***OPERATING SYSTEMS LAB MANUAL***

**AUTHORS:**

**1.Ms.K.Muthukarupayee,AP/IT**

**2. Ms.J.Sangeethapriya,AP/IT**

**3. Ms.A.Sheelavathi,AP/IT**

**Saranathan College of Engineering,Trichirappalli**

**LIST OF EXPERIMENTS**

1. Learning about Basics UNIX commands
2. Write the programs for the following system calls using UNIX operating system for the following commands fork, exec, getpid, exit, wait, close, stat, opendir, readdir
3. To develop the C programs to simulate UNIX commands for cp, ls, grep, etc.
4. Shell Programming
5. To develop the C programs to implement the various CPU Scheduling Algorithms
6. Write a program to Implementation of Semaphores
7. Write a program to Implementation of Shared memory and IPC
8. To develop the Bankers Algorithm for Deadlock Avoidance
9. To write the C program to Implementation of Deadlock Detection Algorithm
10. Write C program to implement Threading & Synchronization Applications
11. Write C program for the following Memory Allocation Methods for fixed partition
12. Implementation of First Fit
13. Implementation of Worst Fit
14. Implementation of Best Fit
15. Write C program for the implementation of Paging Technique of Memory Management
16. Write C program for the implementation of the following Page Replacement Algorithms a) FIFO b) LRU c) LFU . Implementation of the various File Organization Techniques
17. Implementation of the following File Allocation Strategies
    1. Sequential b) Indexed c) Linked

EX:NO:1A: BASIC UNIX COMMANDS

a) **date**–used to check the date and time syn: $date

|  |  |  |  |
| --- | --- | --- | --- |
| Format | Purpose | Example | Result |
| +%m | Used To display only month | $date+%m | 06 |
| +%h | Used To display month name | $date+%h | June |
| +%d | Used To display day of month | $date+%d | O1 |
| +%y | Used To display last two digits of the year | $date+%y | 09 |
| +%H | Used To display hours | $date+%H | 10 |
| +%M | Used To display minutes | $date+%M | 45 |
| +%S | Used To display seconds | $date+%S | 55 |

b) **cal**–used to display the calendar

syn: $cal2 2009

c) **echo**–used to print the message on the screen.

Syn: $echo“text”

d) **ls**–used to list the files.Your files are kept in a directory.

Syn: $ls

ls–s All files(include files with prefix)

ls–l Long detail(provide file statistics)

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–rReverse Order

ls–f Mark directories with /,executable with\* , symbolic links with @, local sockets with =, named pipes(FIFOs)with |

ls–s Show file size

ls– h “Human Readable”, show files izeinKilo Bytes & Mega Bytes (h can be used together with –l or -s)

ls[a-m]\*List all the files whose name begin with alphabets From‘a’ to ‘m’

ls[a]\* List all the files whose name begins with‘a’or‘A’

Eg: $ls>mylist

Output of ‘ls’command is stored to disk file named‘mylist’

e) **lp**–to take the print out ssyn: $lp filename

f) **man**–used to provide manual help on every UNIX commands.

Syn: $man unix command

$man cat

g) **who** &**who ami**–it displays data about all users who have logged into the system currently.The next command displays about current user only.Syn: $who

$who ami

h) **uptime**–tells you how long the computer has been running since its last reboot or power-off.

Syn: $uptime

i) **uname**–it displays the system information such as hardware platform, system name and processor,OS type.

Syn: $uname–a

j) **hostname**–displays and set system host namesyn: $ hostname

k) **bc**–stands for ‘best calculator’

$bc

10/2\*3

15

quit

**Result**: Thus the display commands were executed successfully

**EX:NO:1B FILEMANIPULATIONCOMMANDS**

.

a) **cat**–This command is used to create,view and concatenate the files.

**Creation**:

Syn: $cat>filename

**Viewing**:

Syn: $catfile name

**Add a text to an existing file:**

Syn: $cat>>filename

**Concatenate**:

Syn: $cat file1 file2>file3

$cat file1 file2>>file3(overwriting of file3)

b) **grep**–this command is used to search a given particular word or pattern related to that corresponding word from the file.

Syn: $grep search word filename

Eg: $grep anu studs

c) **rm**–deletes a file from the your directory of the file system syn: $rmfile name

d) **touch**–this command used to creates a blank file.

Syn: $touch file names

e) **cp**–copies the files or directories

syn: $cp source file destination file eg: $cp student studs

f) **mv** –to rename the file or directory syn: $mv old file new file

Eg: $mv–I student studentlist

(-iprompt when over write)

g) **cut**–this cuts or pickup a given number of character or the fields of the file.

Syn: $cut<option><filename>Eg: $cut –cfilename

$cut–c1-10sss

$cut–f 3,6sss

$cut–f 3-sss

-c cutting columns

-f cutting fields

h) **head**–displays 10 lines from the head(top) in a given file syn: $head file name

eg: $head sss

To display the top of two lines:

$head-2sss

1. **tail**– it print last 10 lines of the file

syn: $tail filename

eg: $tail student

To display the bottom two lines;

$tail-2student

j) **chmod**–used to change the permissions of a file or directory.

Syn: $chmod category operation permission file

Where, category–is the user type

Operation–is used to assign or remove permission

Permission–is the type of permission

File–are used to assign or remove permission

|  |  |  |
| --- | --- | --- |
| **Category** | **Operation** | **Permission** |
| u– users  g–group  o– others  a-all | +assign  -remove  =assign absolutely | r– read  w– write  x-execute |

Examples:

$chmodu-wxstudent

Removes write and execute permission for users

$chmodu+rw,g+rwstudent

Assigns read and write permission for users and groups

$chmodg=rwxstudent

Assigns absolute permission for groups of al lread,write and execute permissions

k) **wc**–it counts the number of lines, words,character in a specified file(s)

with the options as–l,-w,-csyn: $wc–lfilename

$wc–wfilename

$wc–cfilename

l) Pr- It is used to display the contents of the file by separating them into pages

and each page begins with the header for information.

pr [options] <file name >

$ pr devi

m) Paste

It concatenates the line from aeach input of file in column by column witha tab characters in between them.

paste [options] <file name >

$ paste f1 f2

p) Uniq

It compares adjacent lines of the file and displays the output by the eliminating of duplicate adjacent lines .

uniq [options] <file name >

$ uniq filename

**Result**:

Thus the file manipulating commands were executed successfully

**EX: NO:1C DIRECTORYCOMMANDS**

a) **mkdir**–used to creating a directory.

Syn: $mkdir directory name

Eg: $ mkdir classes

b) **rmdir**–it is used for deleting a directories.

Syn: $rmdir directory name

Eg: $rmdir classes

c) **cd**–changd to the current directory of the shell.

Syn: $cd~

(changes the path to your home directory)

$cd..

(changes to your parent directory)

$cd

d) **pwd**–(Present Working Directory)it displays the current directory.

Syn: $pwd

**Result**:

Thus the directory commands were executed successfully

**EX:NO:1D PROCESSCOMMANDS**

1. **exit**–terminates the process

syn: $ exit

1. **kill**–terminates or send a signal to the process

syn: $kill

1. kill <PID>

$ kill 10

2. **kill $!**

$! is the system variable which is used to stores the process of id the last background job. $! is used to kill the last process.

kill $!

$ kill $!

c) at

It is used to execute the process at the time specified.

echo <time>

$ at 14:08 (or)$ at 3 PM (or) $ at 4 :50 AM

**d) passwd**–create or change a password

syn: $ passwd

**e) telnet**–connect to remote machine using the telnet protocol

syn: $telnet

**f) ps**

It is used to display the attributes of a process.

ps

$ ps

$ ps -f ( Display the ancestry of a process )

$ ps -u ( Display the activities of a user )

$ ps -a ( Lists processes of all users but not the system processes )

**g) nohup**

It permits the execution of the process even after the user has logged out.

nohup <command>

$ nohup sort emp.txt ( result is available on nohup.out )

**Result**:

Thus the process commands were executed successfully.

**EX:NO: 1E GROUPINGCOMMANDS**

1. The semicolon(**;**)- used to execute more than one command at a time

eg: $who; date ; ls

b) The**&&**operator–signifies the logical AND operation. It means that only if first command is successfully executed,then the nest command will be executed.

Eg: $ls marks &&date

c) The**||**operator–signifies the logical OR operation. It means the first command will happen to be unsuccessful, it will continue to execute next command.

Eg: $ls marks ||date

**Result**:

Thus the grouping commands were executed successfully

**EX:NO 2A SHELL PROGRAMMING - SIMPLE**

**SWAPPING OF TWO NUMBERS**

**AIM:**

To write the shell programming for swapping of two numbers

**ALGORITHM:**

1. Get the user values for *a* and *b*
2. Interchange the values of *a* and *b* using another variable *temp* as follows:

*temp*= *a*

*a* = *b*

*b* = *temp*

1. Display the values of *a* and *b*

***PROGRAM***

first=5

second=10

temp=$first

first=$second

second=$temp

echo"After swapping, numbers are:"

echo"first = $first, second = $second"

**Output:**

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

After swapping, numbers are:

first = 10, second = 5

**AREA AND CIRCUMFERENCE OF CIRCLE**

**AIM:**

To write the shell Program for finding the area and circumference of a circle

**ALGORITHM:**

1. Read the value of *radius*
2. Calculate the *area* using the formulae: *pi* × *radius2*
3. Calculate the *circumference* using formulae: *2* × *pi* × *radius*
4. Print the *area* and *circumference of circle*

**PROGRAM**:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

echo "Enter the radious of the circle"

read r

area=$(echo "3.14\*$r\*$r" |)

circum=$(echo "3.14\*2\*$r" )

echo "area of the circle is " $area

echo "circumference of the circle is    " $circum

OUTPUT:

[@localhost ~]$ sh circlearea.sh

Enter the radius of the circle

3.5

area of the circle is            38.46

circumference of the circle is   21.98

**SIMPLE INTEREST**

**AIM**:

To write the Simple interest program using Shell Programming

**ALGORITHM:**

1. Read the values principal amount, rate of interest and years
2. Compute simple interest using the formulae: p \* n \* r / 100
3. Print the simple interest

**PROGRAM:**

echo " Enter the principle value: "  
read p  
echo " Enter the rate of interest:"  
read r  
echo " Enter the time period:"  
read t  
s=`expr $p \\* $t \\* $r / 100`  
echo " The simple interest is "  
echo $s

**OUTPUT:**  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[@localhost ~]$

Enter the principle value:  
2000  
Enter the rate of interest:  
4  
Enter the time period:  
10  
The simple interest is  
800

**TEMPERATURE CONVERSION**

**AIM:**

To write the shell Program for finding the Temperature Conversion

**ALGORITHM:**

1. Read the Fahrenheit Temperature
2. Compute the conversion from fahrenheit to centigrade using ( 5 / 9 ) \* ( $f - 32 )
3. Print the Centigrade temperature
4. Read the Centigrade temperature and convert it to Fahrenheit using (( 9 / 5 ) \* $c ) + 32
5. Print the result

PROGRAM:  
echo "Enter degree celsius temperature: "

readcelsius

$celsius\*1.8 + 32"

echo"$celsius degree celsius is equal to $fahrenheit degree fahrenheit"

OUTPUT:

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

Enter degree celsius temperature: 37

37 degree celsius is equal to 98.6 degree fahrenheit

**Result:**

Thus all the above simple Shell Programs has been written e and executed successfully.

**EX:NO:2B SHELL PROGRAMMING CONDITIONAL STATEMENT**

**SHELL PROGRAMMING CONDITIONAL STATEMENT**

**FINDING THE NUMBER IS ODD OR EVEN**

**AIM:**

To Find the given number whether is odd or even using shell programming

**ALGORITHM:**

1. Read the number
2. If the given number is divisible by 2

Print "Number is Even"

Else

Print "Number is Odd"

1. Display the result

**Program:**

echo "odd or even using shell programming "

echo "Enter a number"

read n

echo "RESULT: "

if [ `expr $n % 2` == 0 ]

then

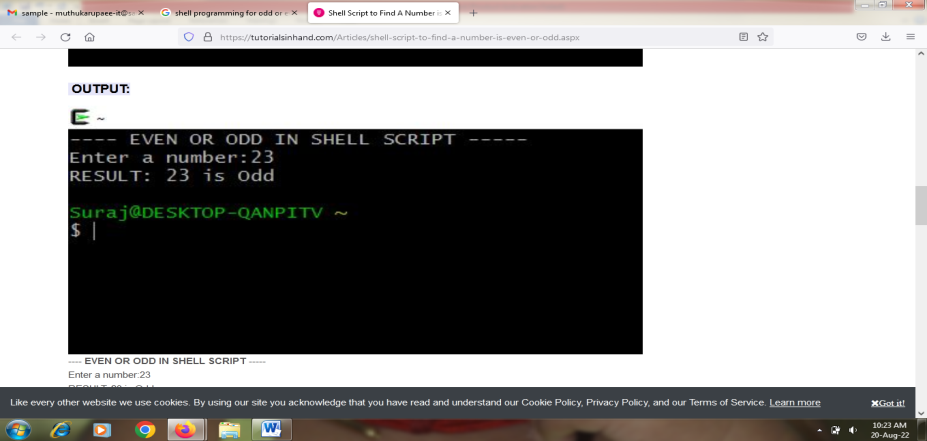
echo "$n is even"

else

echo "$n is Odd"

fi

**OUTPUT:**



**BIGGEST OF THREE NUMBERS**

**AIM:**

To write the shell program for the biggest of three numbers

**ALGORITHM:**

1. Get values of *a*, *b* and *c*
2. If *a > b* and *a > c* then

Print "A is the biggest number"

else if *b*>*c* then

Print "B is the biggest number "

else

Print "C is the biggest number "

1. Display the result

**Program:**

echo " THE GREATEST AMONG THREE NUMBER"

echo "Enter first number:"

read first\_num

echo "Enter second number:"

read second\_num

echo "Enter third number:"

read third\_num

if test $first\_num -gt $second\_num && test $first\_num -gt $third\_num

then

echo $first\_num is the greatest number.

elif test $second\_num -gt $third\_num

then

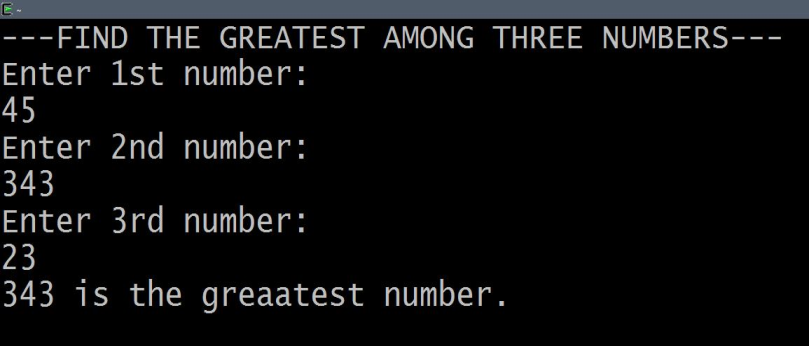
echo $second\_num is the greaatest number.

else

echo $third\_num is the greatest number.

Fi

**OUTPUT:**



**LEAP YEAR OR NOT**

**AIM:**

To Find the given year is leap or not using shell programming

**ALGORITHM:**

1. Read the value as year
2. Check the condition using the given year is divisible by 4 and equal to zero or not

(year % 4 –eq 0)

1. Print the year is leap year otherwise not a leap year

**PROGRAM:**echo "Enter year): "

read y

a = 'expr $y%4'

b = 'expr $y%100'

c = 'expr $y%400'

if[$a -eq 0 -a $b -ne - -o $c -eq 0]

then

echo "$y is leap year"

else

echo "$y is not a leap year"

fi

OUTPUT:

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

Input: 2024

Output: "2024 is leap year"

Input: 2018

Output: "2018 is not leap year"

**Result:** Thus the conditional statement programs using Shell Programming has been written and executed successfully.

**EX:NO:2c SHELL PROGRAMMING LOOPING STATEMENTS**

**SHELL PROGRAMMING – LOOPING STATEMENTS**

**FACTORIAL OF A NUMBER**

**AIM:**

To write the shell program for factorial of a given number

**ALGORITHM:**

1. Read the number n
2. Initialize the value fact=1
3. Repeat step 4 through 6 until i=n
4. Compute fact=fact\*i
5. Increment i=i+1
6. Print fact the value

**PROGRAM:**

echo"Enter a number"

read num

fact=1

**while** [ $num -gt 1 ]

**do**

fact=**$((**fact \* num**))**

num=**$((**num - **1))**

**done**

echo $fact

## Output

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

Enter a number

3

**6**

Enter a number

4

**24**

Enter a number

5

**120**

**FIBONACCI SERIES**

**AIM:**

To write the shell program for Fibonacci series

**ALGORITHM:**

1. Get the number of terms as *n*
2. Assign 0 to *a*, 1 to *b* and 3 to *i*
3. Print initial terms *a and b*
4. Generate the next term using the formula c= a+ b
5. Display the value of c
6. Increment *i* by 1
7. Assign the value b to a
8. Assign value c to b
9. Continue the steps 5–9 until i is less than equal to n

**Program:**

echo "Program to display the Fibonacci Series"

echo "How many number of terms to be generate "

read n

x=0

y=1

i=2

echo "Fibonacci Series up to $n terms :"

echo "$x"

echo "$y"

while [ $i -lt $n ]

do

i=`expr $i + 1 `

z=`expr $x + $y `

echo "$z"

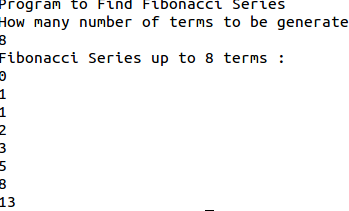
x=$y

y=$z

done

**OUTPUT:**

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



**REVERSE OF THE NUMBER**

**AIM:**

To write the shell program for reverse of a given number

**ALGORITHM:**

1. Get the number as n
2. Assign 0 to reverse variable
3. Using while loop check the given number is not equal to 0
4. Extract lastdigit by computing number modulo 10
5. Compute reverse = reverse10 + lastdigit
6. Divide number by 10
7. Repeat steps 4–6 until number> 0
8. Print the reverse

**Program**

echo enter n

read n

num=0

while [ $n -gt0 ]

do

num=$(expr $num \\* 10)

k=$(expr $n % 10)

num=$(expr $num + $k)

n=$(expr $n / 10)

done

echo number is $num

**OUTPUT:**

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

$ enter n

$456

$number is 654

**ARMSTRONG NUMBER**

**AIM:**

To write the shell program for Armstrong of a number or not

**ALGORITHM:**

1. Read the number as n
2. Initialize 0 to sum and n to temp
3. Using while loop check the given number is not equal to 0
4. Extract lastdigit by computing number modulo 10
5. Cube the lastdigit and add it to sum
6. Divide number by 10
7. Repeat steps 4–6 until number> 0
8. If sum = number then

Print “Armstrong number”

else

Print “Not an Armstrong number”

**Program**

echo"Enter a number: "

read c

x=$c

sum=0

r=0

n=0

while [ $x -gt0 ]

do

r=`expr $x % 10`

n=`expr $r \\* $r \\* $r`

sum=`expr $sum + $n`

x=`expr $x / 10`

done

if [ $sum -eq$c ]

then

echo"It is an Armstrong Number."

else

echo"It is not an Armstrong Number."

fi

**Output:**

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

Enter a number 153

It is an Armstrong Number

**SUM OF DIGITS OF A NUMBER**

**AIM:**

To find the sum of individual digits for a given number

**ALGORITHM:**

1. Get the value from the user and assign to n
2. Assign the value as sum=0
3. if n!=0 goto Step 6 else goto step 7
4. Assign n%10 value in p
5. Add p value to s
6. Assign n/10 value to n
7. Goto Step 5
8. print the result

**Program**

echo enter n

read n

sum=0

while [ $n -gt0 ]

do

r=$(expr $n % 10)

sum=$(expr $sum + $r)

n=$(expr $n / 10)

done

echo sum is $sum

**OUTPUT:**

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

$ enter n

$456

$sum is 15

**Result:** Thus the looping statement programs using Shell Programming has been written and executed successfully.

**EX:NO:3A FORK SYSTEM CALL**

**AIM:**

To implement the fork system call using C programming.

**ALGORITHM:**

1. Start the Algorithm
2. Include the Header files
3. Create a new child process using fork () system call.
4. If return value of fork is equal to -1 then display an error message.
5. If return value of fork is equal to 0 then display it as child process and print the child id and Parent id using getpid () and getppid() system call.
6. If return value of fork is not equal to 0 and display it all parent process and print the parent id using getpid () system call.
7. Print the corresponding result.

#include<stdio.h>

#include<unistd.h>

void main(void)

{

int childpid;

childpid=fork();

printf("\n%d",childpid);

if(childpid== -1)

{

printf("\n Can't fork.\n");

exit(0);

}

else if(childpid == 0)

{ /\* Child process \*/

printf("\n Child: Child pid = %d, Parent pid = %d \n", getpid(), getppid());

exit(0);

}

else

{ /\* Parent Process \*/

printf("\n Parent: Child pid = %d, Parent pid = %d \n", childpid, getpid());

printf("Hai\n");

exit(0);

}}

OUTPUT:

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

14983

Parent: Child pid = 14983, Parent pid = 14982

Child: Child pid = 14983, Parent pid = 1

**EX:NO:3B EXECUTE AND WAIT SYSTEM CALL**

**AIM:**

To implement the execute system call using C programming.

**ALGORITHM:**

1. Start the Algorithm.
2. Get the command from the user
3. Check the process id values is equal to zero after creating a child process using Fork()system call
4. It is equal to print the command for using exec()system call

execl(“/bin/date”,”date”,0);

1. Else print the file is not in existence..
2. The parent waits for the child process to complete with the wait() system call

cid=wait(&status);

1. When the child process completes, the parent process resumes from the call to wait where it completes.

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main()

{

pid\_t id,cid;

int status;

printf("Here comes the date\n");

if((id=fork())==0)

{

printf("pid is %d\n and id is %d\n:",getpid(),status);

execl("/bin/date","date",0);

}

cid=wait(&status);

printf("This was the date:");

printf("wid = %d and status =%d\n",cid,status);

}

**OUTPUT**:

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

Here comes the date

pid is 15538

and id is 21957

Sat Aug 20 05:51:06 UTC 2022

This was the date:wid = 15538 and status =0

**EX:NO:3C STAT SYSTEM CALL**

**AIM:**

To implement the STAT system call using C programming.

**ALGORITHM:**

1. Start the program.
2. Include the Header files.
3. Create the stat Structure with an object.
4. In main() function, passing the command line arguments as

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

1. Call the stat () system call to know about the information of the file

stat(argv[1],&nfile);

1. 6.Assign the stat function call to an integer flag variable

flag = stat(argv[1],&nfile);

1. By using an if condition check the (argc!=2 ) then print File name is not given
2. Else if Check the condition as (flag==-1) then print File name does not exists.
3. Else File exists then print the status of a file.
4. Display the file information by accessing the structure variable like size of the file, owner, block size, access time with an object
5. Print the attributes of the file.
6. Stop the execution.

#include<sys/types.h>

#include<sys/stat.h>

#include<time.h>

struct stat nfile;

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

{

stat (argv[1],&nfile);

int flag;

flag=stat(argv[1],&nfile);

if(argc!=2)

{

printf("File name not given:");

}

else if(flag==-1)

{

}

else

{

printf("File exists and filename are given\n\n");

printf("The information about the file %s\n\n",argv[1]);

printf("%s has %d link\n",argv[1],nfile.st\_nlink);

printf("%s has %d devices\n",argv[1],nfile.st\_dev);

printf("%s has %d inodes\n",argv[1],nfile.st\_ino);

printf("%s has %d protection\n",argv[1],nfile.st\_mode);

printf("%s has %d inode devices\n",argv[1],nfile.st\_rdev);

printf("%s has %d size\n",argv[1],nfile.st\_size);

printf("%s has %d owner\n",argv[1],nfile.st\_gid);

printf("%s has %d block size\n",argv[1],nfile.st\_blocks);

printf("%s has %d time\n",argv[1],nfile.st\_atime);

printf("%s has %d time\n",argv[1],nfile.st\_mtime);

printf("%s has %d inodes\n",argv[1],nfile.st\_dev);

}

}

OUTPUT:

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

[@telnet ~]$ cc first.c hello

File exists and filename are given

The information about the file hello

welcome

**EX:NO:3D OPENDIR CLOSEDIR SYSTEM CALL**

**AIM:**

To implement the OPENDIR & CLOSEDIR system call using C programming

**ALGORITHM:**

1. Start the program.
2. Include the Header files.
3. Create the dirent Structure with an object.
4. In main() function, passing the command line arguments as

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

5. Get the directory name from the user

6. using opendir() check the directory name is exist or not.If it exists read the files in the

directory

**dirp= opendir(buff)**

**dptr=readdir(dirp)**

7. Close the directory name

#include<stdio.h>

#include<dirent.h>

struct dirent \*dptr;

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

{

char buff[100];

DIR \*dirp;

printf(“\n\n ENTER DIRECTORY NAME”);

scanf(“%s”, buff);

if((dirp=opendir(buff))==NULL)

{

printf(“The given directory does not exist”);

exit(1);

}

while(dptr=readdir(dirp))

{

printf(“%s\n”,dptr->d\_name);

}

closedir(dirp);

}

OUTPUT:

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

[@telnet ~]$ cc dir1.c

ENTER DIRECTORY NAMEhello

The given directory does not exist

**EX:NO:4A SIMULATION OF COPY COMMAND**

**AIM:**

To write a program to simulate the COPY Command

**ALGORITHM:**

1. Start the program.
2. Include the Header files.
3. Declare the two character arrays

char src[50],dest[50];

1. In main() function, passing the command line arguments as

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

1. If no files are given then it will display an error message
2. Else copy the file using string function

strcpy(src,argv[1]);

strcpy(dest,argv[2]);

1. Compare the two character arrays .If it is not equal to zero then
   1. Create the link between source and destination file is not equal to -1then print “Copying File succeeded”
   2. Else Display “Error in copying the file”
2. Otherwise display an error message.

#include<stdio.h>

#include<unistd.h>

#include<string.h>

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

{

char src[50],dest[50];

if(argc<=2)

{

printf("\nCopying a file\n");

printf("copy[src],[desc]");

exit(0);

}

else

{

strcpy(src,argv[1]);

strcpy(dest,argv[2]);

}

if(strcmp(src,dest)!=0)

{

if(link(src,dest)!=-1)

{

printf("\n\nCopy the file successfully”);

}

else

{

printf("\n\n error in copying the file\n\n");

}

}

else

{

printf("\n\nSource and destination are not same");

}

}

OUTPUT:

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

[@telnet ~]$ cc copy1.c hello.c hai.text

Copy the file successfully

**EX:NO:4B SIMULATION OF LS COMMAND**

**AIM:**

To write a program to simulate the LS Command

**ALGORITHM:**

1. Start the program.
2. Include the Header files.
3. Declare a structure pointer

struct dirent \*\*namelist;

1. In main() function, passing the command line arguments as

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

1. If all the files is to be listed ,then using the scandir command, a “.” is placed and stored in an alphabetical order using alphasort.
2. If any directory name is specified it will list the files in the directory using scandir command
3. Check if the condition n<0 Print error message.
4. Else copy all the files into structure using pointer variable
5. Free the namelist and display the total number of files.

#include<dirent.h>

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

{

struct dirent \*\*namelist;

int i=0,n;

if(argc<=1)

{

n=scandir(".",&namelist,0,alphasort);

}

else

{

n=scandir(argv[1],&namelist,0,alphasort);

}

if (n<0)

{

printf("\nNo files or directory\n\n");

}

else

{

while (i<n)

{

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

i++;

}

free(namelist);

printf("\n total files %d\n",n);

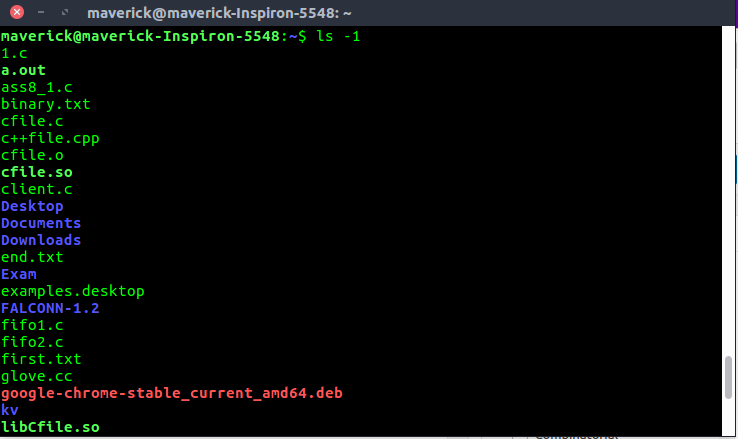
}

}

**OUTPUT:**

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

[@telnet ~]$ cc lsfile.c

****

**Total files:20**

**EX:NO:4C SIMULATION OF GREP COMMAND**

**AIM:**

To write a program to simulate the GREP Command

**ALGORITHM:**

1. Start the program.
2. Include the Header files.
3. Declare the two character arrays
4. In main() function, passing the command line arguments as

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

1. If the argument count is less than 2 then print no pattern and file name are given then it will display an error message with correct syntax
2. Else copy the first argument to the pattern array and second argument to the another array using string function

strcpy(pattern,argv[1]);

strcpy(fname,argv[2]);

1. File is opened in read mode and using getc() read the contents character by character till EOF
2. The pattern to be searched is found and stored in an array.
3. Compare the array and pattern using strstr command
4. If it is same, print the total number of times occurred pattern in a given file else print an error message
5. Close the file descriptor.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

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

{

char pattern[20],fname[25];

FILE \*fp;

char ch=0,temp[100];

int cnt=0,i=0,ln=0;

if(argc<=2)

{

printf("error:too many arguments");

printf("\nsyntax:./grep<pattern><filename>\n");

exit(0);

}

else

{

strcpy(pattern,argv[1]);

strcpy(fname,argv[2]);

}

fp=fopen(fname,"r");

if(fp!=0)

{

while(fgetc(fp)!=EOF)

{

fseek(fp,-1,SEEK\_CUR);

while((ch=fgetc(fp))!='\n')

{

temp[i]=ch;

i++;

}

ln++;

temp[i++]='\0';

i=0;

if(strstr(temp,pattern)!=NULL)

{

printf("%d\t %s\n",ln,temp);

cnt++;

}

}

fclose(fp);

printf("\n%s has occured %d time(s) in files %s\n",pattern,cnt,fname);

}

else

printf("File name not Found");

}

**OUTPUT:**

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

[@telnet ~]$ cc greep1.c wel hello.txt

Wel has occured 3 time(s) in files hello.txt

**CPU SCHEDULING ALGORITHMS**

**EX:NO:5A FCFS SCHEDULING ALGORITHM**

**Aim:**

To write a C program in UNIX environment to implement the First Come First Serve scheduling

**Algorithm:**

Step 1: Start the process

Step 2: Get the number of processes in the ready Queue

Step 3: For each process in the ready Q, intiate the process id and accept the CPU burst time

Step 4: Assign and Set the waiting of the first process as ‘0’ and its burst time to turn around time

Step 5: Every each process in the Ready Q calculate

(a) Waiting time for process(n)= wt (n-1) + bt(n-1)

(b) Turn around time for Process(n)= wt+ bt(n)

Step 6: Calculate

(a) Average waiting time = twt / nop

(b) Average Turnaround time = ttt / nop

Step 7: Stop the process

**PROGRAM:**

#include<stdio.h>

int main()

{

int n,b[20],wt[20],tut[20],avwt=0,avtat=0,i,j;

printf("Enter total number of processes:");

scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

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

{

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

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

}

wt[0]=0; //waiting time for first process is 0

//calculating waiting time

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

{

wt[i]=0;

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

wt[i]+=b[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

//calculating turnaround time

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

{

tat[i]=b[i]+wt[i];

avwt+=wt[i];

avtat+=tut[i];

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

}

avwt/=i;

avtat/=i;

printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

**OUTPUT:**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter total number of processes:5

Enter Process Burst Time

P1:50

2P2:3

P3:2

P4:45

P5:2

Process Burst Time Waiting Time Turnaround Time

P[1] 50 0 50

P[2] 3 50 53

P[3] 2 53 55

P[4] 45 55 100

P[5] 2 100 102

Average Waiting Time:51

Average Turnaround Time:72

**RESULT**

**Thus the FCFS scheduling program has been written and executed successfully**

**EX:NO:5A Implementation of SJF (Non Preemption) Scheduling Algorithm**

**AIM:**

To implement the Shortest job first(Non Preemption) scheduling program with arrival time using C

**ALGORITHM:**

Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time and assign the arrival time for each process

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as ‘0’ and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

1. Waiting time for process(n) as
2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

**PROGRAM:**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("nEnter Burst Time:n");

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

{

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

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

p[i]=i+1;

}

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcesst Burst Time \tWaiting Time\tTurnaround Time");

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

{

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

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

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

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

}

OUTPUT:

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

Enter number of process:5

nEnter Burst Time:np1:5

p2:10

p3:2

p4:5

p5:1

Processt Burst Time Waiting Time Turnaround Time

p5 1 0 1

p3 2 1 3

p4 5 3 8

p1 5 8 13

p2 10 13 23

Average Waiting Time=5.000000

Average Turnaround Time=9.600000

**RESULT**

Thus the SJF(Non pre-emption) scheduling program has been written and executed successfully

**Ex.No : 5C** **IMPLEMENTATION OF PRIORITY SCHEDULING ALGORITHM**

**Aim:**

To write a C program in UNIX environment to implement the Priority Scheduling.

**Algorithm:**

1. Get the number of Processes.

2. Get also the CPU time and Process timeand the priority for each process from the

user.

3. Sort the CPU time of the processes according to the process priority in ascending order.

4. Waiting time for 1st process is always zero

5. For each process the waiting time is equivalent to the CPU time of the previous process.

6. The ratio of waiting time of all the processes to the number process will give the average waitingtime.

7.Calculate the Turnaround time is sum of waiting time and burst time and will give the average turn around time

7. Display the output.

**PROGRAM:**

#include<stdio.h>

main()

{

int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;

float wtavg, tatavg;

printf("Enter the number of processes --- ");

scanf("%d",&n);

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

{

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d --- ",i);

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

}

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

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

if(pri[i] > pri[k])

{

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=pri[i];

pri[i]=pri[k];

pri[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

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

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME"); for(i=0;i<n;i++)

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

printf("\nAverage Waiting Time is --- %f",wtavg/n);

printf("\nAverage Turnaround Time is --- %f",tatavg/n);

}

**OUTPUT:**

**Enter the number of processes 3**

**Enter the Burst Time & Priority of Process 0 --- 23**

**Enter the Burst Time & Priority of Process 1 --- 23**

**Enter the Burst Time & Priority of Process 2 --- 61**

**PROCESS PRIORITY BURST TIME WAITING TIME TURNAROUNDTIME**

**2 1 6 0 6**

**1 3 2 6 8**

**0 3 2 8 10**

**Average Waiting Time is --- 4.666667**

**Average Turnaround Time is --- 8.000000**

**RESULT**

Thus the priority scheduling program has been written and executed successfully

**Ex.No : 5C IMPLEMENTATION OF ROUNDROBIN SCHEDULING ALGORITHM**

**Aim:**

To write a C program in UNIX environment to implement the Priority Scheduling.

Algorithm :

Step 1: Enter value of n where n is no. of process in the ready queue.

Step 2: Enter name and Burst time of process in the array nam[n] and bt[n] respectively. Take one more array to store the Starting time of the process i.e.st[n]

Initialize st[i]=bt[i].

Step 3:Enter time quantum for which each process get the CPU in variable tq.

Step 4:Take three counter variables as count=0 [Count the no of processes which have completed] and temp=0 [To store the time Quantam].

Step 5:Apply infinite loop. [Loop will break when (n==count)].

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

temp=tq

Check the value of st[i].

if(st[i]==0)

then count++ & continue the loop.

elseif the value of st[i]>tq

then st[i]= st[i] - tq

else if the value of st[i]>=0

then put the value of st[i] into temp & initialize st[i] to be 0.

Now , To calculate value of turnaround time take variable sqto be 0.

and sq= sq + temp. & tat[i]= sq.

Step 6: Waiting time for ith process is Turnaround time - Burst time

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

Step 7: Average turnaround time of n processes is avg/n.

Step 8: avg=0.0avg=avg+tr[i]

Step 9: Average waiting time of n processes is avg/n

avg=avg +wt[i]

Step 10:Print all the values.

Step 11:Exit.

PROGRAM:

#include<stdio.h>

intmain()

{

      inti, limit, total = 0, x, counter = 0, time\_quantum;

      intwait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

      floataverage\_wait\_time, average\_turnaround\_time;

      printf("nEnter Total Number of Processes:t");

      scanf("%d", &limit);

      x = limit;

      for(i = 0; i < limit; i++)

      {

            printf("nEnter Details of Process[%d]n", i + 1);

            printf("Arrival Time:t");

            scanf("%d", &arrival\_time[i]);

            printf("Burst Time:t");

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

            temp[i] = burst\_time[i];

      }

      printf("nEnter Time Quantum:t");

      scanf("%d", &time\_quantum);

      printf("nProcess IDttBurst Timet Turnaround Timet Waiting Timen");

      for(total = 0, i = 0; x != 0;)

      {

            if(temp[i] <= time\_quantum && temp[i] > 0)

            {

                  total = total + temp[i];

                  temp[i] = 0;

                  counter = 1;

            }

            elseif(temp[i] > 0)

            {

                  temp[i] = temp[i] - time\_quantum;

                  total = total + time\_quantum;

            }

            if(temp[i] == 0&& counter == 1)

            {

                  x--;

                  printf("nProcess[%d]tt%dtt %dttt %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

                  wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

                  turnaround\_time = turnaround\_time + total - arrival\_time[i];

                  counter = 0;

            }

            if(i == limit - 1)

            {

                  i = 0;

            }

            elseif(arrival\_time[i + 1] <= total)

            {

                  i++;

            }

            else

            {

                  i = 0;

            }

      }

      average\_wait\_time = wait\_time \* 1.0/ limit;

      average\_turnaround\_time = turnaround\_time \* 1.0/ limit;

      printf("nnAverage Waiting Time:t%f", average\_wait\_time);

      printf("nAvg Turnaround Time:t%fn", average\_turnaround\_time);

      return0;

}

OUTPUT:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Total Number of Processes:t3

Enter Details of Process[1]nArrival Time:2

Burst Time:25

Enter Details of Process[2]nArrival Time:1

Burst Time:5

nEnter Details of Process[3]nArrival Time:3

Burst Time:20

Enter Time Quantum:5

Process ID Burst Time Turnaround Time Waiting Timen

Process[2] 5 9 4

Process[3] 20 42 22

Process[1] 25 48 23

Average Waiting Time: 16.333334

Avg Turnaround Time: 33.000000

**RESULT**

Thus the roundrobin scheduling program has been written and executed successfully

**EX: No: 6 INTERPROCESS COMMUNICATION USING SHARED MEMORY**

**Aim:**

To write a c program to develop an application using Inter process Communication (IPC)

using Shared Memory.

**Algorithm:**

1. Create the shared memory for parent process using shmget()system call.

2. Now allow the parent process to write in shared memory using shmget pointer which is return

type of shmget()

3. Now across and attach the same shared memory to the child process

4. The data in the shared memory is read by the child process using the shmdt pointer

5. Now detach and reuse the shared memory.

SENDER:

#include<stdio.h>

#include<sys/shm.h>

#include<sys/ipc.h>

#define size 32

int main()

{

int shmid;

char \*s[100],\*str;

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

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

str=shmat(shmid,0,0);

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

gets(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**

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

ipc message passing using shared memory sender

eneter the message to be sent hai

hai

your mesage has been sent

ipc message passing using shared memory Reciver

received message is...

hai

**Result:**

**Thus the IPC program using shared memory has been written and executed successfully**

**EX:NO:7 IMPLEMENTATION OF SEMAPHORE**

**Aim**

To write a C program to implement producer consumer relationship using semaphore.

**Algorithm**

1. Initialize the semaphore variables mutex =1, full=0,empty=3

2. Invoke the wait() to enter into critical section and signal() is for exit section

2. The Producer will produce until buffer is full.

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

Producer produces the item

mutex=signal(mutex);

3. The Consumer will consume until the buffer is empty.

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

consumes item

x--;

mutex=signal(mutex);

4. If the Buffer is empty then the Consumer will not consume the items.

5. If the Buffer is full then the Producer will not consume the items.

6. Print the result.

**Program**

#include<stdio.h>

#include<stdlib.h>

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

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

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

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

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

x--;

mutex=signal(mutex);

OUTPUT:

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

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:2

Consumer consumes item 2

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

Enter your choice:

**Result :**

**Thus the implementation of producer consumer problem using semaphore has been executed successfully**

**Ex:NO:8 BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE**

AIM:

To implement Bankers Algorithm for Deadlock Avoidance using c

ALGORITHM:

1. Start the program.
2. Create a structure for all vectors
3. Get the number of resources and processes.
4. Get the avail value, allocation and max value for all processes based on each resources
5. Find the need value using max allocation
6. Check whether its possible to allocate.
7. If it is possible then the system is in safe state.
8. Else system is not in safety state.
9. If the new request comes then check that the system is in safety. or not if we allow the request.
10. stop the program.

PROGRAM

#include<stdio.h>

struct file

{

int all[10];

int max[10];

int need[10];

int flag;

};

void main()

{

struct file f[10];

int fl; int i, j, k, p, b, n, r, g, cnt=0, id, newr;

int avail[10],seq[10];

printf("Enter number of processes -- ");

scanf("%d",&n);

printf("Enter number of resources -- ");

scanf("%d",&r);

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

{

printf("Enter details for P%d",i);

printf("\nEnter allocation\t -- \t");

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

scanf("%d",&f[i].all[j]);

printf("Enter Max\t\t -- \t");

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

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

f[i].flag=0;

}

printf("\nEnter Available Resources\t -- \t");

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

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

printf("\nEnter New Request Details -- ");

printf("\nEnter pid \t -- \t");

scanf("%d",&id);

printf("Enter Request for Resources \t -- \t");

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

{

scanf("%d",&newr);

f[id].all[i] += newr;

avail[i]=avail[i] - newr;

}

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

{

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

{

{

f[i].need[j]=f[i].max[j]-f[i].all[j];

if(f[i].need[j]<0)

f[i].need[j]=0;

}

}

cnt=0;

fl=0;

while(cnt!=n)

{

g=0;

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

{

if(f[j].flag==0)

{

b=0;

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

{

if(avail[p]>=f[j].need[p])

b=b+1;

else

b=b-1;

}

if(b==r)

{

printf("\nP%d is visited",j);

seq[fl++]=j;

f[j].flag=1;

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

avail[k]=avail[k]+f[j].all[k];

cnt=cnt+1;

printf("(");

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

printf("%3d",avail[k]);

printf(")");

g=1;

}

}

}

if(g==0)

{

printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");

printf("\n SYSTEM IS IN UNSAFE STATE");

goto y;

}

}

printf("\nSYSTEM IS IN SAFE STATE");

printf("\nThe Safe Sequence is -- (");

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

printf("P%d ",seq[i]); printf(")");

y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");

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

{

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

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

printf("%6d",f[i].all[j]);

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

printf("%6d",f[i].max[j]);

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

printf("%6d",f[i].need[j]);

printf("\n");

}

}

Result:

Thus the bankers algorithm for deadlock avoidance has been written and executed successfully

**EX:NO:9 DEADLOCK DETECTION ALGORITHM**

**AIM:**

To implement the Bankers Algorithm for Deadlock Avoidance using c

**ALGORITHM:**

1. Enter each process that has a row in the Allocation matrix by zeros.

2. Assign a temporary vector W to equal the Available of the vector.

3. Find an index i such that the process i is currently unmarked and the row of Q

is less than or equal to W . .

4. If a row is found, mark process i and then add the corresponding row of the allocation matrixto W . Return back to step 3.

**PROGRAM:**

#include<stdio.h>

static int mark[20];

int i,j,np,nr;

int main()

{

int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");

scanf("%d",&np);

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

scanf("%d",&nr);

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

{

printf("\nTotal Amount of the Resource R%d: ",i+1);

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

}

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

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

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

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

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

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

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

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

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

{

avail[j]=r[j];

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

{

avail[j]-=alloc[i][j];

}

}

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

{

int count=0;

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

{

if(alloc[i][j]==0)

count++;

else

break;

}

if(count==nr)

mark[i]=1;

}

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

w[j]=avail[j];

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

{

int canbeprocessed=0;

if(mark[i]!=1)

{

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

{

if(request[i][j]<=w[j])

canbeprocessed=1;

else

{

canbeprocessed=0;

break;

}

}

if(canbeprocessed)

{

mark[i]=1;

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

w[j]+=alloc[i][j];

}

}

}

int deadlock=0;

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

if(mark[i]!=1)

deadlock=1;

if(deadlock)

printf("\n Deadlock detected");

else

printf("\n No Deadlock possible");

}

**OUTPUT:**

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

Enter the no of process: 4

Enter the no of resources: 5

Total Amount of the Resource R1: 2

Total Amount of the Resource R2: 1

Total Amount of the Resource R3: 1

Total Amount of the Resource R4: 2

Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0

1 1 0 0 0

0 0 0 1 0

0 0 0 0 0

Deadlock detected

**Result:**

Thus the deadlock detection algorithm has been written and executed successfully

**Ex.No:10 IMPLEMENTATION OF THREADING AND**

**SYNCHRONIZATION APPLICATIONS**

**AIM:**

To write a c program to implement Threading and Synchronization Applications.

**ALGORITHM:**

Step 1: Start the process

Step 2: Declare process thread, thread-id.

Step 3: Read the process thread and thread state.

Step 4: Check the process thread equals to thread-id by using if condition.

Step 5: Check the error state of the thread.

Step 6: Display the completed thread process.

Step 7: Stop the process

#include<stdio.h>

#include<string.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

pthread\_t tid[2];

void\* doSomeThing(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, &doSomeThing, 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**

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

[\*\*\*\*\*\*\*@telnet ~]$ cc -pthread -o thread1 thread1.c

[\*\*\*\*\*\*\*\*@telnet ~]$ ./thread1

Thread created successfully

Thread created successfully

First thread processing

Second thread processing \*/

**Ex.No:11a Dynamic Storage Allocation-First Fit**

**Aim:**

To write a ‘C’ program in UNIX to implement Dynamic Storage Allocation Strategy for First

Fit.

**Algorithm:**

1. Start

2. Read the number of free blocks and the size of each free block.

3. Get the process block size to be loaded.

4. Allocate the first hole that is big enough to load the process

5. If no hole is big enough to load the process, then process cannot be allocated.

6. Display the size of all the free blocks.

7. Stop.

Dynamic Storage Allocation-First Fit

#include<stdio.h>

//#include<process.h>

void main()

{

int a[20],p[20],i,j,n,m;

printf("Enter no of Blocks.\n");

scanf("%d",&n);

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

{

printf("Enter the %dst Block size:",i);

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

}

printf("Enter no of Process.\n");

scanf("%d",&m);

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

{

printf("Enter the size of %dst Process:",i);

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

}

printf("\nProcess\tBlockSize\n");

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

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

printf("\n\n");

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

{

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

{

if(p[j]<=a[i])

{

printf("The Process %d allocated to %d\n",j,a[i]);

p[j]=10000;

break;

}

}

}

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

{

if(p[j]!=10000)

{

printf("The Process %d is not allocated\n",j);

}

}

}

/\* OUTPUT

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

[@telnet ~]$ cc firstfit.c

[@telnet ~]$ ./a.out

Enter no of Blocks.

5

Enter the 0st Block size:100

Enter the 1st Block size:200

Enter the 2st Block size:300

Enter the 3st Block size:150

Enter the 4st Block size:250

Enter no of Process.

5

Enter the size of 0st Process:99

Enter the size of 1st Process:160

Enter the size of 2st Process:250

Enter the size of 3st Process:100

Enter the size of 4st Process:300

Process BlockSize

99 100

160 200

250 300

100 150

300 250

The Process 0 allocated to 100

The Process 1 allocated to 200

The Process 2 allocated to 300

The Process 3 allocated to 150

The Process 4 is not allocated

\*/

**Result**

Thus the dynamic storage allocation scheme using firstfit algorithm has been written andexecuted successfully

**Ex.No : 11bDynamic Storage Allocation-Best Fit**

**Aim:**

To write a ‘C’ program in UNIX to implement Dynamic Storage Allocation Strategy for Best Fit.

**Algorithm:**

1. Start

2. Read the number of free blocks and the size of each free block.

3. Get the process block size to be loaded.

4. Allocate the smallest hole that is big enough to load the process

5. If no hole is big enough to load the process, then process cannot be allocated.

6. Display the size of all the free blocks.

7. Stop.

**PROGRAM:**

#include<stdio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

static int bf[max],ff[max];

// clrscr();

printf("\n\tMemory Management Scheme - Best Fit");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++) {printf("Block %d:",i);scanf("%d",&b[i]);}

printf("Enter the size of the files :-\n");

for(i=1;i<=nf;i++) {printf("File %d:",i);scanf("%d",&f[i]);}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}

}

frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}

/\* OUTPUT

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

[@telnet ~]$ cc best11.c

[@telnet ~]$ ./a.out

Memory Management Scheme - Best Fit

Enter the number of blocks:5

Enter the number of files:4

Enter the size of the blocks:-

Block 1:10

Block 2:15

Block 3:5

Block 4:9

Block 5:3

Enter the size of the files :-

File 1:1

File 2:4

File 3:7

File 4:12

File\_no: File\_size : Block\_no: Block\_size: Fragement

1 1 5 3 2

2 4 3 5 1

3 7 4 9 2

4 12 2 15 3 \*/

**Ex.No : 11c Dynamic Storage Allocation-Worst Fit**

**Aim:**

To write a ‘C’ program in UNIX to implement Dynamic Storage Allocation Strategy for Worst

Fit.

**Algorithm:**

1. Start

2. Read the number of free blocks and the size of each free block.

3. Get the process block size to be loaded.

4. Allocate the largest hole that is big enough to load the process

5. If no hole is big enough to load the process, then process cannot be allocated.

6. Display the size of all the free blocks.

7. Stop.

#include<stdio.h>

int main(){

int p,m;

printf("Enter number of processes:");

scanf("%d",&p);

printf("Enter number of Memory blocks:");

scanf("%d",&m);

int pi[p];

struct mem{

int id;

int size;

}m1[m];

int i;

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

{

printf("Enter size of process %d:",i+1);

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

}

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

{

printf("Enter size of memory %d:",i+1);

scanf("%d",&m1[i].size);

m1[i].id=i+1;

}

int j;

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

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

if(m1[i].size<m1[j].size)

{

struct mem t=m1[i];

m1[i]=m1[j];

m1[j]=t;

}

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

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

if(m1[j].size>=pi[i]){

m1[j].size-=pi[i];

printf("Allocating process %d to memory %d\n Size remaining in it after allocation

%d\n\n",i+1,j+1,m1[j].size);

break;

}

}

if(j==m)

{printf("Not enough memory for process %d",i);break;}

}

}

**OUTPUT:**

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

Enter number of processes:3

Enter number of Memory blocks:3

Enter size of process 1:60

Enter size of process 2:20

Enter size of process 3:10

Enter size of memory 1:100

Enter size of memory 2:200

Enter size of memory 3:300

Allocating process 1 to memory 1

Size remaining in it after allocation 240

Allocating process 2 to memory 1

Size remaining in it after allocation 220

**EX:NO : 12 Implementation of Paging Technique of Memory Management**

**AIM:**

To write a c program to implement Paging technique for memory management.

**ALGORITHM:**

Step 1: Start the process

Step 2: Declare page number, page table, frame number and process size.

Step 3: Read the process size, total number of pages

Step 4: Read the relative address

Step 5: Calculate the physical address

Step 6: Display the address

Step 7: Stop the process

**PROGRAM:**

#include<stdio.h>

main()

{

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

int s[10], fno[10][20];

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

scanf("%d",&ms);

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

scanf("%d",&ps);

nop = ms/ps;

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

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

scanf("%d",&np);

rempages = nop;

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

{

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

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

if(s[i] >rempages)

{

printf("\nMemory is Full");

break;

}

rempages = rempages - s[i];

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

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

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

}

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

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

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

if(x>np || y>=s[i] || offset>=ps)

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

else

{

pa=fno[x][y]\*ps+offset;

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

}

}

/\* OUTPUT \*/

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

/\*[\*\*\*\*\*\*\*\*\*\*@telnet ~]$ ./a.out

Enter the memory size -- 1000

Enter the page size -- 100

The no. of pages available in memory are -- 10

Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 4 5 7 3

1

Enter no. of pages required for p[3]-- 5

Memory is Full

Enter Logical Address to find Physical Address

Enter process no. and pagenumber and offset -- 2 3 60

The Physical Address is -- 360 \*/

**Ex.No:13.a PAGE REPLACEMENT ALGORITHMS**

**FIFO**

**AIM:**

To write a C program for implementation of FIFO page replacement algorithm.

**ALGORITHM:**

Step 1: Intiliaze the program.

Step 2: Declare the all variables.

Step 3: Get the number of frames.

Step 4: Get the reference string ends with zero.

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

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

Step 7: Assign all the three frames are empty.

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

Step 9: Display the total number of page to faults.

Step 10: Close.

#include<stdio.h>

main()

{

int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

printf("\n Enter the length of refer- string ");

scanf("%d",&n);

printf("\n Enter the refer- string ");

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

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

printf("\n Enter no of frames ");

scanf("%d",&f);

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

m[i]=-1;

printf("\n The Page Replacement Process is \n");

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

{

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

{

if(m[k]==rs[i])

break;

}

if(k==f)

{

m[count++]=rs[i];

pf++;

}

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

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

if(k==f)

printf("\tPF No. %d",pf);

printf("\n");

if(count==f)

count=0;

}

printf("\n The number of Page Faults FIFO %d",pf); }

/\* OUTPUT

[\*\*\*\*\*\*\*\*\*\*\*e@telnet ~]$ ./a.out

Enter the length of reference string -- 20

Enter the reference string -- 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

The Page Replacement Process is --

7 -1 -1 PF No. 1

7 0 -1 PF No. 2

7 0 1 PF No. 3

2 0 1 PF No. 4

2 0 1

2 3 1 PF No. 5

2 3 0 PF No. 6

4 3 0 PF No. 7

4 2 0 PF No. 8

4 2 3 PF No. 9

0 2 3 PF No. 10

0 2 3

0 2 3

0 1 3 PF No. 11

0 1 2 PF No. 12

0 1 2

0 1 2

7 1 2 PF No. 13

7 0 2 PF No. 14

7 0 1 PF No. 15

The number of Page Faults using FIFO are 15 \*/

**Ex.No:13.b PAGE REPLACEMENT ALGORITHMS**

**LRU**

**AIM:**

To write a c program to implement LRU page replacement algorithm

**ALGORITHM:**

Step 1: Assign the process

Step 2: Initialize the page size

Step 3: From the user get the number of pages which its to be inserted

Step 4: Get the value

Step 5: Assign 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 all values

Step 9: close

#include<stdio.h>

void main()

{

int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;

printf("\nEnter number of page reference ");

scanf("%d",&m);

printf("\nEnter the reference string ");

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

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

printf("\nEnter the available number of frames ");

scanf("%d",&f);

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

{

cntr[i]=0;

a[i]=-1;

}

printf("\n Page Replacement Process is \n");

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

{

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

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

{

cntr[j]++;

break;

}

if(j==f)

{

min = 0;

for(k=1;k<f;k++)

if(cntr[k]<cntr[min])

min=k;

a[min]=rs[i];

cntr[min]=1;

pf++;

}

printf("\n");

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

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

if(j==f)

printf("\tPF No. %d",pf);

}

printf("\n\n Total number of page faults %d",pf);

}

/\* OUTPUT

[\*\*\*\*\*\*\*\*\*@telnet ~]$ ./a.out

Enter number of page references 10

Enter the reference string 1

2

3

4

5

2

5

1

4

3

Enter the available no. of frames 3

The Page Replacement Process is

1 -1 -1 PF No. 1

1 2 -1 PF No. 2

1 2 3 PF No. 3

4 2 3 PF No. 4

5 2 3 PF No. 5

5 2 3 PF No. 5

5 2 3 PF No. 5

5 2 1 PF No. 6

5 2 4 PF No. 7

5 2 3 PF No. 8

Total number of page faults -- 8 \*/

**Ex.No:13.c PAGE REPLACEMENT ALGORITHMS**

**LFU**

**Aim:**

To write C program to implement LFU 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 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>

main()

{

int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

printf("Enter the length of reference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

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

{

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

flag[i]=0;

}

printf("Enter the number of frames -- ");

scanf("%d",&f);

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

{

count[i]=0;

m[i]=-1;

}

printf("\nThe Page Replacement process is -- \n");

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

{

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

{

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

{

flag[i]=1;

count[j]=next;

next++;

}

}

if(flag[i]==0)

{

if(i<f)

{

m[i]=rs[i];

count[i]=next;

next++;

}

else

{

min=0;

for(j=1;j<f;j++)

if(count[min] > count[j])

min=j;

m[min]=rs[i];

count[min]=next;

next++;

}

pf++;

}

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

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

if(flag[i]==0)

printf("PF No. -- %d" , pf);

printf("\n");

}

printf("\nThe number of page faults using LRU are %d",pf);

}

/\* OUTPUT

[\*\*\*\*\*\*\*\*\*@telnet ~]$ ./a.out

Enter the length of reference string -- 20

Enter the reference string -- 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

The Page Replacement process is --

7 -1 -1 PF No. -- 1

7 0 -1 PF No. -- 2

7 0 1 PF No. -- 3

2 0 1 PF No. -- 4

2 0 1

2 0 3 PF No. -- 5

2 0 3

4 0 3 PF No. -- 6

4 0 2 PF No. -- 7

4 3 2 PF No. -- 8

0 3 2 PF No. -- 9

0 3 2

0 3 2

1 3 2 PF No. -- 10

1 3 2

1 0 2 PF No. -- 11

1 0 2

1 0 7 PF No. -- 12

1 0 7

1 0 7

The number of page faults using LRU are 12 \*/

**EX:NO:14 Implementation of the various File Organization Techniques**

**AIM:**

To initialise of the various File Organization Techniques

**ALGORITHM:**

Step 1: Initiate the program.

Step 2: from the user get the directory name.

Step 3: Get the user choice for filetechniques

Step 4: If user choice 1 for create file in specify directory

Step 5: It user choice 2 for delete file in specify directory

Step 6:It user choice 3.Get the file name to be searched. If it is found print the results filename, length , Blocks allocated otherwise File not found

Step 7:If user choice 4 display the all files on the present directory

Step 6: Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

dir.fcnt = 0;

printf("\nEnter name of directory -- ");

scanf("%s", dir.dname);

while(1)

{

printf("\n\n1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5. Exit\n");

printf("Enter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the name of the file -- ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\nEnter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is deleted ",f);

strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;

}

}

if(i==dir.fcnt)

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

else

dir.fcnt--;

break;

case 3:

printf("\nEnter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is found ", f);

break;

}

}

if(i==dir.fcnt)

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

break;

case 4:

if(dir.fcnt==0)

printf("\nDirectory Empty");

else

{

printf("\nThe Files are -- ");

for(i=0;i<dir.fcnt;i++)

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

}

break;

default: exit(0);

}

}

}

[@telnet ~]$ cc fileteq.c

[@telnet ~]$ ./a.out

Enter name of directory -- os

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 1

Enter the name of the file -- hello

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 4

The Files are -- hell

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 3

Enter the name of the file -- h

File h not found

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 1

Enter the name of the file -- welcome

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 4

The Files are -- hello welcome

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 2

Enter the name of the file -- hello

File hello is deleted

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 4

The Files are -- welcome

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice -- 5

[@telnet ~]$

**EX:NO:15A Implementation of the following File Allocation Strategies - Sequential**

**AIM:**

To implement sequential file allocation technique.

**ALGORITHM:**

Step 1: Intiate the program.

Step 2: From the user get the no. of files.

Step 3: Assign the memory requirement of each file with start block and length of a file

Step 4: Assign the required locations to each in sequential order.

Step 5: From the user get which the file name to be searched. If file found print the results filename, length , Blocks allocated otherwise File not found

Step 6: Close the program.

**SOURCE CODE:**

#include<stdio.h>

Struct fileTable

{

char name[20];

int sb, nob;

}ft[30];

void main()

{

int i, j, n;

char s[20];

printf("Enter number of files :");

scanf("%d",&n);

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

{

printf("\ntype file name %d :",i+1);

scanf("%s",ft[i].name);

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

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

printf("Type no of blocks in file %d :",i+1);

scanf("%d",&ft[i].nob);

}

printf("\nType the file name to be searched ");

scanf("%s",s);

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

if(strcmp(s, ft[i].name)==0)

break;

if(i==n)

printf("\nFile Not Found");

else

{

printf("\nfile name start block no of blocks blocks occupied\n"); printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);

for(j=0;j<ft[i].nob;j++)

printf("%d, ",ft[i].sb+j);

}

}

**EX:NO:15B Implementation of the following File Allocation Strategies - Indexed**

#include<stdio.h>

struct FT

{

char name[20];

int nob, blocks[30];

}ft[30];

void main()

{

int i, j, n;

char s[20];

printf("Type no of files :");

scanf("%d",&n);

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

{

printf("\nType the file name %d :",i+1);

scanf("%s",ft[i].name);

printf("Type no of blocks in file %d :",i+1);

scanf("%d",&ft[i].nob);

printf("Type the blocks of the file:");

for(j=0;j<ft[i].nob;j++)

scanf("%d",&ft[i].blocks[j]);

}

printf("\nType the file name to be searched ");

scanf("%s",s);

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

if(strcmp(s, ft[i].name)==0)

break;

if(i==n)

printf("\nFile Not Found");

else

{

printf("\nfile name no of blocks blocks occupied");

printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);

for(j=0;j<ft[i].nob;j++)

printf("%d, ",ft[i].blocks[j]);

}

}

**EX:NO:15C Implementation of the following File Allocation Strategies Linked**

#include<stdio.h>

#include<malloc.h>

struct FT

{

char name[20];

int nob;

struct block \*sb;

}ft[30];

struct block

{

int bno;

struct block \*next;

};

void main()

{

int i, j, n;

char s[20];

struct block \*temp;

printf("Type no of files :");

scanf("%d",&n);

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

{

printf("\nType file name %d :",i+1);

scanf("%s",ft[i].name);

printf("Type no of blocks in file %d :",i+1);

scanf("%d",&ft[i].nob);

ft[i].sb=(struct block\*)malloc(sizeof(struct block));

temp = ft[i].sb;

printf("Type the blocks of the file :");

scanf("%d",&temp->bno);

temp->next=NULL;

for(j=1;j<ft[i].nob;j++)

{

temp->next = (struct block\*)malloc(sizeof(struct block));

temp = temp->next;

scanf("%d",&temp->bno);

}

temp->next = NULL;

}

printf("\nType the file name to be searched -- ");

scanf("%s",s);

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

if(strcmp(s, ft[i].name)==0)

break;

if(i==n)

printf("\nFile Not Found");

else

{

printf("\nfile name no of blocks blocks occupied");

printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);

temp=ft[i].sb;

for(j=0;j<ft[i].nob;j++)

{

printf("%d -->",temp->bno);

temp = temp->next;

}

}

}