {
wait[i]=0;
for(j=1;j<i;j++)
 {
wait[i]+=t[j];
 }
tot_wait+=wait[i];
turn[i]=wait[i]+t[j];
totturn+=turn[i];
 }
printf("\n\nProcess\tBurst time\tPriority\tWaiting time\tTurnaround time");
for(i=1;i<=n;i++)
 {
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",pro[i],t[i],p[i],wait[i],turn[i]);
 }
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n------------------------------------------------------------\n");
for(i=1;i<=n;i++)
printf("|\tP%d\t",pro[i]);
printf("|\t\n");
printf("\n------------------------------------------------------------\n");
printf("\n");
for(i=1;i<=n;i++)
printf("%d\t\t",wait[i]);
printf("%d",wait[n]+t[n]);
printf("\n------------------------------------------------------------\n");
printf("\n");
printf("\n\nTotal waiting time=%d\nAverage waiting time=%f\n\n",tot_wait,(float)tot_wait/n);
printf("\n\nTotal turnaround time=%d\n Average turnaround
```

time=%f\n\n",totturn,(float)totturn/n);
}

OUTPUT

[it17@localhost ~]$ cc priority.c
[it17@localhost ~]$ ./a.out
Enter the no. of processes:5
Enter burst time for process 1:3
Enter priority for process 1:1
Enter burst time for process 2:5
Enter priority for process 2:2
Enter burst time for process 3:8
Enter priority for process 3:1
Enter burst time for process 4:5
Enter priority for process 4:4
Enter burst time for process 5:8
Enter priority for process 5:3

Process Burst time Priority Waiting time Turnaround time
1 3 1 0 3
3 8 1 3 11
2 5 2 11 16
5 8 3 16 24
4 5 4 24 29

GANTT CHART

- -
| P1 | P3 | P2 | P5 | P4|

- -
0 3 11 16 24 29

- -
Total waiting time=54
Average waiting time=10.800000
Total turnaround time=83
Average turnaround time=16.600000

RESULT

Thus a UNIX C program to implement priority scheduling algorithm is executed successfully.

EX.NO: 4d

DATE:

IMPLEMENTATION OF ROUND ROBIN SCHEDULING ALGORITHM AIM

To write a UNIX C program to implement round robin scheduling algorithm.

ALGORITHM

Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on priority.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time. Step 6: Print the details about all the processes.
Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
main()
{
int i,n,j=0;
int b[20],b1[20],f[20],w[20];
int start,finish,t,total=0;
float aw=0.0,at=0.0;
printf("\nEnter no. of process:");
scanf("%d",&n);
printf("\nEnter the time slice:");
scanf("%d",&t);
for(i=1;i<=n;i++)
{
printf("\nEnter the burst time of process %d:",i);
scanf("%d",&b[i]);
b1[i]=b[i];
total=total+b[i];
}
printf("\n\t\t RoundRobbin Scheduing");
start=0;
printf("\nThe process Scheduling is as follows");
printf("\nTime slice:%d ms",t);
while(j<total)
{
for(i=1;i<=n;i++)
{
if(b[i]==0)
continue;
if(b[i]>t)
{
printf("\nProcess %d:",i);
```

```
finish=start+t;
j=j+t;
start=finish;
b[i]=b[i]-t;
printf("\nRemaining burst time:%d ms",b[i]);
}
else
{
printf("\nProcess %d:",i);
finish=start+b[i];
j=j+b[i];
start=finish;
f[i]=finish;
b[i]=0;
w[i]=finish-b1[i];
printf("\tRemaining burst time:%dms",b[i]);
}
}
}
printf("\n\nProcess no\tBurst time\tWaiting time\tTurnaround time");
for(i=1;i<=n;i++)
{
printf("\n\n%d%20d%20d%20d",i,b1[i],w[i],f[i]);
aw=aw+w[i];
at=at+f[i];
}
aw=aw/(float)n;
at=at/(float)n;
printf("\nAvg wt time:%f",aw);
printf("\nAvg tat:%f\n",at);
}
```

OUTPUT

```
[it17@localhost ~]$ ./a.out
Enter no. of process:5
Enter the time slice:2
Enter the burst time of process 1:5
Enter the burst time of process 2:2
Enter the burst time of process 3:1
Enter the burst time of process 4:3
Enter the burst time of process 5:2
 RoundRobin Scheduing
The process Scheduling is as follows
```

Time slice:2 ms
Process 1:
Remaining burst time:3 ms
Process 2: Remaining burst time:0ms
Process 3: Remaining burst time:0ms
Process 4:
Remaining burst time:1 ms
Process 5: Remaining burst time:0ms
Process 1:
Remaining burst time:1 ms
Process 4: Remaining burst time:0ms
Process 1: Remaining burst time:0ms

Process no Burst time Waiting time Turnaround time
1 5 8 13
2 2 2 4
3 1 4 5
4 3 9 12
5 2 7 9

Avg wt time:6.000000
Avg tat:8.600000

RESULT

Thus a UNIX C program to implement round robin scheduling algorithm is executed successfully.

EX.NO: 5a **SEQUENTIAL FILE ALLOCATION** **DATE:**

AIM

To write a UNIX C program to simulate contiguous file allocation.

ALGORITHM

Step 1: Start the program.
Step 2: Read the number of files.
Step 3: Input the filename.
Step 4: Open the file in read only mode and find start, end and length of the file.
Step 5: Display the name, start, end and length of the file.

PROGRAM

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
```

```
void main()
 {
char buff[10][10],name[10][10];
int fp,n[20],start=0,num,i,temp;
printf("\n Enter the number of files:");
 scanf("%d",&num);
for(i=0;i<num;i++)
 {
printf("\n Enter the name of the file %d:",i+1);
 scanf("%s",&name[i]);
fp=open(name[i],O_RDONLY);
read(fp,buff[i],100);
n[i]=strlen(buff[i]);
printf("%d",n[i]);
 }
printf("\n Filename\t Start \t End \t Length\n");
for(i=0;i<num;i++)
 {
if(i==0)
temp=0;
else
temp=start+1;
 start=start+n[i]-1;
if(i!=0)
 start++;
printf("\n %s\t\t %d\t %d\t%d\n",name[i],temp,start,n[i]);
 }
 }
```

OUTPUT

```
[it3@syamantaka ~]$ ./a.out
 Enter the number of files:2
 Enter the name of the file 1:f1
12
 Enter the name of the file 2:f2
8

Filename Start End Length

 f1 0 11 12

 f2 12 19 8
```

RESULT

Thus a UNIX C program to simulate contiguous file allocation scheme is executed successfully.

EX.NO: 5b <u>LINKED FILE ALLOCATION</u> DATE :

AIM

To write a UNIX C program to simulate linked file allocation.

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
Step 6: Stop the allocation

PROGRAM

```
#include<stdio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
 }f[10];
void main()
{
```

```
int i,j,n;
printf("Enter no. of files:");
 scanf("%d",&n);
for(i=0;i<n;i++)
 {
printf("Enter file name:");
 scanf("%s",&f[i].fname);
printf("Enter starting block:");
 scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("Enter no.of blocks:");
 scanf("%d",&f[i].size);
printf("Enter block numbers:");
for(j=1;j<=f[i].size;j++)
 {
scanf("%d",&f[i].block[j]);
 }
 }
printf("File\tstart\tsize\tblock\n");
for(i=0;i<n;i++)
 {
printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
for(j=1;j<=f[i].size-1;j++)
printf("%d--->",f[i].block[j]);
printf("%d",f[i].block[j]);
printf("\n");
 }
}
```

OUTPUT

```
Enter file name : index
Enter starting block:20
Enter no.of blocks:5
Enter block numbers:
4
 12
 15
45
32
Enter file name : optimal
Enter starting block:12
Enter no.of blocks:4
Enter block numbers:
6
5
4
3
```

File start size block
index 20 5 4--->12--->15--->45--->32
optimal 12 4 6--->5--->4--->3

RESULT

Thus a UNIX C program to simulate linked file allocation scheme is executed successfully.

EX.NO: 5c INDEXED FILE ALLOCATION DATE :

AIM

To write a UNIX C program to simulate indexed file allocation.

ALGORITHM

Step 1: Start the program.
Step 2: Read the number of files.
Step 3: Input the filename.
Step 4: Open the file in read only mode and find start, end and length of the file.
Step 5: Display the name, start, end and length of the file.

PROGRAM

```
#include<stdio.h>
void main()
{
int f[50],i,k,j,inde[50],n,c,count=0,p;
for(i=0;i<50;i++)
f[i]=0;
x:
printf("Enter the index block\t");
scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;
printf("Enter the number of files on index\t");
scanf("%d",&n);
}
else
{
printf("Block already allocated\n");
```

```
goto x;
 }
for(i=0;i<n;i++)
 scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
 {
printf("Block already allocated");
goto x;
 }
for(j=0;j<n;j++)
f[inde[j]]=1;
printf("\n Allocated");
printf("\n File indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf("\n Enter 1 to enter more files and 0 to exit\t");
 scanf("%d",&c);
if(c==1)
goto x;
else
exit(0);
 }
```

OUTPUT

```
Enter the index block 9
Enter the number of files on index 3
 1 2 3
 Allocated
 File indexed
 9->1:1
 9->2:1
 9->3:1
Enter 1 to enter more files and 0 to exit 0
```

RESULT

Thus a UNIX C program to simulate indexed file allocation scheme is executed successfully.

EX.NO: 6 PRODUCER CONSUMER PROBLEM USING SEMAPHORES DATE:

AIM

To write a UNIX C program to implement producer consumer problem using semaphores.

ALGORITHM

Step 1: Start the program.
Step 2: Initialize mutex to 1, full to 0 and empty to n.
Step 3: If mutex=1 and empty is greater than zero, invoke producer function. Otherwise display the message buffer is full.
Step 4: If mutex=1 and full is greater than zero, invoke consumer function. Otherwise display the message buffer is empty.
Step 5: In producer function, produce an item wait for empty buffer and mutex. If any buffer is empty add the produced item to the buffer and signal mutex and full buffer. Step 6: In consumer function, wait for full buffer and mutex. If any buffer is full, remove the item from the buffer to the consumer. Signal mutex and empty buffer.
Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
#define n 3
int bu[n],a,in=1,out=1,mutex=1,full=0,empty=n,x=0,nc;
main()
{
int ch;
void producer();
void consumer();
```

```
int wait(int);
int signal(int);
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1)
{
printf("\nENTER YOUR CHOICE\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
if((mutex==1)&&(empty>0))
producer();
else
printf("BUFFER IS FULL");
break;
case 2:
if((mutex==1)&&(full>0))
consumer();
else
printf("BUFFER IS EMPTY");
break;
case 3:
exit(0);
break;
}
}
}
int wait(int s)
{
return(--s);
}
int signal(int s)
{
return(++s);
}
producer()
{
printf("\nEnter the item produced by
producer:"); scanf("%d",&a);
empty=wait(empty);
mutex=wait(mutex);
bu[in]=a;
printf("\nproducer produces the item%d",bu[in]);
in++;
mutex=signal(mutex);
full=signal(full);
}
void consumer()
{
full=wait(full);
```

```
mutex=wait(mutex);
nc=bu[out];
printf("\n consumer consumes
item%d",nc); out++;
mutex=signal(mutex);
empty=signal(empty);
}
```

OUTPUT

```
 [it3@localhost ~]$ ./a.out
 1.PRODUCER
2.CONSUMER
3.EXIT

ENTER YOUR CHOICE
 1

Enter the item produced by producer:10
producer produces the item10

ENTER YOUR CHOICE
 1
Enter the item produced by producer:20
producer produces the item20

ENTER YOUR CHOICE
2
 consumer consumes item10

ENTER YOUR CHOICE
2
 consumer consumes item20

ENTER YOUR CHOICE
2
BUFFER IS EMPTY

ENTER YOUR CHOICE
3
```

RESULT

Thus a UNIX C program to implement producer consumer problem using semaphores is executed successfully.

EX.NO: 7a **SINGLE LEVEL DIRECTORY FILE ORGANIZATION**

DATE:

AIM

To write a UNIX C Program to implement single level directory file organization

ALGORITHM

Step 1: Start the process.
Step 2: Get the number of directories.
Step 3: Get the name of the directory.
Step 4: Display the size of the directory.
Step 5: Display the number and name of the files which are presented in the directory.
Step 6: Stop the pocess.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
 {
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
clrscr();
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]);
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");
getch();
```

}

OUTPUT

Enter number of directories:2
Enter name of directories:abc
Xyz
Enter size of directories:2
3
Enter the file name:A
B
C
D
Directory size filenames
abc 2 a
b
xyz 3 c
d
e

RESULT

Thus a UNIX C Program to implement single level directory file organization was executed successfully.

EX.NO: 7b TWO LEVEL DIRECTORY FILE ORGANIZATION DATE:

AIM

To write a UNIX C Program to implement two level directory file organization

ALGORITHM

Step 1: Start the process.
Step 2: Get the number of directories.
Step 3: Get the name of the main directory.

Step 4: Display the size of the directory.
Step 5: Get the name of the sub-directory which is in the main directory.
Step 6: Display the size of the sub-directory.
Step 5: Display the number and name of the files which are presented in the directory.
Step 6: Stop the pocess.

PROGRAM

```
#include<stdio.h>
#include<conio.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;
clrscr();
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(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(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");
}
getch();
}
```

OUTPUT

```
enter number of directories:2
enter directory 1 names:abc
enter size of directories:1
enter subdirectory name and size:c
3
enter file name:cde
enter file name:efg
enter file name:ghi
enter directory 2 names:abc1
enter size of directories:2
enter subdirectory name and size:aa1
2
enter file name:cde1
enter file name:efg1
enter subdirectory name and size:aa2
1
enter file name:cde2

dirname size subdirname size files abc 1 c 3 cde efg ghi

abc1 2 aa1 2 cde1 efg1 aa2 1 cde2
```

RESULT

Thus a UNIX C Program to implement two level directory file organization was executed successfully.

EX.NO: 8 BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE

DATE:

AIM

To write a UNIX C Program to implement deadlock avoidance using Banker's Algorithm .

ALGORITHM

Step 1: Start the program.
Step 2: Get the number of process, resource, allocation matrix, Maximum matrix and Available matix.
Step 3: Calcualte need matrix using Maximum matrix and allocation matrix i.e., NEED[i,j]=MAX[i,j]-ALLOC[i,j]. Also initialize work matrix i.e.,WORK[j]=AVAIL[j]. Step 4:

Now find safety sequence of the system using work matrix,need matrix. Step 5: Now compare NEED and WORK matrix, if NEED[i,j]<=WORK[j] then execute the corresponding process also calculate new work matrix i.e.,WORK[j]=WORK[j]+ALLOC[i,j]. Step 6: If condition doesn't satisfies, move to next process and repeat Step 5,until completion of all processes and print the safety sequence.

Step 7: If any process request for resource, get the process id and request matrix from the user.

Step 8: Check the following conditions for the respective process using Request matrix,Need matrix and Available matrix i.e., REQ[j]<=NEED[k,j] and REQ[j]<=AVAIL[j]. Step 9: If the above condition satisfies, request can be granted and proceed next step otherwise display error as "Request cannot be granted".

Step10: Now calculate new Available matrix,Allocation matrix,Need matrix and Work matrix i.e., AVAIL[j]=AVAIL[j]-REQ[j],
WORK[j]=AVAIL[j],ALLOC[k,j]=ALLOC[k,j]+REQ[j],
NEED[k,j]=NEED[k,j]-REQ[j].

Step 11: To find safety sequence of the system, repeat Step 5 and Step 6 for all processes.

Step 12: Then print the safety sequence of execution of processes.

Step 13: Stop the program .

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int alloc[20][20],max[20][20],avail[20],need[20][20],work[20]={0};
int newavail[20],req[20]={0},check=0,check2=0,cond=0,p;
int i=0,j=0,m=0,n=0,t=0,x=0,c[20]={0},k=0,count,count2,a[20],b;
int x2=0,c2[20];
printf("Enter the no. of resources\n");
scanf("%d",&m);
printf("Enter the no. of process");
scanf("%d",&n);
printf("\n Enter the resources for available\n");
for(j=1;j<=m;j++)
{
printf("Enter the %d resources of avail",j);
scanf("%d", &avail[j]);
work[j]=avail[j];
}
for(i=1;i<=n;i++)
{
for(j=1;j<=m;j++)
{
printf("Enter the %d resources of %d
alloc",j,i); scanf("%d", &alloc[i][j]);
}
for(j=1;j<=m;j++)
{
printf("\n Enter the %d resource of %d max",j,i);
scanf("%d",&max[i][j]);
```

```
need[i][j]=max[i][j]-alloc[i][j];
 }
 }
printf("\n Allocation max need\n");
for(i=1;i<=n;i++)
 {
for(j=1;j<=m;j++)
printf("%2d",alloc[i][j]);
printf("\t");
for(j=1;j<=m;j++)
printf("%2d",max[i][j]);
printf("\t");
for(j=1;j<=m;j++)
printf("%2d",need[i][j]);
printf("\n");
 }
printf("\n Process executes in this order\n");
do{
for(i=1;i<=n;i++)
 {
count=0;
if(c[i]!=i+1)
 {
for(j=1;j<=m;j++)
 {
if(need[i][j]<=work[j])
count = count+1;
 }
if(count == m)
 {
printf("p%d\t",i);
c[i]=i+1;
x = x+1;
for(j=1;j<=n;j++)
work[j]=work[j]+alloc[i][j];
 }
 }
 }
check = check +1;
 }
while(x<n && check <=n);
if(x==n)
printf("\n system is in saftey\n");
else
printf("\n System is not in saftey");
printf("\n Checking the bankers algorithm after the request");
printf("\n Enter the request process number");
 scanf("%d",&p);
printf("\n Enter the values");
for(j=1;j<=m;j++)
```

```
{
scanf("%d",&req[j]);
}
for(j=1;j<=m;j++)
{
if(req[j]<=avail[j]&&req[j]<=need[p][j])
cond=cond+1;
}
if(cond==m)
{
for(j=1;j<=m;j++)
{
alloc[p][j]=alloc[p][j]+req[j];
avail[j]=avail[j]-req[j];
need[p][j]=need[p][j]-req[j];
}
}
else
{
printf("req is not satisfied");
exit(0);
}
printf("\n Execution of process after request");
do
{
for(i=1;i<=n;i++)
{
count2=0;
if(c2[i]!=i+1)
{
for(j=1;j<=m;j++)
{
if(need[i][j]<=avail[j])
count2 = count2+1;
}
if(count2 == m)
{
rintf("p%d\t",i);
c2[i]=i+1;
x2=x2+1;
for(j=1;j<=n;j++)
{
avail[j]=avail[j]+alloc[i][j];
}
}
}
}
check2 = check2 +1;
}while((x2<n)&&(check2<=n));
if(x2==n)
```

printf("\n System is in safe state we can grant the request");
else
printf("\n System is in unsafe state we cannot grant the request"); }

OUTPUT

```
[it16@syamantaka ~]$ cc banker.c
[it16@syamantaka ~]$ ./a.out
Enter the no. of resources:3
Enter the no. of process:5
Enter the resources for available
Enter the 1 resources of avail:3
Enter the 2 resources of avail:3
Enter the 3 resources of avail:2
Enter the 1 resources of 1 alloc:0
Enter the 2 resources of 1 alloc:1
Enter the 3 resources of 1 alloc:0
Enter the 1 resource of 1 max: 7
Enter the 2 resource of 1 max: 5
Enter the 3 resource of 1 max:3
Enter the 1 resources of 2 alloc:2
Enter the 2 resources of 2 alloc:0
Enter the 3 resources of 2 alloc:0
Enter the 1 resource of 2 max:3
Enter the 2 resource of 2 max:2
Enter the 3 resource of 2 max:2
Enter the 1 resources of 3 alloc:3
Enter the 2 resources of 3 alloc:0
Enter the 3 resources of 3 alloc:2
Enter the 1 resource of 3 max: 9
Enter the 2 resource of 3 max: 0
Enter the 3 resource of 3 max: 2
Enter the 1 resources of 4 alloc: 2
Enter the 2 resources of 4 alloc: 1
Enter the 3 resources of 4 alloc: 1
Enter the 1 resource of 4 max:2
Enter the 2 resource of 4 max:2
Enter the 3 resource of 4 max:2
Enter the 1 resources of 5 alloc:0
Enter the 2 resources of 5 alloc:0
Enter the 3 resources of 5 alloc:2
Enter the 1 resource of 5 max:4
Enter the 2 resource of 5 max:3
Enter the 3 resource of 5 max:3
Allocation max need
 0 1 0 7 5 3 7 4 3
 2 0 0 3 2 2 1 2 2
 3 0 2 9 0 2 6 0 0
 2 1 1 2 2 2 0 1 1
```

```
 0 0 2 4 3 3 4 3 1
 Process executes in this order
 p2 p4 p5 p1 p3
 system is in saftey
Checking the bankers algorithm after the request
 Enter the request process number2
 Enter the values1
0
2
Execution of process after requestp2 p4 p5 p1 p3
 System is in safe state we can grant the request[it16@syamantaka ~]$
```

RESULT

Thus a UNIX C Program to implement deadlock avoidance using Banker's Algorithm was executed successfully.

EX.NO: 9 <u>IMPLEMENTATION OF DEADLOCK DETECTION ALGORITHM</u>
DATE:

AIM

To write a C unix program to implement the Deadlock Detection algorithm .

ALGORITHM

Step 1: Start the Program
Step 2: Obtain the required data through char and in data types.
Step 3: Enter the filename, index block.
Step 4: Print the file name index loop.
Step 5: File is allocated to the unused index blocks
Step 6: This is allocated to the unused linked allocation.
Step 7: Stop the program.

PROGRAM

```
#include <stdio.h>
#include <conio.h>
void main()
 {
 int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
 clrscr();
 printf("Enter total no of processes");
 scanf("%d",&tp);
 printf("Enter total no of resources");
 scanf("%d",&tr);
 printf("Enter claim (Max. Need) matrix\n");
 for(i=1;i<=tp;i++)
  {
 printf("process %d:\n",i);
 for(j=1;j<=tr;j++)
 scanf("%d",&c[i][j]);
```

```
}
printf("Enter allocation matrix\n");
for(i=1;i<=tp;i++)
{
printf("process %d:\n",i);
for(j=1;j<=tr;j++)
scanf("%d",&p[i][j]);
}
printf("Enter resource vector (Total resources):\n");
for(i=1;i<=tr;i++)
{
scanf("%d",&r[i]);
}
printf("Enter availability vector (available resources):\n");
for(i=1;i<=tr;i++)
{
scanf("%d",&a[i]);
temp[i]=a[i];
}
for(i=1;i<=tp;i++)
{
sum=0;
for(j=1;j<=tr;j++)
{
sum+=p[i][j];
}
if(sum==0)
{
m[k]=i;
k++;
}
}
for(i=1;i<=tp;i++)
{
for(l=1;l<k;l++)
if(i!=m[l])
{
flag=1;
for(j=1;j<=tr;j++)
if(c[i][j]<temp[j])
{
flag=0;
break;
}
}
if(flag==1)
{
m[k]=i;
k++;
for(j=1;j<=tr;j++)
```

```
temp[j]+=p[i][j];
 }
 }
printf("deadlock causing processes are:");
for(j=1;j<=tp;j++)
 {
found=0;
for(i=1;i<k;i++)
 {
if(j==m[i])
found=1;
 }
if(found==0)
printf("%d\t",j);
 }
getch();
}
```

OUTPUT

```
 Enter total no. of processes : 4
Enter total no. of resources : 5
Enter claim (Max. Need) matrix :
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter allocation matrix :
1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Enter resource vector (Total resources) :
2 1 1 2 1
Enter availability vector (available resources) :
0 0 0 0 1
deadlock causing processes are : 2 3
```

RESULT

Thus the C program to implement the deadlock detection algorithm was executed successfully.

EX.NO: 10a **IMPLEMENTATION OF FIFO PAGE REPLACEMENT ALGORITHM**
DATE:

AIM

To write a UNIX C program to implement FIFO page replacement algorithm.

ALGORITHM

Step 1: Start the process
Step 2: Declare the size with respect to page length
Step 3: Check the need of replacement from the page to memory
Step 4: Check the need of replacement from old page to new page in memory
Step 5: Format queue to hold all pages
Step 6: Insert the page require memory into the queue
Step 7: Check for bad replacement and page fault
Step 8: Get the number of processes to be inserted
Step 9: Display the values
Step 10: Stop the process

PROGRAM

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
for(i=0;i<no;i++)
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])
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("Page Fault Is %d",count);
return 0;
 }
```

OUTPUT

```
[it16@syamantaka ~]$ cc pr.c
[it16@syamantaka ~]$ ./a.out
 ENTER THE NUMBER OF PAGES:
20
 ENTER THE PAGE NUMBER :
7
0
1
2
0
3
0
4
2
3
0
3
2
1
2
0
1
7
0
1
ENTER THE NUMBER OF FRAMES :3
ref string page frames
7 7 -1 -1
0 7 0 -1
1 7 0 1
2 2 0 1
0
3 2 3 1
0 2 3 0
4 4 3 0
2 4 2 0
3 4 2 3
0 0 2 3
3
```

2
1 0 1 3
2 0 1 2
0
1
7 7 1 2
0 7 0 2
1 7 0 1

Page Fault Is 15

RESULT

Thus a UNIX C program to implement FIFO page replacement is executed successfully.

EX.NO: 10b **IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM**
DATE:

AIM

To write UNIX C program a program to implement LRU page replacement algorithm

ALGORITHM

Step 1: Start the process
Step 2: Declare the size
Step 3: Get the number of pages to be inserted
Step 4: Get the value
Step 5: Declare counter and stack
Step 6: Select the least recently used page by counter value
Step 7: Stack them according the selection.
Step 8: Display the values

Step 9: Stop the process

PROGRAM

```
#include<stdio.h>
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

```
[it16@syamantaka ~]$ cc pr.c
[it16@syamantaka ~]$ ./a.out
 ENTER THE NUMBER OF PAGES:
20
 ENTER THE PAGE NUMBER :
7
0
1
2
0
3
0
4
2
3
0
3
2
1
2
0
1
```

```
7
0
1
Enter no of frames:3
 7
 7 1
 7 1 2
 0 1 2
 0 3 2
 0 3 4
 0 2 4
 3 2 4
 3 2 0
 3 2 1
 0 2 1
 0 7 1
The no of page faults is 12
```

RESULT

Thus a UNIX C program to implement LRU page replacement is executed successfully.

EX.NO: 10c IMPLEMENTATION OF LFU PAGE REPLACEMENT ALGORITHM
DATE:

AIM

To write a UNIX C program to implement LFU page replacement algorithm.

ALGORITHM

Step 1: Start the program
Step 2: Declare the size
Step 3: Get the number of frames to be inserted
Step 4: Get the number of pages to be inserted and get the value.
Step 5: Get the page sequence
Step 6: Select the least frequently used page by counter value
Step 7: Stack them according the selection.
Step 8: Display the values
Step 9: Stop the process

PROGRAM

```
#include< stdio.h >
#include< conio.h >
struct frame
 {
int val;
int freq;
int pos;
 }f[30];
int no_frame,no_page,page_seq[100],page_fault=0;
```

```
void input()
 {
int i;
printf("\n\n least freqently used\n");
printf("\n enter no_of frame");
 scanf("%d",&no_frame);
printf("\n enter no_of pages");
 scanf("%d",&no_page);
printf("enter page sequence");
for(i=0;i<3;i++)
 scanf("%d",&page_seq[i]);
printf("\n");
for(i=0;i<3;i++)
 {
f[i].pos=-1;
f[i].val=-1;
f[i].freq=0;
 }
 }
void display()
 {
 static int
preev_page_fault=0; int i;
for(i=0;i<3;i++)
 {
if(f[i].val==-1)
 {
printf("0");
 }
else
 {
printf("%d",f[i].val);
 }
 }
if(preev_page_fault!=page_fault)
 {
printf("F");
preev_page_fault=page_fault;
 }
printf("\n");
 }
int search(int i)
 {
int j;
for(j=0;j<3;j++)
 {
if(f[j].val==page_seq[i])
return j;
 }
return -1;
```

```
}
int position()
{
int i,j=0,k;
for(i=0;i<3;i++)
{
if(f[i].pos==-1)
return i;
}
for(i=0;i<3;i++)
{
if(f[j].freq>f[i].freq)
{
j=i;
}
}
k=j;
for(i=0;i<3;i++)
{
if(f[j].freq==f[i].freq&&j!=i)
{
if(f[j].pos>f[i].pos)
j=i;
}
k=j;
return k;
}
return 0;
}
void LFU()
{
int i,k;
input();
for(i=0;i<3;i++)
{
k=search(i);
if(k!=-1)
{
f[k].freq++;
f[k].pos=i;
}
if(k==-1)
{
k=position();
f[k].pos=i;
f[k].val=page_seq[i];
f[k].freq=1;
page_fault++;
}
display();
```

```
}
}
void main()
{
clrscr();
LFU();
printf("\n no.of page faults %d",page_fault);
getch();
}
```

OUTPUT

```
Least frequently used

Enter no of frame 3

Enter no of pages 4

Enter page sequence 4
5
8

400F
450F
458F

No of page faults 3
```

RESULT

Thus a UNIX C program to implement LFU page replacement is executed successfully.

EX.NO: 11a INTERPROCESS COMMUNICATION USING PIPES DATE:

AIM

To write a UNIX C program to implement interprocess communication using pipes.

ALGORITHM

Step 1: Create a pipe structure using pipe() system call.Pipe() system call returns 2 file descriptors fd[0] and fd[1].fd[0] is opened for reading and fd[1] is opened for writing.

Step 2:Create a child process using fork() system call.

Step 3:Close the read end of the parent process using close().

Step 4:Write the data in the pipe using write().

Step 5:Close the write end of the child process using close().

Step 6:Read the data in the pipe using read()..

PROGRAM

```
#include<stdio.h>
int main()
 {
int fd[2],child;
char a[20];
printf("\nEnter the string to enter into the pipe:");
 scanf("%s",a);
pipe(fd);
child=fork();
```

```
if(!child)
 {
close(fd[0]);
write(fd[1],a,5);
 }
else
 {
close(fd[1]);
read(fd[0],a,5);
printf("\n The string retrieved from the pipe is %s\n",a);
 }
return 0;
 }
```

OUTPUT

```
[it3@localhost ~]$ vi iiiiiiiic.c
[it3@localhost ~]$ cc iiiiiiiic.c
[it3@localhost ~]$ ./a.out
EnTer the string to enter into the pipe:operatingsystem
 The string retrieved from the pipe is operatingsystem
```

RESULT

Thus a UNIX C program to implement interprocess communication using pipes is executed successfully.

EX.NO: 11b INTERPROCESS COMMUNICATION USING SHARED MEMORY
DATE:

AIM

To write a UNIX C program to implement interprocess communication using shared memory.

ALGORITHM

Step 1: Start the program.
Step 2: Create the shared memory for parent process using shmget() system
call. Step 3: Attach the shared memory to the child process.
Step 4: Create child process using fork ().
Step 5: Parent process writes the content in the shared memory.
Step 6: The child process reads the content from the shared memory.
Step 7: Detach and release the shared memory.
Step 8: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<fcntl.h>
main()
 {
```

```
char *shmptr;
int shmid,child,i;
shmid=shmget(2041,30,IPC_CREAT|0666);
shmptr=shmat(shmid,0,0);
child=fork();
if(!child)
{
printf("PARENT WRITING\n");
for(i=0;i<20;i++)
{
shmptr[i]='a'+i;
putchar(shmptr[i]);
}
wait(0);
}
else
{
printf("\nCHILD READING\n");
for(i=0;i<20;i++)
putchar(shmptr[i]);
shmdt(NULL);
shmctl(shmid,IPC_RMID,0);
}
}
```

OUTPUT

```
[it3@localhost ~]$ vi iiiii.c
[it3@localhost ~]$ cc iiiii.c
[it3@localhost ~]$ ./a.out
PARENT WRITING
abcdefghijklmnopqrst
CHILD READING
```

RESULT

Thus a UNIX C program to implement interprocess communication using shared memory is executed successfully.

EX.NO: 12 PAGING TECHNIQUE OF MEMORY MANAGEMENT SCHEME
DATE:

AIM

To write a UNIX C program to implement paging memory management scheme.

ALGORITHM

Step 1: Start the program
Step 2: Enter the number of pages and page size.
Step 3: Calculate the limit of logical address.
Step 4: Enter the logical address within the limit.
Step 5: Enter the page table entries (i.e)the page number and the relevant frame number.
Step 6: For the logical address, calculate the page number and frame number. Step 7:Calculate the physical address.
Step 8:Stop the program.

PROGRAM

```
#include<stdio.h>
main()
 {
int n,size,la,i,pno[10],fno[10],offset,pa,limit,p;
printf("Enter the number ofpages:");
scanf("%d",&n);
printf("Enter the size of page:");
```

```
scanf("%d",&size);
limit=n*size-1;
printf("Enter logical address within the limit %d:",limit);
scanf("%d",&la);
for(i=0;i<n;i++)
{
printf("Enter the pageno:");
scanf("%d",&pno[i]);
printf("Enter the frame no:");
scanf("%d",&fno[i]);
}
printf("pageno\tframeno");
for(i=0;i<n;i++)
{
printf("\n%d\t\t%d",pno[i],fno[i]);
}
p=la/size;
printf("\n The page number is %d",p);
offset=la%size;
printf("\nOffset is %d",offset);
pa=fno[p]*size+offset;
printf("\n The physical address is %d",pa);
}
```

OUTPUT

```
[it3@syamantaka ~]$ cc page.c
[it3@syamantaka ~]$ ./a.out
Enter the number of pages:4
Enter the size of page:4
Enter logical address within the limit 15:6
Enter the pageno:0
Enter the frame no:5
Enter the pageno:1
Enter the frame no:3
Enter the pageno:2
Enter the frame no:8
Enter the pageno:3
Enter the frame no:6
pageno frameno
0 5
1 3
2 8
3 6
 The page number is 1
Offset is 2
```

The physical address is 14

RESULT

Thus a UNIX C program to implement paging memory management scheme was executed successfully.

EX.NO: 13 THREADING AND SYNCHRONIZATION APPLICATIONS
DATE:

AIM

To write a UNIX C program to create the thread.

ALGORITHM

Step 1: Start the program
Step 2: Include the header file pthread.h
Step 3: Create the thread ID by using the pthread_create() function to create two threads Step 4: The starting function for both the threads is kept same.
Step 5: Inside the function 'createthread()', the thread uses pthread_self() and pthread_equal() functions to identify whether the executing thread is the first one or the second one as created.
Step 6: Also, Inside the same function 'createthread ()' a for loop is run so as to simulate some time consuming work.
Step 7: Compile the program with the –lpthread.
Step 8: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
void* createthread(void *arg)
{
unsigned long i = 0;
pthread_t id = pthread_self();
if(pthread_equal(id,tid[0]))
{
printf("\n First thread processing\n");
}
else
```

```
{
printf("\n Second thread processing\n");
}
for(i=0; i<(0xFFFFFFFF);i++);
return NULL;
}
int main(void)
{
int i = 0;
int err;
while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, & createthread, NULL);
if (err != 0)
printf("\ncan't create thread :[%s]", strerror(err));
else
printf("\n Thread created successfully\n");
i++;
}
sleep(5);
return 0;
}
```

OUTPUT

```
[it1@localhost ~]$ vi thread1.c
[it1@localhost ~]$ cc thread1.c -lpthread
[it1@localhost ~]$ ./thread1

Thread created successfully

 First thread processing

 Thread created successfully

 Second thread processing
```

RESULT

Thus a UNIX C program to create the thread was executed successfully.