**AL-AMEEN ENGINEERING COLLEGE**

**(AUTONOMOUS)**

**KARUNDEVANPALAYAM, ERODE – 638 104.**

**[[**

****

**LABORATORY RECORD**

**Name : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Register No. : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Class : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Subject & Code : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AL-AMEEN**

**ENGINEERING COLLEGE**

**(AUTONOMOUS)**

**KARUNDEVANPALAYAM, ERODE – 638 104.**

**[[**

**Department of**

**COMPUTER SCIENCE AND ENGINEERING**

**LABORATORY RECORD**

Name : ………………………………………………RollNo : …………………………

Reg. No: ……………………………………………… Year / Sem:…………………....

Certified that this is the Bonafide Record of work done by the above student in the ............................................………………………Laboratory during the year ………………….

**STAFF INCHARGE HEAD OF THE DEPARTMENT**

Submitted for the University Practical Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**CONTENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No.** | **Date** | **Name of the Experiments** | **Page No.** | **Marks** | **Signature with Date** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Internal Marks: Staff In-charge**

|  |  |
| --- | --- |
| **Ex.No: 1** | **BASIC COMMANDS IN UNIX** |
| **Date:** |

AIM

To study and execute Unix commands.

LOGIN

User has to authenticate himself by providing *username* and *password*. Once verified, a greeting and **$** prompt appears. The shell is now ready to receive commands from the user. Options suffixed with a hyphen (–) and arguments are separated byspace.

FILE COMMANDS

|  |  |
| --- | --- |
| **Command** | **Function** |
| cat >*filename* | To create a file with some contents. To end typing press Ctrl+d. The **>**symbol means redirecting output to a file. (**<**for input) |
| cat*filename* | Displays the file contents. |
| cat>>*filename* | Used to append contents to a file |
| cp*src des* | Copy files to given location. If already exists, it will be overwritten |
| cp –i*src des* | Warns the user prior to overwriting the destination file |
| cp –r *src des* | Copies the entire directory, all its sub-directories and files. |
| mv *old new* | To rename an existing file or directory. –i option can also be used |
| mv *f1 f2 f3 dir* | To move a group of files to a directory. |
| mv –v *old new* | Display name of each file as it is moved. |
| rm *file* | Used to delete a file or group of files. –i option can also be used |
| rm \* | To delete all the files in the directory. |
| rm –r \* | Deletes all files and sub-directories |
| rm –f \* | To forcibly remove even write-protected files |
| Ls | Lists all files and subdirectories (blue colored) in sorted manner. |
| ls*name* | To check whether a file or directory exists. |
| ls*name***\*** | Short-hand notation to list out filenames of a specificpattern. |
| ls –a | Lists all files including hidden files (files beginning with **.**) |
| ls –x*dirname* | To have specific listing of a directory. |
| ls –R | Recursive listing of all files in the subdirectories |
| ls –l | Long listing showing file access rights (read/write/execute-**rwx** for  user/group/others**-ugo**). |
| cmp *file1 file2* | Used to compare two files. Displays nothing if files are identical. |
| wc *file* | It produces a statistics of lines (**l**), words(**w**), and characters(**c**). |
| chmod *perm file* | Changes permission for the specified file. (r=4, w=2,x=1)  chmod 740 *file* sets all rights for user, read only for groups and no rights forothers |

GENERAL COMMANDS

|  |  |
| --- | --- |
| **Command** | **Function** |
| Date | Used to display the current system date and time. |
| date +%D | Displays date only |
| date +%T | Displays time only |
| date +%Y | Displays the year part of date |
| date +%H | Displays the hour part of time |
| Cal | Calendar of the current month |
| cal*year* | Displays calendar for all months of the specified year |
| cal*month year* | Displays calendar for the specified month of the year |
| Who | Login details of all users such as their IP, Terminal No, User name, |
| who am i | Used to display the login details of the user |
| Uname | Displays the Operating System |
| uname –r | Shows version number of the OS (kernel). |
| uname –n | Displays domain name of the server |
| echo$HOME | Displays the user's home directory |
| Bc | Basic calculator. Press Ctrl+dto quit |
| lp *file* | Allows the user to spool a job along with others in a print queue. |
| man*cmdname* | Manual for the given command. Press qto exit |
| history | To display the commands used by the user since log on. |
| exit | Exit from a process. If shell is the only process then logs out |

DIRECTORY COMMANDS

|  |  |
| --- | --- |
| **Command** | **Function** |
| Pwd | Path of the present working directory |
| mkdir*dir* | A directory is created in the given name under the current directory |
| mkdir*dir1 dir2* | A number of sub-directories can be created under onestroke |
| cd*subdir* | Change Directory. If the *subdir* starts with **/** then path starts from  **root** (absolute) otherwise from current working directory. |
| cd | To switch to the home directory. |
| cd / | To switch to the root directory. |
| cd .. | To move back to the parent directory |
| rmdir*subdir* | Removes an empty sub-directory. |

The commands can be combined using the pipeline (|) operator. For example, number of users logged in can be obtained as.

who | wc -l

Finally to terminate the unix session execute the command **exit** or **logout**.

**OUTPUT**

$ **date**

Wed Jan 8 13:03:47 IST2020

$ date +%D

01/08/20

$ date +%T

13:05:33

$ date +%Y

2011

$ date +%H

13

$ cal 01 2020

January 2020

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

$ **who**

root :0 Jan 808:41

ibm pts/0 Jan 8 13:00(scl-64)

cse4001 pts/3 Jan 8 13:18(scl-41.smkfomra.com)

$ **uname**

Linux

$ uname-r

2.4.20-8smp

$ uname-n

localhost.localdomain

$ echo $HOME

/home/ibm

$ echo $USER

ibm

$ **bc**

3+5

8

$ **pwd**

/home/ibm/shellscripts/loops

$ mkdir filter

$ **ls**

filter list.sh regexpr shellscripts

$ cd shellscripts/loops/

$

$ **cd**

$

$ **cd /**

$

$ **cd /home/ibm/shellscripts/loops/**

$ **cd ..**

$

$ rmdir filter

$ **ls**

list.sh regexpr shellscripts

$ cat > greet

hi cse

wishing u the best

$ cat greet

hi ece-a

wishing u the best

$ cat >> greet

bye

$ cat greet

hi cse

wishing u the best bye

$ **ls**

greet list.sh regexpr shellscripts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $ **ls -a** |  | | | |
| . | .bash\_logout | .canna | .gtkrc | regexpr .viminfo.tmp |
| .. | .bash\_profile | .emacs | .kde | shellscripts .xemacs |
| .bash\_history | .bashrc | greet | list.sh | .viminfo |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $ **ls -l** |  | | | | | | | |
| -rw-rw-r-- | 1 | vijai | vijai | 32 | Jan | 22 | 14:52 | greet |
| -rw-rw-r-- | 1 | vijai | vijai | 30 | Jan | 8 | 13:58 | list.sh |
| drwxrwxr-x | 2 | vijai | vijai | 4096 | Jan | 13 | 14:30 | regexpr |

$ cp greet ./regexpr/

$ **ls**

greet list.sh regexpr shellscripts

$ ls ./regexpr

demo greet

$ cp -i greet ./regexpr/

cp: overwrite 'greet'? n

$ mv greet greet.txt

$ **ls**

greet.txt list.sh regexpr shellscripts

$ mv greet.txt ./regexpr/

$ **ls**

list.sh regexpr shellscripts

$ rm -i \*.sh

rm: remove regular file 'fact.sh'? y rm: remove regular file 'prime.sh'?y

$ **ls**

list.sh regexpr shellscripts

$ wc list.sh

4 9 30list.sh

$ wc -l list.sh

4 list.sh

$ cmp list.sh fact.sh

list.sh fact.sh differ: byte 1, line 1

$ ls -l list.sh

-rw-rw-r-- 1vijai vijai 30Apr 4 13:58list.sh

$ chmod ug+x list.sh

$ ls -l list.sh

-rwxrwxr-- 1vijai vijai 8 Jan 4 13:58list.sh

$ chmod 740 list.sh

$ ls -l list.sh

-rwxr----- 1vijai vijai 8 Jan 4 13:58list.sh

RESULT

Thus the study and execution of basic Unix commands has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 2** | **SYSTEM CALLS OF UNIX OPERATING SYSTEM** |
| **Date:** |

**AIM**

To implement the system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir

SYSTEM CALLS

fork()

* The fork system call issued to create new process called *child* process.
  + The return value is 0 for a child process.
  + The return value is negative if process creation is unsuccessful.
  + For the parent process, return value is positive
* The child process is an exact copy of the parent process.
* Boththechildandparentcontinuetoexecutetheinstructionsfollowingforkcall.
* The child can start execution before the parent or vice-versa

getpid()and getppid()

* The getpid system call returns process ID of the calling process
* The getppid system call returns parent process ID of the calling process

wait()

* Thewaitsystemcallcausestheparentprocesstobeblockeduntilachildterminates.
* Whenaprocessterminates,thekernelnotifiestheparentbysendingtheSIGCHLDsignalto the parent.
* Withoutwait,theparentmayfinishfirstleavinga*zombie*child,tobeadoptedbyinitprocess

execl()

* Theexecfamilyoffunction(execl,execv,execle,execve,execlp,execvp)isusedbythe child process to load a program and execute.
* execl system call requires path, program name and null pointer

exit()

* Theexitsystemcallisusedtoterminateaprocesseithernormallyorabnormally
* Closes all standard I/O streams.

stat()

* Thestatsystemcallisusedtoreturninformationaboutafileasastructure.

opendir()

* The opendir system call is used to open a directory
  + It returns a pointer to the first entry
  + It returns NULL on error.

**readdir()**

* Thereaddirsystemcallisusedtoread adirectoryasa*dirent*structure
  + It returns a pointer pointing to the next entry in directorystream
  + It returns NULL if an error or end-of-fileoccurs.

**closedir()**

* The closedir system call is used to close the directorystream
* Write to a directory is done only by thekernel**.**

open()

* + - Used to open an existing file for reading/writing or to create a newfile.
    - Returns a file descriptor whose value is negative onerror.
    - The mandatory flags are O\_RDONLY, O\_WRONLY andO\_RDWR
    - Optional flags include O\_APPEND, O\_CREAT, O\_TRUNC,etc
    - The flags areORed.
    - The mode specifies permissions for thefile.

creat()

* + - Used to create a new file and open it forwriting.
    - It is replaced with open() with flags O\_WRONLY|O\_CREAT |O\_TRUNC

read()

* + - Reads no. of bytes from the file or from theterminal.
    - If read is successful, it returns no. of bytesread.
    - The file offset is incremented by no. of bytesread.
    - If end-of-file is encountered, it returns0.

write()

* + - Writes no. of bytes onto thefile.
    - Afterasuccessfulwrite,file'soffsetisincrementedbytheno.ofbyteswritten.
    - If any error due to insufficient storage space, writefails.

close()

* + - Closes a openedfile.
    - Whenprocessterminates,filesassociatedwiththeprocessareautomaticallyclosed.

**PROGRAM**

#include<stdio.h>

main(int arc,char\*ar[])  
  
{  
  
int pid;  
char s[100];  
pid=fork();  
  
if(pid<0)  
printf("error");  
  
else if(pid>0)  
{  
  wait(NULL);  
  printf("\n Parent Process:\n");  
  printf("\n\tParent Process id:%d\t\n",getpid());  
  execlp("cat","cat",ar[1],(char\*)0); error("can’t execute cat %s,",ar[1]);  
  
}  
  
else  
{  
  
printf("\nChild process:");  
printf("\n\tChildprocess parent id:\t %d",getppid());  
sprintf(s,"\n\tChild process id :\t%d",getpid());  
write(1,s,strlen(s));  
printf(" ");  
printf(" ");  
printf(" ");  
execvp(ar[2],&ar[2]);  
error("can’t execute %s",ar[2]);  
}  
  
}

**OUTPUT**

**$ cc syscalls.c**

**$ ./a.out**

Child process:

Child process id : 186

Childprocess parent id:185

Parent Process:

Parent Process id:185

RESULT

Thus the system calls of UNIX operating system been implemented successfully.

|  |  |
| --- | --- |
| **Ex.No: 3** | **SIMULATE UNIX COMMANDS** |
| **Date:** |

**AIM**

To simulate UNIX commands cp, ls, grep.

1. **CP COMMAND**

ALGORITHM

1. Get source and destination *filename* as command-lineargument.
2. Declare a buffer of size1KB
3. Open the source file in readonly mode using open systemcall.
4. If file does not exist, thenstop.
5. Create the destination file using creat systemcall.
6. If file cannot be created, thenstop.
7. File copy is achieved asfollows:
   1. Read1KBdatafromsourcefileandstoreontobufferusingreadsystemcall.
   2. Write the buffer contents onto destination file using write systemcall.
   3. If end-of-file then step 8 else step7a.
8. Close source and destination file using close systemcall.
9. Stop.

PROGRAM

/\* cp command simulation - copy.c \*/

#include <stdio.h>

#include <stdlib.h> #include <fcntl.h> #include<sys/stat.h>

#define SIZE 1024

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

{

int src, dst, nread; char buf[SIZE];

if (argc != 3)

{

printf("Usage: gcc copy.c -o copy\n"); printf("Usage: ./copy <filename><newfile> \n"); exit(-1);

}

if ((src = open(argv[1], O\_RDONLY)) == -1)

{

perror(argv[1]);

exit(-1);

}

if ((dst = creat(argv[2], 0644)) == -1)

{

perror(argv[1]);

exit(-1);

}

while ((nread = read(src, buf, SIZE)) >0)

{

if (write(dst, buf, nread) == -1)

{

printf("can't write\n"); exit(-1);

}

}

close(src); close(dst);

}

OUTPUT

$ gcc copy.c -o copy

$ ./copy hello hello.txt

1. **LS COMMAND**

ALGORITHM

1. Store path of current working directory using getcwd systemcall.
2. Scan directory of the stored path using scandir system call and sort the resultant array ofstructure.
3. Display dname member for all entries if it is not a hiddenfile.
4. Stop.

PROGRAM

/\* ls command simulation - list.c \*/

#include <stdio.h>

#include <dirent.h>

main()

{

struct dirent \*\*namelist; int n,i;

char pathname[100];

getcwd(pathname);

n = scandir(pathname, &namelist, 0, alphasort);

if(n < 0)

printf("Error\n"); else

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

if(namelist[i]->d\_name[0] != '.') printf("%-20s",namelist[i]->d\_name);

}

|  |  |  |  |
| --- | --- | --- | --- |
| **OUTPUT**  **$ gcc list.c** | **-o** | **list** |  |
| **$ ./list a.out** |  | **cmdpipe.c** | **consumer.c** |
| **dirlist.c ex6c.c fappend.c fork.c list producer.c**  **sjf.c** |  | **ex6a.c ex6d.c fcfs.c fread.c list.c rr.c**  **stat.c** | **ex6b.c exec.c fcreate.c hello pri.c simls.c**  **wait.c** |

1. **GREP COMMAND**

ALGORITHM

1. Get filename and search string as command-lineargument.
2. Open the file in read-only mode using open systemcall.
3. If file does not exist, thenstop.
4. Let length of the search string be*n*.
5. Read line-by-line untilend-of-file
   1. Check to find out the occurrence of the search string in a line by examining characters in the range 1–n, 2–n+1,etc.
   2. If search string exists, then print theline.
6. Close the file using close systemcall.
7. Stop.

PROGRAM

/\* grep command simulation - mygrep.c \*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

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

{

FILE \*fd;

char str[100]; char c;

int i, flag, j, m, k; char temp[30];

if(argc !=3)

{

printf("Usage: gcc mygrep.c –o mygrep\n");

printf("Usage: ./mygrep <search\_text><filename>\n");

exit(-1);

}

fd = fopen(argv[2],"r"); if(fd == NULL)

{

printf("%s is not exist\n",argv[2]);

exit(-1);

}

while(!feof(fd))

{

i = 0;

while(1)

{

c = fgetc(fd); if(feof(fd))

{

str[i++] = '\0'; break;

}

if(c == '\n')

{

str[i++] = '\0'; break;

}

str[i++] = c;

}

if(strlen(str) >= strlen(argv[1]))

for(k=0; k<=strlen(str)-strlen(argv[1]); k++)

{

for(m=0; m<strlen(argv[1]); m++) temp[m] = str[k+m];

temp[m] = '\0'; if(strcmp(temp,argv[1]) ==0)

{

printf("%s\n",str); break;

}

}

}

}

OUTPUT

$ gcc mygrep.c -o mygrep

$ ./mygrep printf dirlist.c

printf("Usage: ./a.out <dirname>\n"); printf("%s\n", dptr->d\_name);

RESULT

Thus the UNIX commands cp, ls, grep has been simulated successfully

|  |  |
| --- | --- |
| **Ex.No: 4** | **SHELL PROGAMMING** |
| **Date:** |

**AIM**

To write and execute basic Shell Programs in UNIX.

# A. GREATEST AMONG 3 NUMBERS

**ALGORITHM**

Step 1: Start the program.

Step 2: Read a, b,c

Step 3: Check a is greater than b and a is greater than c. If true goto step 4.

Step 4: Print A is greater else if goto step 5.

Step 5: Check b is greater than c. If true goto step 6.

Step 6: Print B is greater else goto step 7.

Step 7: Print C is greater.

Step 8: Stop the program.

**PROGRAM**

echo " Enter the three numbers"

read a b c

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

then

echo " A is greater"

elif [ $b -gt $c ]

then

echo " B is greater"

else

echo " C is greater"

fi

**OUTPUT**

$ sh great.sh

Enter the three numbers

8 6 4

A is greater

$ sh great.sh

Enter the three numbers

5 7 9

C is greater

**B. SWAPPING TWO NUMBERS**

**ALGORITHM**

Step 1 : Start the program.

Step 2 : Read a and b.

Step 3 : Print the two numbers of a and b

Step 4 : Assign c = a

Step 5 : Assign a = b

Step 6 : Assign b = c

Step 7 : Print the swapped numbers of a and b.

Step 8 : Stop the program.

**PROGRAM**

echo "Enter the two numbers"

read a

read b

echo " The Numbers are $a and $b"

c=$a

a=$b

b=$c

echo " The swapped numbers are $a and $b"

**OUTPUT**

$ sh swap.sh

Enter the two numbers

5

6

The Numbers are 5 and 6

The swapped numbers are 6 and 5

**C. FIBONACCI SERIES**

### ALGORITHM

Step 1: Start the program.

Step 2 : Assign a is –1, b is 1, c is 0.

Step 3: Print enter the limit.

Step 4: Read n value.

Step 5: Using for loop print fibonacci series up to n value.

Step 6: Inside the for loop add a and b, store it in c.

Step 7: Print c.

Step 8: Repeat the step loop gets closed.

Step 9: Stop the program.

**PROGRAM**

a=-1

b=1

c=0

echo " enter the limit"

read n

echo " Fibonacci Series"

for n in 1 2 3 4 5

do

c=`expr $a + $b`

echo $c

a=$b

b=$c

done

**OUTPUT**

$ sh fibo.sh

enter the limit

5

Fibonacci Series

0

1

1

2

3

**D. AREA AND CIRCUMFERENCE OF CIRCLE**

**ALGORITHM:**

Step 1: Start the program.

Step 2 : Read radius.

Step 3: Area= 3.14 \* r\*r

Circumference = 2\*3.14\*r

Step 4 : Print the value.

Step 5: Stop the program.

**PROGRAM:**

echo “Enter radius”

read radius

area=`echo 3.14 \\* $radius \\* $radius | bc`

cir=`echo 2 \\* 3.14 \\* $radius | bc`

echo Area of circle = $area and Circumference = $cir

**OUTPUT:**

$ sh area.sh

Enter radius

10

Area of circle = 314.00 and Circumference = 62.80

**E. ARMSTRONG NUMBER**

**ALGORITHM**

Step 1: Start the program.

Step 2: Read n value

Step 3 : Assign n to x

Step 4: Initialize sum is equal to 0.

Step 5: Check n is greater than 0. If true goto step 6.

Step 6: Find reminder value for n and store it in y.

Step 7: Multiply the y value three times and store it in z.

Step 8: z value added with sum and store it in sum.

Step 9: Divide remaining n value by 10.

Step 10: Repeat the step until loop gets closed.

Step 11: Check x is equal to sum. If true goto step 12

Step 12: Print the given number is armstrong else goto step 13.

Step 13: Print the given number is not an armstrong number.

Step 14: Stop the program.

**PROGRAM**

echo "Enter a number "

read n

x=$n

sum=0

while [ $n -gt 0 ]

do

y=`expr $n % 10`

z=`expr $y \\* $y \\* $y`

sum=`expr $sum + $z`

n=`expr $n / 10`

done

if [ $x -eq $sum ]

then

echo " $x is an Armstrong number "

else

echo "$x is not an Armstrong Number"

fi

**OUTPUT**

$ sh armstrong.sh

Enter a number: 153

153 is an Armstrong number

$ sh armstrong.sh

Enter a number: 25

25 is not an Armstrong Number

**F. PALINDROME**

**ALGORITHM:**

Step 1: Start the program.

Step 2: Read the variable str.

Step 3: Count the character from given string using wc command and store it in len

Step 4: Assign ptr variable is 1

Step 5: Get the character from string by ptr using cut command and store it in lchar

Step 6: Get the character from string by len using cut command and store it in rchar.

Step 7: Using if condition check whether the lchar and rchar are equal

Step 8: If satisfies the condition assign flag equal to 1 and decrement len value and

increment ptr value.

Step 9 : If not satisfies the condition print str is not a palindrome and assing flag

equal to 0 and break the statement.

Step 10 : Using if condition check whether flag is equal to 1, if it is print the given

string is palindrome.

Step 11 : Stop the program.

**PROGRAM**

echo "Enter the string"

read str

len=`expr $str | wc -c`

len=`expr $len - 1`

echo " $len "

ptr=1

while test $len -gt 0

do

lchar=`expr $str | cut -c $ptr`

rchar=`expr $str | cut -c $len`

if test $lchar = $rchar

then

flag=1

len=`expr $len - 1`

ptr=`expr $ptr + 1`

else

echo " $str is not a palindrome "

flag=0

break

fi

done

if test $flag -eq 1

then

echo " $str is a palindrome "

fi

**OUTPUT**

$ sh palindrome.sh

Enter the string

madam

5

madam is a palindrome

$ sh palindrome.sh

Enter the string

welcome

7

welcome is not a palindrome

**RESULT**

Thus the basic shell program has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 5** | **CPU SCHEDULING ALGORITHMS** |
| **Date:** |

**AIM**

To write a C program to implement the CPU scheduling algorithms – FCFS, SJF and Priority, Round robin scheduling.

**A.FCFS SCHEDULING**

**ALGORITHM**

1. Start the program.

2. Get the number of processes.

3. Get the burst time and arrival time of each process.

4. Start with the process having minimum arrival time, let other process wait in queue.

5. Calculate the turn around time and waiting time of the process. Repeat it for other processes.

6.  Display the values

S7. Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int n,a[10],b[10],t[10],w[10],g[10],i,m;

float att=0,awt=0;

            for(i=0;i<10;i++)

            {

                        a[i]=0; b[i]=0; w[i]=0; g[i]=0;

            }

printf("enter the number of process");

            scanf("%d",&n);

printf("enter the burst times");

            for(i=0;i<n;i++)

                scanf("%d",&b[i]);

     printf("\nenter the arrival times");

            for(i=0;i<n;i++)

                scanf("%d",&a[i]);

g[0]=0;

             for(i=0;i<10;i++)

                   g[i+1]=g[i]+b[i];

             for(i=0;i<n;i++)

            {

w[i]=g[i]-a[i];

                       t[i]=g[i+1]-a[i];

                       awt=awt+w[i];

                       att=att+t[i];

            }

      awt =awt/n;

            att=att/n;

            printf("\n\tprocess\twaiting time\tturn arround time\n");

            for(i=0;i<n;i++)

            {

                        printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);

            }

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

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

}

**OUTPUT**

$ cc fcfs.c

$ ./a.out

enter the number of process4

enter the burst times

4 9 8 3

enter the arrival times

0 2 4 3

process waiting time turn arround time

p0 0 4

p1 2 11

p2 9 17

p3 18 21

The average waiting time is 7.250000

The average turnaround time is 13.250000

**B.SJF SCHEDULING**

**ALGORITHM**

Step 1: Start the program.

Step 2: Initialize the variable.

Step 3: Read the number of process.

Step 4: Enter the runtime of each process using for loop.

Step 5: Process having shortest burst time will be executed using swapping

Step 6: Using while loop print the burst and waiting time.

Step 7: Calculate and print the average Turnaround time.

Step 8: Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int i,j,n,temp,wtp,twt,tatp,ttat,burst[10];

float avgtat,avgwt;

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

scanf("%d",&n);

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

{

printf("Enter the runtime of process[%d]:",i+1);

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

}

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

{

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

{

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

{

temp=burst[i];

burst[i]=burst[j];

burst[j]=temp;

}

}

}

i=ttat=twt=tatp=wtp=0;

printf("\nProcess no.\tRuntime\tTurnaroundtime\tWaitingTime");

while(i<n)

{

tatp=tatp+burst[i];

ttat=ttat+tatp;

twt=twt+wtp;

printf("\n%d\t\t%d\t\t%d\t\t%d",i+1,burst[i],wtp,tatp);

wtp=burst[i]+wtp;

i=i+1;

}

avgwt=twt\*1.0/n;

avgtat=ttat\*1.0/n;

printf("\n Avg TURNAROUND TIME=%0.2f",avgtat);

printf("\n Avg WAITING TIME=%0.2f",avgwt);

return 0;

}

**OUTPUT**

$ cc sjf.c

$ ./a.out

Enter the number of process :3

Enter the runtime of process[1]:12

Enter the runtime of process[2]:3

Enter the runtime of process[3]:2

Process no. Runtime Turnaroundtime WaitingTime

1 2 0 2

2 3 2 5

3 12 5 17

AVG TURNAROUND TIME=8.00

AVG WAITING TIME=2.33

**C. PRIORITY SCHEDULING**

**ALGORITHM**

Step 1: Start the program.

Step 2: Prompt for number of processor from the user.

Step 3: Get the priority process name, burst time and the corresponding priority of the process.

Step 4: Sort the process according to the priority and allocate the one with highest priority to be executed first.

Step 5: If number of process has the same priority then FCFS-scheduling algorithm is used.

Step 6: Calculate the average waiting time and total running time for the processors.

Step 7: Display the result.

Step 8: Stop the program.

**PROGRAM**

#include<stdio.h>

void swap(int j);

int priority[100];

char procno[100];

float burst[100];

int main()

{

int num,i;

float start=0.00,bt=0.00,sum=0.00,avgwait,wait=0.00; printf("Enter the number of

processes:\n");

scanf("%d",&num);

printf("Enter the process name,priority and burst time:\n");

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

{

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

scanf("\t%d",&priority[i]);

scanf("\t%f",&burst[i]);

}

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

{

if(priority[i]>priority[i+1])

{

swap(i); i=-1;

}

}

printf("\n\t\t Gantt chart \n\n"); printf("ProcessName\tPriority\tStart\tBurst \n");

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

{

bt=start+burst[i];

printf("%c\t\t\t%d\t\t%f\t%f\n",procno[i],priority[i],start,bt);

wait=wait+start;

start=start+burst[i];

}

avgwait=wait/num;

printf("Average waiting time of a process= %f\n",avgwait); printf("Total running

time of process= %f\n",start);

return 0;

}

void swap(int j)

{

char temp; float tbt; int pri;

temp=procno[j];

tbt=burst[j];

pri=priority[j];

procno[j]=procno[j+1];

priority[j]=priority[j+1];

}

**OUTPUT**

$ cc priority.c

$ ./a.out

Enter the number of processes:

5

Enter the process name,priority and burst time:

a 3 10

b 1 1

c 4 2

d 5 1

e 2 5

Gantt chart

ProcessName Priority Start Burst

b 1 0.000000 1.000000

e 2 1.000000 6.000000

a 3 6.000000 16.000000

c 4 16.000000 18.000000

d 5 18.000000 19.000000

Average waiting time of a process= 8.200000

Total running time of process= 19.000000

**D. ROUND ROBIN SCHEDULING**

**ALGORITHM**

Step 1: Start the program.

Step 2: Prompt for number of processor from the user.

Step 3: Get the priority process name, burst time and the corresponding priority of the process.

Step 4: Sort the process according to the priority and allocate the one with highest priority to be executed first.

Step 5: If number of process has the same priority then FCFS-scheduling algorithm is used.

Step 6: Calculate the average waiting time and total running time for the processors.

Step 7: Display the result.

Step 8: Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int st[10],bt[10],wt[10],tat[10],n,tq;

int i,count=0,swt=0,stat=0,temp,sq=0;

float awt=0.0,atat=0.0;

printf("Enter number of processes:");

scanf("%d",&n);

printf("Enter burst time for sequences:");

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

{

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

st[i]=bt[i];

}

printf("Enter time quantum:"); scanf("%d",&tq);

while(1)

{

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

{

temp=tq;

if(st[i]==0)

{

count++;

continue;

}

if(st[i]>tq)

st[i]=st[i]-tq;

else if(st[i]>=0)

{

temp=st[i];

st[i]=0;

}

sq=sq+temp;

tat[i]=sq;

}

if(n==count)

break;

}

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

{

wt[i]=tat[i]-bt[i]; swt=swt+wt[i]; stat=stat+tat[i];

}

awt=(float)swt/n;

atat=(float)stat/n;

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

printf("\nProcess\_no \tBurst time \tWait time \tTurn around time \n");

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

printf("\n%d\t\t%d\t\t%d\t\t%d ",i+1,bt[i],wt[i],tat[i]);

printf("\n Average wait time is %f\t\n Average turnaround time is %f\t\n",awt,atat);

}

**OUTPUT**

$ cc rr.c

$ ./a.out

Enter number of processes:4

Enter burst time for sequences:5

7

3

9

Enter time quantum:3

Gantt chart

Process\_no Burst time Wait time Turn around time

1 5 9 14

2 7 14 21

3 3 6 9

4 9 15 24

Average wait time is 11.000000

Average turnaround time is 17.000000

**RESULT**

Thus a C program to implement the CPU scheduling algorithms of FCFS, SJF and Priority, Round robin scheduling has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 6** | **IMPLEMENTATION OF SEMAPHORES** |
| **Date:** |

**AIM**

To write a C program to implement Semaphores using Producer Consumer Problem.

**ALGORITHM**

Step 1: Start the program.

Step 2: Declare global variables and functions to be used in the program.

Step 3: Get the choice from the user 1.Producer 2.Consumer 3.Exit.

Step 4: If Choice 1, Producer() functions is called and produces item.

Step 5: If Choice 2, Consumer() functions is called and item consumed.

Step 6: If Choice 3, exit() functions is called and process terminates.

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

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

main()

{

int n;

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",&n);

switch(n)

{

case 1:

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

producer();

else

printf("BUFFER IS FULL");

break;

case 2:

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

consumer();

else

printf("BUFFER IS EMPTY");

break;

case 3:

exit(0);

break;

}

}

}

int wait(int s)

{

return(--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

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

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

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

x--;

mutex=signal(mutex);

}

**OUTPUT**

$ cc semaphore.c

$ ./a.out

1.PRODUCER

2.CONSUMER

3.EXIT

ENTER YOUR CHOICE

1

Producer produces the item1

ENTER YOUR CHOICE

1

Producer produces the item2

ENTER YOUR CHOICE

1

Producer produces the item3

ENTER YOUR CHOICE

1

BUFFER IS FULL

ENTER YOUR CHOICE

2

Consumer consumes item3

ENTER YOUR CHOICE

2

Consumer consumes item2

ENTER YOUR CHOICE

2

Consumer consumes item1

ENTER YOUR CHOICE

2

BUFFER IS EMPTY

ENTER YOUR CHOICE

3

**$**

**RESULT**

Thus a C program to implement the Semaphores using Producer Consumer Problem has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 7** | **IMPLEMENTATION OF SHARED MEMORY AND IPC** |
| **Date:** |

**AIM**

To write a C program to implement Inter Process Communication using shared memory

**ALGORITHM**

Step 1: Start the process

Step 2: Initiate IPC communication using shmget() function at sender side

Step 3: Get the message to be sent to the receiver

Step 4: Copy the string in another variable

Step 5: Initiate the IPC at receiver side

Step 6: Receive the message using shared memory.

Step 7: Display the message sent by sender

Step 8: Stop the process

**PROGRAM**

**SENDER**

#include<stdio.h>

#include<string.h>

#include<sys/shm.h>

#include<sys/ipc.h>

#define size 32

int main()

{

int shmid;

char \*s,\*str;

printf("\nIPC message passing using shared memory sender");

shmid=shmget(60,size,IPC\_CREAT|0666);

str=shmat(shmid,0,0);

printf("\nEnter the message to be sent");

scanf("%s",s);

strcpy(str,s);

printf("\nYour mesage has been sent");

return 0;

}

**RECEIVER**

#include<stdio.h>

#include<sys/shm.h>

#include<sys/ipc.h>

#define size 32

int main()

{

printf("\nIPC message passing using shared memory-receiver");

int shmid;

char \*str;

shmid=shmget(60,size,IPC\_CREAT|0666);

str=shmat(shmid,0,0);

printf("\nReceived message is....");

puts(str);

return 0;

}

**OUTPUT:**

**Sender Window**

$ cc sender.c

$ ./a.out

IPC message passing using shared memory sender

Enter the message to be sent: hello

Your mesage has been sent

**Receiver Window**

$ cc receiver.c

$ ./a.out

IPC message passing using shared memory-receiver

Received message is....hello

**RESULT**

Thus a C program to implement Inter Process Communication using shared memory has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 8** | **IMPLEMENTATIONOF BANKER’S ALGORITHM FOR**  **DEADLOCK AVOIDANCE** |
| **Date:** |

**AIM**

To write a C program to implement Banker’s Algorithm for Deadlock Avoidance.

**ALGORITHM**

Step 1 : Start the program.

Step 2 : Get the number of process, N and number of resource types, M.

Step 3: Enter the available resources for each process.

Step 4: Enter the maximum demand of each process.

Step 5: Enter the allocation of resources for each process.

Step 6: If need[i,j]=k then process Pi may need k more instances of resource type Rj to complete its task.

Need[i,j] = Max[i,j] – Allocation[i,j]

Step 7: Apply safety algorithm to find whether the system is in a safe state or unsafe state.

Step 8: If the system is in safe state, apply resource-request algorithm to fulfill the request made by each process.

Step 9: Display the system as safe state or unsafe state with the safe sequence.

Step 10: Stop the process.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

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

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

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

{

printf("\nFor process %d : ", i + 1);

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

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

}

printf("\n\nEnter the allocation for each process : ");

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

{

printf("\nFor process %d : ",i + 1);

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

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

}

printf("\n\nEnter the Available Resources : ");

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

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

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

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

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

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

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

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

printf("\t\t");

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

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

printf("\n");

}

process = -1;

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

{

if(completed[i] == 0)//if not completed

{

process = i ;

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

{

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

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

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

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]);

printf(">\n");

}

else

printf("\nThe system is in an unsafe state!!");

}

**OUTPUT**

$ cc dlockavoid.c

$ ./a.out

Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :

For process 1 : 7 5 3

For process 2 : 3 2 2

For process 3 : 7 0 2

For process 4 : 2 2 2

For process 5 : 4 3 3

Enter the allocation for each process :

For process 1 : 0 1 0

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2

Enter the Available Resources : 3 3 2

Max matrix: Allocation matrix:

7 5 3 0 1 0

3 2 2 2 0 0

7 0 2 3 0 2

2 2 2 2 1 1

4 3 3 0 0 2

Process 2 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

7 0 2 3 0 2

2 2 2 2 1 1

4 3 3 0 0 2

Process 3 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

0 0 0 0 0 0

2 2 2 2 1 1

4 3 3 0 0 2

Process 4 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 3 3 0 0 2

Process 1 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 3 3 0 0 2

Process 5 runs to completion!

The system is in a safe state!!

Safe Sequence : < 2 3 4 1 5 >

**RESULT**

Thus a C program to implement Banker’s Algorithm for Deadlock Avoidance has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 9** | **IMPLEMENTATION OF BANKER’S ALGORITHM FOR**  **DEADLOCK DETECTION** |
| **Date:** |

**AIM**

To write a C program to implement Banker’s Algorithm for Deadlock Detection.

**ALGORITHM**

Step1: Start the program.

Step2: Get the values of resources and processes.

Step3: Get the avail value.

Step4: After allocation find the need value.

Step5: Check whether it’s possible to allocate.

Step6: If it is possible then the system is in safe state.

Step7: Else system is not in safety state.

Step8: If the new request comes then check that the system is in safety or not if we allow the request.

Step9: Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int cl[10][10],al[10][10],av[10],i,j,k,m,n,ne[10][10],flag=0;

printf("\nEnter the rows and columns of matrix");

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

printf("\nEnter the claim matrix:");

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

{

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

{

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

}

}

printf("\nEnter allocated matrix:");

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

{

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

{

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

}

}

printf("\nThe need matrix:\n");

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

{

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

{

ne[i][j]=cl[i][j]-al[i][j];

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

}

printf("\n");

}

printf("\nEnter avaliable matrix");

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

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

printf("\nClaim matrix:\n");

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

{

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

{

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

}

printf("\n");

}

printf("\n Allocated matrix:\n");

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

{

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

{

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

}

printf("\n");

}

printf("Available matrix:\n");

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

{

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

}

//for(k=0;k<m;k++)

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

{

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

{

if(av[j]>=ne[i][j])

flag=1;

else

flag=0;

}

}

if(flag==0)

printf("\nUnsafe State\n");

else

printf("\nSafe State\n");

}

**OUTPUT:**

$ cc dlockdect.c

$ ./a.out

Enter the rows and columns of matrix:4 3

Enter the claim matrix:

3 2 2

6 1 3

3 1 4

4 2 2

Enter allocated matrix:

1 0 0

5 1 1

2 1 1

0 0 2

The need matrix:

2 2 2

1 0 2

1 0 3

4 2 0

Enter avaliable matrix: 1 1 2

Claim matrix:

3 2 2

6 1 3

3 1 4

4 2 2

Allocated matrix:

1 0 0

5 1 1

2 1 1

0 0 2

Available matrix: 1 1 2

Safe State

**RESULT**

Thus a C program to implement Banker’s Algorithm for Deadlock Detection has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 10** | **IMPLEMENTATIONOFTHREADING AND SYNCHRONIZATION** |
| **Date:** |

**AIM**

To write a C program to implement the synchronization applications with threading.

**ALGORITHM**

Step1: Start the program.

Step 2: Initializing the Pthread libraries.

Step 3: Process of the synchronization techniques starts.

Step 4: Enter the user data.

Step 5: Enter starting position of each process.

Step 6: Enter data to which mapping address to be found.

Step 7: Calculate and display the physical address for the corresponding logical address.

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]; int counter;

void\* doSomeThing(void \*arg)

{

unsigned long i = 0; counter += 1;

printf("\n Job %d started\n", counter);

for(i=0; i<(0xFFFFFFFF);i++); printf("\n Job %d finished\n", counter);

return NULL;

}

int main(void)

{

int i = 0; int err;

while(i < 2)

{

err = pthread\_create(&(tid[i]), NULL, &doSomeThing, NULL);

if (err != 0)

printf("\ncan't create thread :[%s]", strerror(err)); i++;

}

pthread\_join(tid[0], NULL); pthread\_join(tid[1], NULL);

return 0;

}

**OUTPUT:**

$ cc thread.c

$ ./a.out

Job 1 started

Job 1 finished

Job 2 started

Job 2 finished

**RESULT**

Thus a C program to implement the synchronization applications with threading has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 11** | **IMPLEMENTATIONOFMEMORY ALLOCATION METHODS** |
| **Date:** |

Aim

To write a C program to allocate memory requirements for processes using first fit, best-fit or worst-fit allocation strategy.

Memory Management

* The first-fit, best-fit, or worst-fit strategy is used to select a free hole from the set of available holes.

1. First fit

* Allocate the first hole that is big enough.
* Searching starts from the beginning of set of holes.

Algorithm

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say*nh*.
3. Get the size of each hole
4. Get number of processes, say*np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
   1. If hole size > process size then
      1. Mark process as allocated to that hole.
      2. Decrement hole size by process size.
   2. Otherwise check the next from the set of hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Program

/\* First fit allocation - ffit.c \*/

#include <stdio.h>

Struct process

{

int size; int flag; intholeid;

} p[10];

struct hole

{

int size;

int actual;

} h[10];

main()

{

int i, np, nh,j;

printf("Enter the number of Holes : "); scanf("%d",&nh);

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

{

printf("Enter size for hole H%d : ",i); scanf("%d", &h[i].size);

h[i].actual= h[i].size;

}

printf("\nEnter number of process : " ); scanf("%d",&np);

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

{

printf("enter the size of process P%d : ",i); scanf("%d", &p[i].size);

p[i].flag = 0;

}

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

{

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

{

if(p[i].flag != 1)

{

if(p[i].size <=h[j].size)

{

p[i].flag = 1; p[i].holeid = j; h[j].size -=p[i].size;

}

}

}

}

printf("\n\tFirst fit\n"); printf("\nProcess\tPSize\tHole"); for(i=0; i<np; i++)

{

if(p[i].flag != 1)

printf("\nP%d\t%d\tNot allocated", i, p[i].size); else

printf("\nP%d\t%d\tH%d", i, p[i].size,p[i].holeid);

}

printf("\n\nHole\tActual\tAvailable"); for(i=0; i<nh ;i++)

printf("\nH%d\t%d\t%d", i, h[i].actual, h[i].size); printf("\n");

|  |  |
| --- | --- |
| } |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OUTPUT**  **Enter Enter Enter Enter Enter Enter** | **the number of Holes : 5 size for hole H0 : 100 size for hole H1 : 500 size for hole H2 : 200 size for hole H3 : 300 size for hole H4 :600** | | |  |
| **Enter** | **number of process : 4** | | |  |
| **enter** | **the size of process P0:** | | | **212** |
| **enter** | **the size of process P1:** | | | **417** |
| **enter** | **the size of process P2:** | | | **112** |
| **enter** | **the size of process P3:** | | | **426** |
|  | **First fit** | | |  |
| **Process** | | **PSize** | **Hole** | |
| **P0** | | **212** | **H1** | |
| **P1** | | **417** | **H4** | |
| **P2** | | **112** | **H1** | |
| **P3** | | **426** | **Not allocated** | |
| **Hole** | | **Actual** | **Available** | |
| **H0** | | **100** | **100** | |
| **H1** | | **500** | **176** | |
| **H2** | | **200** | **200** | |
| **H3** | | **300** | **300** | |
| **H4** | | **600** | **183** | |

b. Best fit

* Allocate the smallest hole that is big enough.
* The list of free holes is kept sorted according to size in ascending order.
* This strategy produces smallest left over holes

Algorithm

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say*nh*.
3. Get the size of each hole
4. Get number of processes, say*np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
   1. Sort the holes according to their sizes in ascending order
   2. If hole size > process size then
      1. Mark process as allocated to that hole.
      2. Decrement hole size by process size.
   3. Otherwise check the next from the set of sorted hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Program

#include <stdio.h>

Struct process

{

int size; int flag; intholeid;

} p[10];

struct hole

{

int hid; int size; intactual;

} h[10];

main()

{

int i, np, nh,j;

void bsort(struct hole[], int); printf("Enter the number of Holes : "); scanf("%d",&nh);

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

{

printf("Enter size for hole H%d : ",i); scanf("%d", &h[i].size);

h[i].actual= h[i].size; h[i].hid =i;

}

printf("\nEnter number of process : " ); scanf("%d",&np);

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

{

printf("enter the size of process P%d : ",i); scanf("%d", &p[i].size);

p[i].flag = 0;

}

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

{

bsort(h, nh); for(j=0; j<nh;j++)

{

if(p[i].flag != 1)

{

if(p[i].size <=h[j].size)

{

p[i].flag = 1; p[i].holeid =h[j].hid;

h[j].size -=p[i].size;

}

}

}

}

printf("\n\tBest fit\n"); printf("\nProcess\tPSize\tHole"); for(i=0; i<np; i++)

{

if(p[i].flag != 1)

printf("\nP%d\t%d\tNot allocated", i, p[i].size); else

printf("\nP%d\t%d\tH%d", i, p[i].size,p[i].holeid);

}

printf("\n\nHole\tActual\tAvailable"); for(i=0; i<nh ;i++)

printf("\nH%d\t%d\t%d", h[i].hid, h[i].actual, h[i].size);

printf("\n");

}

void bsort(struct hole bh[], int n)

{

struct hole temp; int i,j;

for(i=0; i<n-1; i++)

{

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

{

if(bh[i].size > bh[j].size)

{

temp = bh[i]; bh[i] = bh[j]; bh[j] = temp;

}

}

}

}

Output

Enter the number of Holes : 5 Enter size for hole H0 : 100 Enter size for hole H1 : 500 Enter size for hole H2 : 200 Enter size for hole H3 : 300 Enter size for hole H4 : 600 Enter number of process :4

enter the size of process P0 : 212 enter the size of process P1 : 417 enter the size of process P2 : 112 enter the size of process P3 :426

Best fit

|  |  |  |
| --- | --- | --- |
| **Process** | **PSize** | **Hole** |
| **P0** | **212** | **H3** |
| **P1** | **417** | **H1** |
| **P2** | **112** | **H2** |
| **P3** | **426** | **H4** |
| **Hole** | **Actual** | **Available** |
| **H1** | **500** | **83** |
| **H3** | **300** | **88** |
| **H2** | **200** | **88** |
| **H0** | **100** | **100** |
| **H4** | **600** | **174** |

RESULT

Thus a C program to allocate memory requirements for processes using first-fit, best-fit or worst-fit allocation strategy has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 12** | **IMPLEMENTATION OF PAGING TECHNIQUE OF MEMORY MANAGEMENT** |
| **Date:** |

**AIM**

To write a C program to implement Paging Technique of memory management.

**ALGORITHM**

Step1: Start the program.

Step 2: Enter the page size.

Step3: Enter the logical memory address.

Step 3: Enter the user data.

Step 4: Enter starting position of each page.

Step 5: Enter data to which mapping address to be found.

Step 6: Calculate and display the physical address for the corresponding logical address.

Step 7: Stop the program.

**PROGRAM**

#include<stdio.h>

void main()

{

int a[20],lmem,pmem,ch,psize,str,b[35],c[10];

int i,j,k,INDEX,page,frame,addr;

printf("\nEnter page size: ");

scanf("%d",&psize);

printf("\nEnter logical memory in bytes: ");

scanf("%d",&lmem);

printf("\nEnter physical memory: ");

scanf("%d",&pmem);

printf("\nEnter user data: ");

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

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

for(i=0;i<32;i++)

{

b[i]=-1;

}

for(i=0;i<lmem/psize;i++)

{

printf("\nEnter starting position of each page:");

scanf("%d",&str);

c[i]=str/4;

for(j=str,k=i\*psize;j<(str+psize),k<(i\*psize)+psize;j++,k++)

{

b[j]=a[k];

}

}

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

{

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

}

printf("\nEnter dat to which mapping addr to be found: ");

scanf("%d",&ch);

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

{

if(ch==a[i])

{

INDEX=i;

page=INDEX/psize;

frame=c[page];

addr=(frame\*psize)+(INDEX%psize);

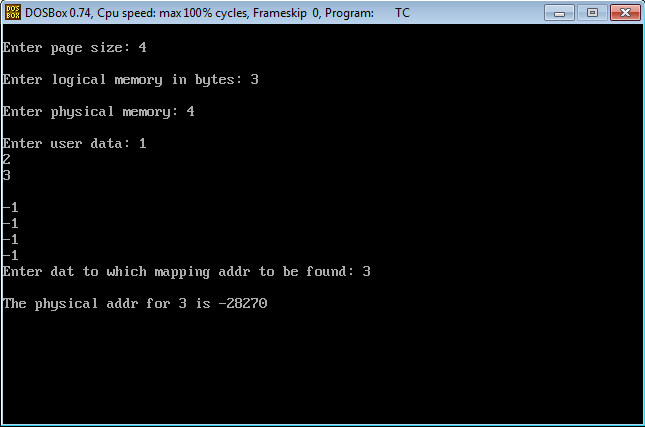
}

}

printf("\nThe physical addr for %d is %d ",ch,addr);

}

**OUTPUT:**



**RESULT**

Thus a C program to implement Paging Technique of memory management has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 13** | **IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHM** |
| **Date:** |

**AIM**

To implement page replacement algorithm using FIFO, LRU, LFU

**A. FIFO**

**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: Forma 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("\nPage Fault Is %d\n",count);

return 0;

}

**OUTPUT:**

$ cc fifo.c

$ ./a.out

ENTER THE NUMBER OF PAGES: 4

ENTER THE PAGE NUMBER: 7 2 1 0

ENTER THE NUMBER OF FRAMES: 3

ref string page frames

7 7 -1 -1

2 7 2 -1

1 7 2 1

0 0 2 1

Page Fault Is 4

$ cc fifo.c

$ ./a.out

ENTER THE NUMBER OF PAGES: 10

ENTER THE PAGE NUMBER: 7 2 1 0 3 4 2 3 2 1

ENTER THE NUMBER OF FRAMES: 3

ref string page frames

7 7 -1 -1

2 7 2 -1

1 7 2 1

0 0 2 1

3 0 3 1

4 0 3 4

2 2 3 4

3

2

1 2 1 4

Page Fault Is 8

$

**B. LRU**

**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:**

$ cc lru.c

$ ./a.out

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1

Enter no of frames:3

7

7 5

7 5 9

4 5 9

4 3 9

4 3 7

9 3 7

9 6 7

9 6 2

1 6 2

The no of page faults is 10

$

**RESULT**

Thus a C program to implement page replacement algorithm has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 14** | **IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUES** |
| **Date:** |

**AIM**

To write a C program to implement File Organization Techniques (Singlelevel directory, twolevel directory, hierarchical level directory).

**A. SINGLE LEVEL DIRECTORY**

**ALGORITHM**

Step 1: Start the program.

Step 2: Initialize the graphics setting and declare the required variables.

Step 3: Get the number of files to be created under the single level representation of files.

Step 4: Get the name of the files one by one from the user.

Step 5: Create a Root Directory in rectangle shape.

Step 6: Represent files ellipse shape below that folder.

Step 7: Stop the process.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm,count,i,j,mid,cir\_x;

char fname[10][20];

clrscr();

initgraph(&gd,&gm,"c:\\tc\\bgi");

cleardevice();

setbkcolor(GREEN);

puts("Enter no of files do u have?");

scanf("%d",&count);

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

{

cleardevice();

setbkcolor(GREEN);

printf("Enter file %d name",i+1);

scanf("%s",fname[i]);

setfillstyle(1,MAGENTA);

mid=640/count;

cir\_x=mid/3;

bar3d(270,100,370,150,0,0);

settextstyle(2,0,4);

settextjustify(1,1);

outtextxy(320,125,"RootDirectory");

setcolor(BLUE);

for(j=0;j<=i;j++,cir\_x+=mid)

{

line(320,150,cir\_x,250);

fillellipse(cir\_x,250,30,30);

outtextxy(cir\_x,250,fname[j]);

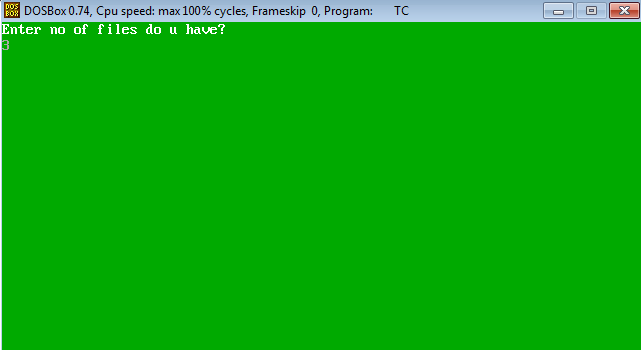
}

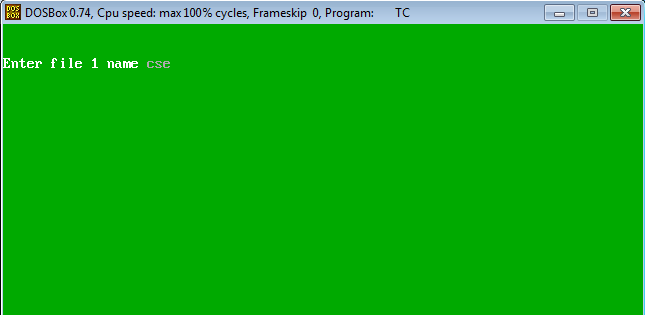
}

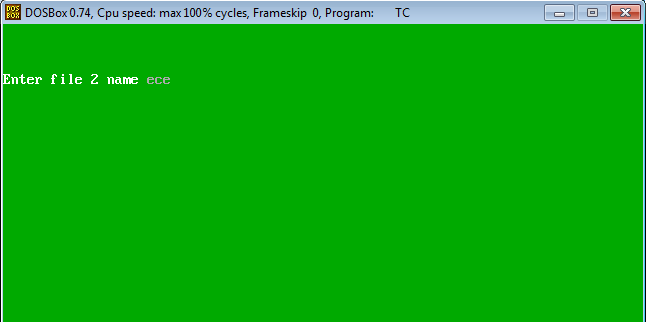
getch();

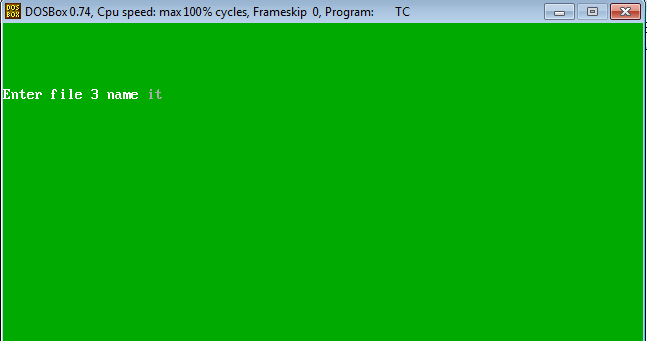
}

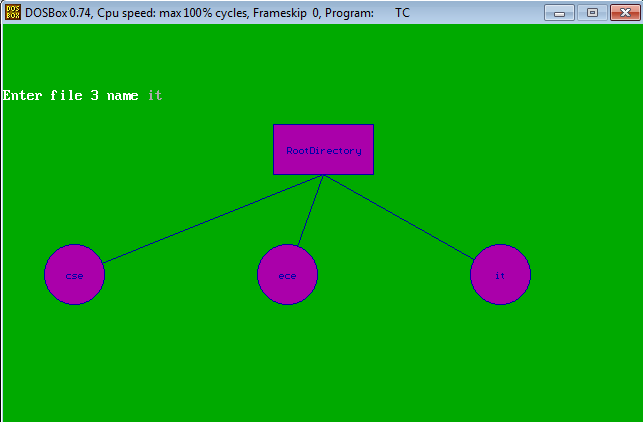
**OUTPUT**

****

****

****

****

****

**B. TWO LEVEL DIRECTORY**

**ALGORITHM**

Step 1: Start the program.

Step 2: Create a structure ‘tree\_element’ with required structure members.

Step 3: Initialize the graphics setting.

Step 4: Define the function create, get numbers of files and file names from the user.

Step 5: Display the values in two level directory format.

Step 6: Stop the process.

**PROGRAM**

#include<stdio.h>

#include<graphics.h>

struct tree\_element

{

char name[20];

int x,y,ftype,lx,rx,nc,level;

struct tree\_element \*link[5];

};

typedef struct tree\_element node;

void main()

{

int gd=DETECT,gm;

node \*root;

root=NULL;

clrscr();

create(&root,0,"null",0,639,320);

clrscr();

initgraph(&gd,&gm,"c:\\tc\\bgi");

display(root);

getch();

closegraph();

}

create(node \*\*root,int lev,char \*dname,int lx,int rx,int x)

{

int i,gap;

if(\*root==NULL)

{

(\*root)=(node\*)malloc(sizeof(node));

printf("enter name of dir/file(under %s):",dname);

fflush(stdin);

gets((\*root)->name);

if(lev==0||lev==1)

(\*root)->ftype=1;

else

(\*root)->ftype=2;

(\*root)->level=lev;

(\*root)->y=50+lev\*50;

(\*root)->x=x;

(\*root)->lx=lx;

(\*root)->rx=rx;

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

(\*root)->link[i]=NULL;

if((\*root)->ftype==1)

{

if(lev==0||lev==1)

{

if((\*root)->level==0)

printf("How many users");

else

printf("how many files");

printf("(for %s):",(\*root)->name);

scanf("%d",&(\*root)->nc);

}

else

(\*root)->nc=0;

if((\*root)->nc==0)

gap=rx-lx;

else

gap=(rx-lx)/(\*root)->nc;

for(i=0;i<(\*root)->nc;i++)

create(&((\*root)->link[i]),lev+1,(\*root)->name,lx+gap\*i,lx+gap\*i+gap,lx+gap\*i+gap/2);

}

else

(\*root)->nc=0;

}

}

display(node \*root)

{

int i;

settextstyle(2,0,4);

settextjustify(1,1);

setfillstyle(1,BLUE);

setcolor(14);

if(root!=NULL)

{

for(i=0;i<root->nc;i++)

{

line(root->x,root->y,root->link[i]->x,root->link[i]->y);

}

if(root->ftype==1)

bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);

else

fillellipse(root->x,root->y,20,20);

outtextxy(root->x,root->y,root->name);

for(i=0;i<root->nc;i++)

{

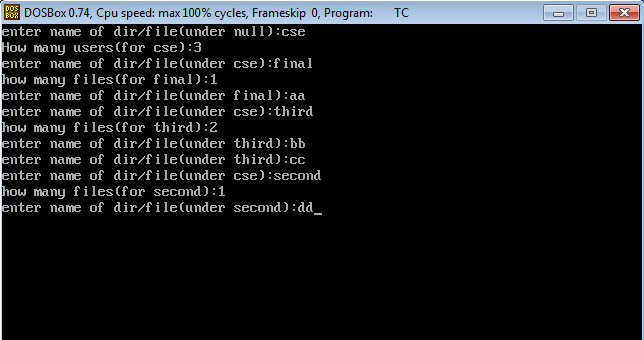
display(root->link[i]);

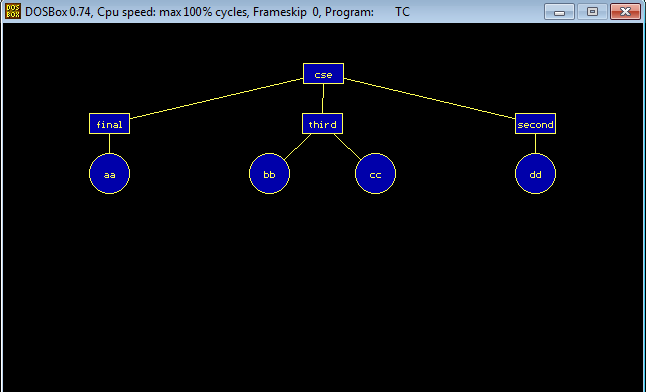
}

}

}

**OUTPUT**

****

****

**C. HIERARCHICAL LEVEL DIRECTORY**

**ALGORITHM**

Step 1: Start the program.

Step 2: Create a structure ‘tree element’ with required structure members.

Step 3: Initialize the graphics setting.

Step 4: Get numbers of folders, sub folders and files from the user.

Step 5: Display the folders and files in hierarchical manner.

Step 6: Stop the process.

**PROGRAM**

#include<stdio.h>

#include<graphics.h>

struct tree\_element

{

char name[20];

int x,y,ftype,lx,rx,nc,level;

struct tree\_element \*link[5];

};

typedef struct tree\_element node;

void main()

{

int gd=DETECT,gm;

node \*root; root=NULL;

clrscr();

create(&root,0,"root",0,639,320);

clrscr();

initgraph(&gd,&gm,"c:\\tc\\BGI");

display(root);

getch();

closegraph();

}

create(node \*\*root,int lev,char \*dname,int lx,int rx,int x)

{

int i,gap;

if(\*root==NULL)

{

(\*root)=(node \*)malloc(sizeof(node));

printf("Enter name of dir/file(under %s) : ",dname);

fflush(stdin);

gets((\*root)->name);

printf("enter 1 for Dir/2 for file :");

scanf("%d",&(\*root)->ftype);

(\*root)->level=lev;

(\*root)->y=50+lev\*50;

(\*root)->x=x;

(\*root)->lx=lx;

(\*root)->rx=rx;

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

(\*root)->link[i]=NULL;

if((\*root)->ftype==1)

{

printf("No of sub directories/files(for %s):",(\*root)->name);

scanf("%d",&(\*root)->nc);

if((\*root)->nc==0)

gap=rx-lx;

else

gap=(rx-lx)/(\*root)->nc;

for(i=0;i<(\*root)->nc;i++)

create(&((\*root)->link[i]),lev+1,(\*root)->name,lx+gap\*i,lx+gap\*i+gap,lx+gap\*i+gap/2);

}

else

(\*root)->nc=0;

}

}

display(node \*root)

{

int i;

settextstyle(2,0,4);

settextjustify(1,1);

setfillstyle(1,BLUE);

setcolor(14);

if(root !=NULL)

{

for(i=0;i<root->nc;i++)

{

line(root->x,root->y,root->link[i]->x,root->link[i]->y);

}

if(root->ftype==1)

bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);

else

fillellipse(root->x,root->y,20,20);

outtextxy(root->x,root->y,root->name);

for(i=0;i<root->nc;i++)

{

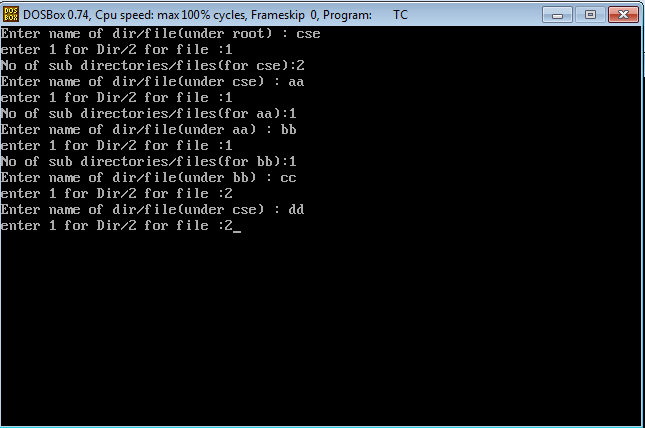
display(root->link[i]);

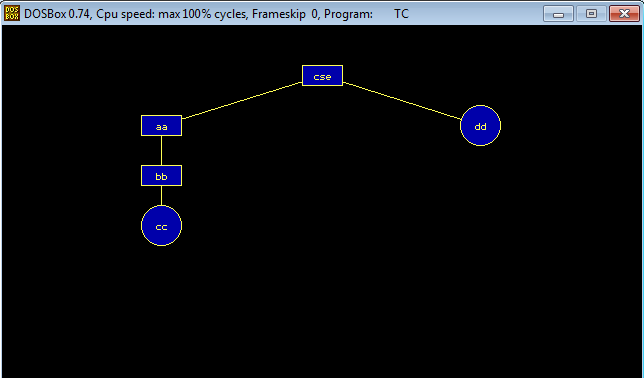
}

}

}

**OUTPUT**

****

****

**RESULT**

Thus a C program to implement File Organization Techniques (Single level directory, two level directory, hierarchical level directory) has been executed successfully.

|  |  |
| --- | --- |
| **Ex.No: 15** | **IMPLEMENTATION OF FILE ALLOCATION STRATEGIES** |
| **Date:** |

**AIM**

To write a C program to implement the Sequential, Indexed and Linked File Allocation Strategies.

**A. SEQUENTIAL FILE ALLOCATION**

**ALGORITHM**

Step 1: Start the program.

Step 2: Get the number of files to be allocated

Step 3: Get number of blocks, starting block for each file to be allocated.

Step 4: Allocate the file in sequential order as user enters.

Step 5: Display file name, start block and length of the file.

Step 6: Display details of file tat user wants to view.

Step 7: Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int n,i,j,b[20],sb[20],t[20],x,c[20][20];

int ch;

printf("Enter no.of files:");

scanf("%d",&n);

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

{

printf("Enter no. of blocks occupied by file%d:",i+1);

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

printf("Enter the starting block of file%d:",i+1);

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

t[i]=sb[i];

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

c[i][j]=sb[i]++;

}

printf("Filename\tStart block\tlength\n");

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

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

do

{

printf("Enter file name:");

scanf("%d",&x);

printf("File name is:%d\n",x);

printf("length is:%d\n",b[x-1]);

printf("blocks occupied:");

for(i=0;i<b[x-1];i++)

printf("%4d",c[x-1][i]);

printf("*\n*Do you want to continue(1/0):");

scanf("%d",&ch);

}while(ch!=0);

}

**OUTPUT**

$ cc sequentialfile.c

$ ./a.out

Enter no.of files:3

Enter no. of blocks occupied by file1:5

Enter the starting block of file1:1

Enter no. of blocks occupied by file2:3

Enter the starting block of file2:8

Enter no. of blocks occupied by file3:8

Enter the starting block of file3:3

Filename Start block length

1 1 5

2 8 3

3 3 8

Enter file name:2

File name is:2

length is:3

blocks occupied: 8 9 10

Do you want to continue(1/0):1

Enter file name:1

File name is:1

length is:5

blocks occupied: 1 2 3 4 5

Do you want to continue(1/0):1

Enter file name:3

File name is:3

length is:8

blocks occupied: 3 4 5 6 7 8 9 10

Do you want to continue(1/0):0

$

**B. INDEXED FILE ALLOCATION**

**ALGORITHM**

Step 1: Start the program.

Step 2: Get the number of files to be allocated

Step 3: Get starting block, size, number of blocks and individual block values.

Step 4: Display file name, index and length of the file.

Step 5: Display details of file tat user wants to view.

Step 6: Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int n,m[20],i,j,sb[20],s[20],b[20][20],x;

printf("Enter no. of files:");

scanf("%d",&n);

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

{

printf("Enter starting block and size of file%d:",i+1);

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

printf("Enter blocks occupied by file%d:",i+1);

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

printf("Enter blocks of file%d:",i+1);

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

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

}

printf("\nFile\t Index\t Length\n");

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

printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);

printf("\nEnter file name:");

scanf("%d",&x);

printf("\nFile name is:%d\n",x);

i=x-1;

printf("\nIndex is:%d",sb[i]);

printf("\nBlock occupied are:");

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

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

}

**OUTPUT**

$ cc indexedfile.c

$ ./a.out

Enter no. of files:3

Enter starting block and size of file1:1 10

Enter blocks occupied by file1:3

*E*nter blocks of file1:2 3 4

Enter starting block and size of file2:5 20

Enter blocks occupied by file2:4

*E*nter blocks of file2:5 6 7 8

Enter starting block and size of file3:12 30

Enter blocks occupied by file3:2

Enter blocks of file3:1 2

File Index Length

1 1 3

2 5 4

3 12 2

Enter file name:2

File name is:2

Index is:5

Block occupied are: 5 6 7 8

$

**C. LINKED FILE ALLOCATION**

**ALGORITHM**

Step 1: Start the program.

Step 2: Declare a structure with required structure members.

Step 3: Get the number of files to be allocated.

Step 4: Get file name, starting block, size, number of blocks and individual block values one by one.

Step 5: Display file name, Start block, Size and blocks of the file.

Step 6: Display details of file in which blocks of files in linked list format.

Step 7: Stop the program.

**PROGRAM**

#include<stdio.h>

struct file

{

char fname[10];

int start,size,block[10];

}f[10];

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]);

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**

$ cc linkedfile.c

$ ./a.out

Enter no. of files:2

Enter file name:Asif

Enter starting block:5

Enter no.of blocks:3

Enter block numbers:2 3 4

Enter file name:Prabakaran

Enter starting block:10

Enter no.of blocks:4

Enter block numbers:5 6 7 8

File Start Size Block

Asif 5 3 2--->3--->4

Prabakaran 10 4 5--->6--->7--->8

$

**RESULT**

Thus a C program to implement the Sequential, Indexed and Linked File Allocation Strategies has been executed successfully.