**EX: NO: 1**

**BASIC COMMANDS FOR LINUX**

**AIM:**

 To study the basic **commands for LINUX** command.

**COMMAND:**

**1.DATE COMMAND:**

 SYNTAX: $ date

 DESCRIPTION: The date command tells us the correct date and time.

Eg: $date

Nov 1 09:34:50 IST 1984

$

OUTPUT:

Fri Jul 13 15:57:19 IST 2011

**2. WHO COMMAND:**

 SYNTAX: $ who

 DESCRIPTION: The output of the who command gives us the details of the user

 who has logged into the system currently

 Eg: $who

 abc hyo sep26 11:17

xyz hyo sep26 11:17

lkg hyo sep26 11:17

$

OUTPUT:

User 07 pts/6 2011-07-13 15:46 (169.254.195.8)

User 16 pts/8 2011-07-13 15:47 (169.254.195.7)

User 15 pts/16 2011-07-13 15:57 (169.254.195.45)

**3. WHO AM I COMMAND:**

 SYNTAX: $ who am i

 DESCRIPTION: Which gives us the details regarding the login time and the

 system name for the connection being used

Eg: $ who am i

 User1 HYA sep25 12:20

 $

 OUTPUT:

 User 15 pts/16 2011-07-03 15:57 (169.254.195.45)

**4. MAN COMMAND:**

 SYNTAX: $ man

 DESCRIPTION: If we get struck on something and cannot find an except to help,

 we can point any manual page on our terminal with the command ‘man’.

 Eg: $ man who

 OUTPUT:

 What manual page do you want?

**5. HEAD AND TAIL COMMAND:**

 SYNTAX: $ head [count] [filename....]

 DESCRIPTION: The head and tail command is used to display the initial part of a

text file. This can be as a complete command to put and which displays the last

part andtext file. By default head and display 1st 10 lines of 6 files and last 10

lines.

 Eg: $ head 4 word list

 Pen

 Bat

 Ink

 Eraser

 Syntax: $ tail (+/- number) file

 Eg: $ tail 10mall

 $

 OUTPUT:

 Head. 4 round.c

 Tail .+10 man

**6. PUT COMMAND:**

 SYNTAX: $ pwd

DESCRIPTION: The pwd command has no options. It displays full name for the

 event directory use all looking for.

 Eg: $ pwd

 1 user 1temp

 $

OUTPUT:

 /home /user15/cse

**7. LS COMM**

**AND:**

 SYNTAX: $ ls

 DESCRIPTION: The ls command displays the list of files in the current working

directory.

 Eg: $ ls

 abc.c

 xy z

 lk z

 $

**SUB COMMANDS OF LS:**

LS-L list file in long format. The files are displayed along with their node number

of links, owner of file, modification and time and file name.

LS-T list in a order of list modification time.

LS-A list all enteries, including hidden files.

LS-D list directories file name instead if its contents.

LS-P puts a flash after each directories.

LS-U list in order of last access time.

OUTPUT:

Priority.c round.c

**8.MKDIR COMMAND:**

SYNTAX: $ mkdir

DESCRIPTION: This command is used to create a new directory.

Eg: $ mkdir temp

$

OUTPUT:

mkdir cse

**9. CD COMMAND:**

SYNTAX: $ cd

DESCRIPTION: The cd command used to change from the working directory to

any other directory specified. There is no option connected with this.

Eg: $ cd jack

$pwd

$jack

OUTPUT:

cd cse

[user 15 @local host cse]$

**10. CAT COMMAND:**

SYNTAX: $ cat[option....][file.....]

DESCRIPTION: The cat command helps us to list the contents of file use specify.

If we do not specify the name of the file it task input from the standard

input/output

CAT-S- support about non-existents files.

Eg: $ cat-arg

To be (or) not to be that in question

 $.

OUTPUT:

cat[4] angry

**11. RMDIR COMMAND:**

SYNTAX: $ rmdir directory name

DESCRIPTION: This command is used to remove a directory specified in the

command line. It requires the specified directories to be empty before removing it.

Eg: $ rmdir temp

$

OUTPUT:

rmdir cse

**12. CP COMMAND**:

SYNTAX: $cp file target

DESCRIPTION: The cp command is used to create duplicate copier of ordinary

file.

Eg: $ cp abc lkj

OUTPUT:

cp priority.c

**13. LN COMMAND:**

SYNTAX: $ ln first name second name

DESCRIPTION: The ln command is to established an additional file name for the

same ordinary files.

Eg: $ ln rules

$

OUTPUT:

ln cse

**14. MV COMMAND:**

SYNTAX: $ mv file target

DESCRIPTION: The mv command is used to rename and more ordinary and

directory file. To do thus we need both execute and write permission

Eg: $ mv old name

 $

OUTPUT:

mv cseit cseeee

mv cseeee

**15. RM COMMAND:**

SYNTAX: $ rm [option] file

DESCRIPTION: The rm command is used to remove or move file from a directory.

This can be used to delete all files as well as directory.

RM-i- asks user is wants to delete the file mentioned. When this is combined with

option run asks whether to examine each file in the directory rm-s-recursively delete

theentire of directory itself.

Eg: $ rm temp1 not

OUTPUT:

rm cse not

**MODELS IN VI:**

The vi editor works in these different models shown below.

 **1. INPUT MODE:** Any key that is pressed in this mode is entered as text.

The following are some of commands:

| **Commands**  | **Functions** |
| --- | --- |
| i | Inserts any character at cursor position |
| a | Appends text |
| o | Opens a new blank line |
| r | Replace a single character |
| R | Replaces more than single character |

 **2. EX MODE:**

In this mode the commands are entered in the last line after a “;” symbol. Some of the commands used in the mode are:

| **Commands**  | **Functions** |
| --- | --- |
| :w | Saves a file and remains in editing mode |
| :wq | Saves file and quits the editor |
| :qa | Quits editor without saving changes |

 **3.COMMAND MODE:**

| **Command** | **Functions** |
| --- | --- |
| H | Moves the cursor left |
| L | Moves the cursor right  |
| J | Moves the cursor down |
| K | Moves the cursor up |
| x | Deletes the entire character |
| dd | Deletes the entire line  |
| <ctrl-f> | Scrolls full page forward |
| <ctrl-b> | Scrolls full page backward |
| <ctrl-d> | Scrolls half page forward |
| <ctrl-u> | Scroll half page backward |

The key used here act as commands on the text. We can invoke this mode by pressing <esc> key

**RESULT:**

Thus the **LINUX commands** are implemented and the output was verified

**EX:NO: 2 SHELL PROGRAMMING**

**THE SHELL:**

The shell process the instructions that are issued to the system by the user.

The shell can also be used as a programming language because it provides many

features and special commands.

Any file consisting of a sequence of commands is known as a shell program or a

shell script.

* **VARIABLES:**

Variables in the shell are indicated by the symbol “$” which precedes the name

of the variable.

A variable can either be an integer of a string and is automatically declared at its

first usage.

* **INPUT/OUTPUT COMMANDS:**

 **read**------ used to read one or more variables.

**echo**------- prints either srtings or values or both

* **ARITHMETIC OPERATION:**

 **expr**------- used to evaluate the expression.

 Eg: X= `expr $a+$b`

* **CONDITIONAL STATEMENTS:**

1) **if statement:**

**Syntax:**

 if(condition)

 then

 commands

 else

 commands fi

2) **case statements:**

**Syntax:**

 case $ variable in

 value 1) commands

 “

 “

value 2) commands

 “

 “

 value n) commands

 esac

3) **while statement:**

**Syntax:**

 while(condition)

 do

 commands

 done

4) **until statement:**

**Syntax:**

 until(condition)

 do

 commands

 done

5) **for statement:**

 **Syntax:**

 for variables in the <list of values>

 do

 commands

 done

* **BREAK AND CONTINUE:**

Break is used to transfer control out of the loop.

 Continue is used to continue with the next iteration.

* **RELATIONS OPERATOR:**

**-**eq: equal to

-ne: not equal to

-gt: greater than

-ge:greater than equal to

-lt: less than

-le: lesser than equal to

**RESULT:**

 Thus the **Shell programming** was studied

**EX: NO: 2a FIBONACCI SERIES**

**AIM:**

To write a Shell program to generate **Fibonacci series**in LINUX operating system.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Get the limit from the user.

**STEP 3:** Check the condition to generate the Fibonacci series.

**STEP 4:** By using the temp variable the Fibonacci series is generated and

is displayed.

**STEP 5:** Stop the program.

**PROGRAM:**

clear

echo “OUTPUT”

echo “Enter the limit”

read n

echo “The Fibonacci series”

b=0

c=1

d=0

i=0

if[$n-ge 2]

then

echo “$b $c”

n=$(($n-2))

while[$i-lt $n]

do

a=$(($b+$c))

b=$c

c=$a

echo “$c”

i=$(($i+1))

done

else

echo “$b”

fi

**OUTPUT:**

Enter the limit:

5

The Fibonacci series:

0

1

1

2

3

**RESULT:**

Thus the Shell program for generating **Fibonacci series** in LINUX OS

was executed and output was verified successfully.

**EX: NO: 2b SUM OF DIGITS**

**AIM:**

 To write a Shell program to compute **sum of digits** in LINUX operating system

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Get the value from the user.

**STEP 3:** Display the number of digits in the integer value.

**STEP 4:** Add all the digits in the number and display it.

**STEP 5:** Stop the program.

**PROGRAM:**

clear

echo “OUTPUT:”

echo “Enter an integer”

read num

sum=0

digit=0

while [$num-gt 0]

do

a=$(($num%10))

digit=$(($digit+1))

sum=$(($sum+$a))

num=$(($num/10))

done

echo “The no of digits in the integer is $digit”

echo “The sum of digits is $sum”

**OUTPUT:**

Enter an integer:

1234

The no of digits in the integer is:4

The sum of digits is: 10

**RESULT:**

Thus the Shell program for performing **sum of digits**in LINUX OS

was executed and output was verified successfully.

**EX: NO: 2c CONVERT LOWER TO UPPER**

**AIM:**

To write a Shell program to**convert lower to upper**case letters in

LINUX operating system.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Get the input string from the user.

**STEP 3:** Using the condition convert the string in lower case to upper case.

**STEP 4:** Display the converted string.

**STEP 5:** Stop the program.

**PROGRAM:**

clear

echo “OUTPUT”

echo “Enter a string”

read str

temp=’echo $str| tr[a=z][A-Z]’

echo “The case changed String is $temp”

**OUTPUT:**

Enter a string:

system

The case changed string is SYSTEM

**RESULT:**

 Thus the SHELL program for **converting lower to upper**in LINUX OS was executed and output was verified successfully.

**EX: NO: 2d CURRENTLY LOGGED IN OR NOT**

**AIM:**

To write a Shell program to check whether a user is **currently logged**

**In or not** in Linux operating system.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Get the name of the user.

**STEP 3:** Using the condition check whether the user is logged in or not.

**STEP 4:** Display the current position of the user.

**STEP 5:** Stop the program.

**PROGRAM:**

clear

echo “OUTPUT:”

echo “enter the name of the user:”

read name

who>temp

if(grep $name temp>temp1)

then

 echo “$name is currently logged in”

else

 echo “$name is not currently logged in”

fi

**OUTPUT:**

Enter the name of the user: Bhuvana

Bhuvana is currently logged in

**RESULT:**

Thus the Shell program to whether a user is **currently logged in or not**

in LINUX OS was executed and output was verified success

**EX: NO: 2e SWAPPING TWO NUMBERS**

**AIM:**

To write a Shell program to **swap two numbers**in LINUX operating system.

**ALGORITHM:**

**STEP 1:**Start the program.

**STEP 2:** Get the two input values from the user.

**STEP 3:** Swap the values of two variables using the temp variable

 t=a, a=b, b=t .

**STEP 4:**The swapped values are displayed as output.

**STEP 5:** Stop the program.

**PROGRAM:**

echo “Enter the two numbers: “

read a b

echo $a $b

$t=$a

$a=$b

$b=$t

echo “After swapping:”

echo $a $b

**OUTPUT:**

Enter the two numbers:

5 9

After swapping:

9 5

**RESULT:**

Thus the Shell program to **swap two numbers**in LINUX OS was executed and output was verified successfully

**EX: NO: 2f COMPARE TWO STRINGS**

**AIM:**

To write a Shell program **to compare two strings** in LINUX

operating system.

**ALGORITHM:**

**STEP 1:**Start the program.

**STEP 2:** Get the two input strings from the user.

**STEP 3:** Compare whether the contents in both the strings are equal or not.

**STEP 4:**If the two strings are same display the strings are equal if not display

they are not equal.

**STEP 5:** Stop the program.

**PROGRAM:**

clear

echo “Output:”

echo “Enter the first string:”

read str1

echo “Enter the second string:”

read str2

if[$str1=$str2]

then

echo “Strings are equal”

else

echo “Strings are not equal”

fi

**OUTPUT:**

Enter the first string:

good

Enter the second string:

good

Strings are equal

Enter the first string:

christ

Enter the second string:

christian

Strings are not equal

**RESULT:**

Thus the Shell program for **comparing two strings** in LINUX OS

was executed and output was verified successfully.

**EX: NO: 2g REVERSE A STRING**

**AIM:**

To write a SHELL program to**reverse a string** in LINUX

operating system.

**ALGORITHM:**

**STEP 1:**Start the program.

**STEP 2:** Get the input string from the user.

**STEP 3:** Calculate the length of the string.

**STEP 4:**Reverse the string and display the reversed string.

**STEP 5:** Stop the program.

**PROGRAM:**

clear

echo “Output:”

echo “Enter the string:”

read str

len=’echo $str / wc-c’

while[$len-gt 0]

do

temp=’echo $str / cut-c $len’

rev=’echo $rev $temp ‘

len=’expr $ len-1’

done

echo “The reverse string is $ rev”

**OUTPUT:**

Enter the string:

morning

The reverse string is :

gninrom

**RESULT:**

Thus the SHELL program for **reversing a string** in LINUX OS

 was executed and output was verified successfully.

**EX: NO: 3a SYSTEM CALLS OF LINUX OPERATING SYSTEM**

**FORK AND WAIT SYSTEM CALL**

**AIM:**

To write a c program to perform **fork and wait system cal**l in LINUX

 operating system.

**ALGORITHM:**

**STEP1:** Start the program.

**STEP 2:** Declare the variable pd as integer.

**STEP 3:** Assign pd to fork (). and print hello.

**STEP 4:** Check condition pi==0 using if statement.

**STEP 5:** If condition satisfied then block the child process for 20 sec by using wait (20) method.

**STEP6:** Now print the no. of child process which is executed by using process

 and get the process id by using get () method and print child process is

 executed

**STEP 7:** Otherwise check condition pd> 0 using if statement. and block the

parent process for 50 sec by using wait (50) method.

**STEP 8:** Now print the no. of parent process executed by using process id. and

get the process id by using get () method and print that the parent

process is executed

**STEP 9:** Stop the program.

**PROGRAM:**

#include<stdio.h>

main()

{

int pid;

pid=fork();

printf("\nHELLO\n");

if(pid==0)

{

wait(20);

printf("\nChild process is execuited");

printf("\nThe ID number of child process is %d\n",getpid());

}

else if(pid>0)

{

wait(50);

printf("\nParent process is execuited");

printf("The ID number of parent process is %d\n",getpid());

}

}

**OUTPUT:**

HELLO

Child process is execuited

The ID number of child process is 2492

Parent process is execuited

The ID number of parent process is 2491

**RESULT:**

Thus the c program to perform **fork and wait system call** for LINUX operating system is executed and the output is verified successfully.

**EX: NO: 3b**

**CREATE A CHILD PROCESS USING SLEEP SYSTEM CALL**

**AIM:**

To write a c program **to create a child process using sleepcommand**

 in LINUX operating system.

**ALGORITHM:**

**STEP 1:**Start the program.

**STEP 2:**Declare the variable pid as integer.

**STEP 3:**Assign pid to fork(). and print hello.

**STEP4:**Check condition pid==0 using if statement

**STEP5:**Now print the no. of child process which is executed by using process id and get the process id by using getpid() method and print child process is executed.

**STEP6:** Otherwise check condition pid>0 using if statement. and stop the parent process for 20 sec by using sleep(20) method.

**STEP7:**Now print the no. of parent process executed by using process id. and the process id by using getpid() method and print that the parent process is executed.

**STEP8:**Stop the program.

**PROGRAM:**

#include<stdio.h>

main()

{

int pid;

pid=fork();

printf("\nHELLO\n");

if(pid==0)

{

printf("\nChild process is execuited");

printf("\nThe ID number of child process is %d\n",getpid());

}

else if(pid>0)

{

sleep(20);

printf("\nParent process is execuited");

printf("The ID number of parent process is %d\n",getpid());

}

}

**OUTPUT:**

HELLO

HELLO

Child process is execuited

The ID number of child process is 2492

Parent process is execuited

The ID number of parent process is 2491

**RESULT:**

Thus the c program to **create a child process using sleep system call** in LINUX operating system is executed and the output is verified successfully.

**EX: NO: 3c EXIT SYSTEM CALL**

**AIM:**

To write a c program to **exit a system call** in LINUX operating system.

**ALGORITHM**

**STEP1:** Start the programs.

**STEP2:** Declare the required variable

**STEP3:** By using for () enter your choice to continue or exit.

**STEP4:** Check condition a==1using if statement.

**STEP5:** If condition satisfied the print your choice to exit and perform exit(0).

**STEP6:**Else print your choice to continue

**STEP7:** Stop the program.

**PROGRAM:**

#include<stdio.h>

main()

{

int i,a;

for(i=1;i<=3;i++)
{

printf("\nEnter your choice [1=exit/2=continue]:");
scanf(%d”,&a);
if(a==1)

{

printf("Your choice to exit");
exit(0);
}
else
{

printf("Your choice to continue”);
}

}

}

**OUTPUT:**

 Enter your choice [1=exit/2=continue]:

 Your choice to exit

 Enter your choice [1=exit/2=continue]:

 Your choice to continue

**RESULT:**

Thus the C program to **exit the system call** in LINUX operating system is executed and the output is verified successfully.

**EX: NO: 3d DIRECTORY SYSTEM CALL**

**AIM:**

To write a c program to perform a **directory system calls** in LINUX operating system.

**ALGORITHM:**

**STEP1:** Start the program.

**STEP2:** Get directory name by using dirname

**STEP3:** If the given directory is open then read the directory. and store it in preaddr.

**STEP4:** Check condition pread dr==null using if statement.

**STEP5:** If condition satisfied the close the directory using close(dirname) and exit it.

**STEP6:** Print the found entry of the directory.

**STEP7:**Stop the program.

**PROGRAM:**

#include<stdio.h>
#include<dirent.h>

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

{
DIR \*dirname;
struct dirent\* preaddr;
dirname=opendir(argv[1]);
while(1)
{
preaddr=readdir(dirname);
if(preaddr==NULL)
{
closedir(dirname);
exit(0);
}
printf(“\n\nFound Entry %s %s “,argv[i],preaddr->dirname);

}

}

**OUTPUT:**

 Found Entry jerline

 Found Entry jerline

**RESULT:**

Thus the c program to perform a **directory system call** in LINUX operating system is executed and the output is verified successfully.

**EX: NO: 4**

**INPUT OUTPUT SYSTEM CALLS**

 **FILE SYSTEM CALL USING OPEN AND CLOSE**

**AIM:**

To write a c program to perform **file system callusing open and close** in

 Linux operating system.

**ALGORITHM:**

**STEP1:** Start the program.

**STEP2:** Declare the required variables.

**STEP3:**Check condition argc<3 using if statement. then print illegal input and exit the process.

**STEP4:**Now open the file and assign it to fd.

**STEP5:**Check condition fd==-1 then print error occurred and exit the process

**STEP6:**Now create a process by using create(argr[2],9999) and assign it to r.

 check condition r= =-1 using if statement and print file is not created

and exit the process .

**STEP7:**Now read the file and assign sign to the file by using read(fd,s,size).

while (rd>0) perform read and write operation using write(r,s,size) and read(fd,s,size) methods.

**STEP8:**Close all the read and write process.

**STEP9:**Stop the program.

**PROGRAM:**

#include<stdio.h>
#define size 10
main(int argc,char \*argr[])
{
int i,n,rd,wd,r,fd;
char s[size];
if(argc<3)
{
printf(“ILLEGAL INPUT”);
exit(1);
}
fd=open(argr[1]);
if(fd==-1)
{
printf(“ERROR OCCURRED”);
exit(1);
}
r=create(argr[2],9999);
if(r==-1)
{
printf(“FILE NOT CREATED”);
exit(1);
}
rd=read(fd,s,size);
while(rd>0)
{
wd=write(r,s,size);
rd=read(fd,s,size);
}
close(fd);
close(r);
printf(“FILE COMPLETED”);
}

**OUTPUT:**

 FILE COMPLETED

 How are you

**RESULT:**

Thus the c program to perform **file system call using open and close**

in LINUXoperating system is executed and the output is verified successfully.

**EX: NO: 5a SIMULATION OF LINUX COMMANDS**

 **SIMULATION OF LS COMMAND**

**AIM:**

To write a c program to **simulate the LS command** in LINUX operating system.

**ALGORITHM:**

**STEP1:** Start the program.

**STEP2**: Initially assign number of character is three using an array of character.

**STEP3:**Assign array of character for index zero as ls.

**STEP 4:**Assign array of character for index one as -1.

**STEP 5:**Finally assign array of character of 3 as (char \*) zero.

**STEP 6:**Print all the values stored in an array.

**STEP7:** Stop the program.

**PROGRAM:**

#include<stdio.h>
#include<stdlib.h>
main()
{
char \*one[3];
one[0]=”\*LS”;
one[1]=”\*-1”;
one[2]=(char \*)0;

Exec1(“/bin/LS/”,one[0],one[1],one[2]);
}

**OUTPUT:**

total 44
-rwx rw\*r-xi iiii to 8 iiii to 8 11570 jan 100:54a.out
-rw rw-r-1 iiii to 08 iiii to8 709 jan 11999 fr c

**RESULT:**

Thus the c program for **simulation of LS command** in LINUX operating system is executed and the output is verified successfully.

**EX: NO: 5b SIMULATION OF GREP COMMAND**

**AIM:**

To write a c program to **simulate the GREP command** in LINUX operating system.

**ALGORITHM:**

**STEP1:**Start the program.

**STEP2:**Create a file and declare the required variables.

**STEP3:**Enter the file name and string to enter into a file to open.

**STEP4:** While file is not equal to end of file the perform the following operation.

**STEP5:** Assign mstr[i]=c and check condition c==’\n’ using if statement.

**STEP6:** If condition satisfied then assign mstr[i]=’\0’, rtr=str(mstr,cstr)and check

condition ptr!=null

**STEP 7:**Then print that the word is found in the file and increment the count value and i

value by one.

**STEP 8:**Check condition if count==0 then print no occurrence is found else print occurrence found in count.

 **STEP 9:** Stop the program.

**PROGRAM:**

#include<stdio.h>
#include<conio.h>
void main()
{
File \*fp;
char mstr[500],cst[50],c,\*ptr,fname[20];
int i=0,count=0;
fflush(stdin);
printf(“\nEnter the string and file name:”);
scanf(“%s%s”,cstr,fname);
fp=fopen(fname,”r”);
while((c=getc(fp))!=EOF)
{
mstr[i]=c;
if(c==’\0’)
{
mstr[i]=c;
if(C==’\n’)
{
mstr[i]=’\0\;
rtr=str(mstr,cstr);
if(ptr!=NULL)
{
printf(“\n[%s] Found in \n\n%s \n”,cstr,mstr);
count++;
}
i=0;
}
i++;
}
if(count==0)
printf(“\nNo occurrence is found”);
else
printf(“\nOccurence is found in : %d”,count);
}

**OUTPUT:**

Enter the string and file name: hello

File1.text

 [hello] Found in

 Hello world , how are u?

**RESULT:**

Thus the c program to **simulate GREP command in** LINUX operating system is executed and the output is verified successfully.

 **SIMULATION OF PROCESSES SCHEDULING ALGORITHM**

**EX: NO: 6a FIRST COME FIRST SERVE**

**AIM:**

 To write a C program to illustrate **CPU scheduling using FCFS** method

**ALGORITHM:**

**STEP 1:** Start the program

**STEP 2:** Get the number of process

**STEP 3:**Get the processing or execution time

**STEP 4:**Calculate average waiting time

**STEP 5:**Average waiting time=sum of waiting time/no of process

**STEP 6:** Calculate average turnaround time

**STEP 7:** Average turnaround time=sum of turnaround time /no of process

**STEP 8:** Display the average waiting time and turnaround time

**STEP 9:** Stop the program

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

int main()

{

int n,i,bt[10],tt=0,wt=0;

float ta,wa,a=0,b=0;

clrscr();

printf("\n enter the no.of.process...");

scanf("%d",&n);

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

{

printf("\n enter the burst time for p%d...",i);

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

}

printf("\n\n process\t ta time\t wait time\n\n");

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

{

printf("\n p%d",i);

tt=tt+bt[i];

a=a+tt;

printf("\t\t%d",tt);

printf("\t\t%d",wt);

b=b+wt;

wt=tt;

}

ta=a/n;

wa=b/n;

printf("\n\n average turnaround time:\t%f\n average waiting time:\t%f",ta,wa);

getch();

}

**OUTPUT:**

Enter the no.of.process…4

Enter the burst time for P1…1

Enter the burst time for P2…2

Enter the burst time for P3…3

Enter the burst time for P4....4

Process ta time wait time

P1 1 0

P2 3 1

P3 6 3

P4 10 6

 average turnaround time:2.50000

average waiting time: 5.00000

**RESULT:**

 Thus the C program to illustrate **CPU scheduling using FCFS method** was executed successfully and output was verified

**EX: NO: 6b SHORTEST JOB FIRST**

**AIM:**

To write a program to implement the SJF scheduling algorithm

**ALGORITHM:**

 **STEP 1:** Start the program

**STEP 2:** Get the number of process

**STEP 3:** Get the processing or execution time

**STEP 4:** Sort the execution time

**STEP 5:** Calculate average waiting time

 Average waiting time=sum of waiting time/no of process

**STEP 6:** Calculate average turnaround time

 Average turnaround time=sum of turnaround time /no of process

**STEP 7:** Display the average waiting time and turnaround time

**STEP 8:** Stop the program

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

int o[10],j,t,n,i,bt[10],tt=0,wt=0;

float ta,wa,a=0,b=0;

clrscr();

printf("\n enter the no.of.process...");

scanf("%d",&n);

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

{

printf("\n enter the burst time for p%d...",i);

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

o[i]=1;

}

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

{

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

 {

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

 {

 t=bt[i];

 bt[i]=bt[j];

 bt[j]=t;

 t=o[i];

 o[i]=o[j];

 o[j]=t;

 }

 }

 }

printf("\n\n process\tTA time \twait time\n\n");

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

{

printf("\n\tp%d",i);

tt=tt+bt[i];

a=a+tt;

printf("\t\t%d",tt);

printf("\t\t%d",wt);

b=b+wt;

wt=tt;

}

ta=a/n;

wa=b/n;

printf("\n\n average turnaround time:\t%f\n average waiting time:\t%f",ta,wa)

getch();}

**OUTPUT:**

Enter the no.of.process…4

Enter the burst time for P1…1

Enter the burst time for P2…2

Enter the burst time for P3…3

Enter the burst time for P4....4

Process ta time wait time

P1 1 0

P2 3 1

P3 6 3

P4 10 6

 average turnaround time:2.50000

average waiting time: 5.00000

**RESULT:**

Thus the C program to implement FCFS scheduling was executed and output was verified

**EX:NO: 6c PRIORITY SCHEDULING**

**AIM:**

To write a program to implement **priority the scheduling** algorithm.

**ALGORITHM:**

**STEP1**: Start

**STEP2:** Declare the array size

**STEP3:** Get the number of process and its corresponding execution time

**STEP4:** Get the valuewith priority the process number and its execution time

 has been calculated for n numbers process do,

**STEP5:** Print the process and its waiting time

**STEP6:** Calculate by using formula

 **STEP7:** Find the total waiting time that is total waiting time=sum of waiting

time of n process

**STEP 8:** Find the average and print it

**STEP 9:** Stop

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

int p[10],o[10],j,t,n,i,bt[10],tt=0,wt=0;

float ta,wa,a=0,b=0;

clrscr();

printf("\n enter the no.of.process...");

scanf("%d",&n);

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

{

printf("\n enter the burst time for p%d...",i);

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

printf("\n enter the priority for p%d...",i);

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

o[i]=i;

}

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

{

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

 {

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

 {

 t=p[i];

 p[i]=p[j];

 p[j]=t;

 t=o[i];

 o[i]=o[j];

 o[j]=t;

 t=bt[i];

 bt[i]=bt[j];

 bt[j]=t;

 }

 }

 }

printf("\n\n process\tTA time \twait time\n\n");

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

{

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

tt=tt+bt[i];

a=a+tt;

printf("\t\t%d",tt);

printf("\t\t%d",wt);

b+=wt;

wt=tt;

}

ta=a/n;

wa=b/n;

printf("\n\n average turnaround time:\t%f\n average waiting time:\t%f",ta,wa);

getch();

}

**OUTPUT:**

Enter the no.of.process…4

Enter the burst time for P1…5

Enter the priority for P1…3

Enter the burst time for P2…7

Enter the priority for P2*....2*

Enter the burst time for P3…4

Enter the priority burst time for P3…4

Enter the burst time for P4…4

Enter the priority for P4....1

Process ta time wait time

P4 0 4

P2 4 11

P1 11 16

P3 16 19

 average turnaround time:12.50000

average waiting time: 7.7500000

**RESULT:**

 Thus the C program to implement**priority scheduling**was executed and output is verified.

**Ex:no:6d ROUND ROBIN SCHEDULING**

**AIM:**

To write a C program to implement the concept of **round robin scheduling** algorithm

**ALGORITM:**

**STEP1:** Start the program.

**STEP2:** Get the execution time of number of process and the time

**STEP3:** Perform step 4-6 until the completion of all the n numbers of process

**STEP4:** If the process needs another cycle for connection to go to step5 else.

**STEP5**: Calculate the remaining execution time and print the current status,

total time spend andthe individual completion time

**STEP6:** Print the process is complete and calculate the waiting time

 a)Waiting time of ith process= total time spent so for

b) It’s individual completion time in previous cycle

 c) Print the individual waiting time of n process and find the total waiting time

 d) Print the average waiting time and print the total and average waiting time

 **STEP7:**Stop the program

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

int o[10],j,t,n,i,bt[10],tt=0,wt=0,ts;

float ta,wa,a=0,b=0;

clrscr();

printf("\n enter the no.of.process...");

scanf("%d",&n);

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

{

printf("\n enter the burst time for p%d...",i);

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

}

printf(“\n enter the time slice….”);

scanf(“%d”,&ts);

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

{

no+=bt[i]/ts;

if((bt[i]/ts;

no++;

r[i]=bt[i];

ct[i]=0;}

do

{

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

 {

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

 {

 t=bt[i];

 bt[i]=bt[j];

 bt[j]=t;

 t=o[i];

 o[i]=o[j];

 }

 }

 }

printf("\n\n process\tTA time \twait time\n\n");

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

{

printf("\n\tp%d",i);

tt=tt+bt[i];

a=a+tt;

printf("\t\t%d",tt);

printf("\t\t%d",wt);

b=b+wt;

wt=tt;

}

ta=a/n;

wa=b/n;

printf("\n\n average turnaround time:\t%f\n average waiting time:\t%f",ta,wa);

getch();

}

**OUTPUT:**

Enter the no.of.process…3

Enter the burst time for P1…5

Enter the burst time for P2…2

Enter the burst time for P3…4

Enter the time slice....2

Process ta time wait time

P1 11 6

P2 4 2

P3 10 6

 average turnaround time:4.667

average waiting time: 8.333

**RESULT:**

Thus the program to implement **Round robin scheduling** was executed

 and output wasverified

**SIMULATION OF SYNCHORIZATION PROBLEMS USING SEMAPHORE**

**EX: NO: 7a PRODUCER AND CONSUMER PROBLEM**

**AIM:**

To write a C program to implement the concept of **producer consumer problem**

**ALGORITHM:**

**STEP1**: Start the program.

**STEP2:** Declare the required variables of the program as global variables.

**STEP3:** In function producer() assign n=n+1 , f=f+1 and print producer has produce f.

**STEP4:** In function consumer() assign n=n-1 and print consumer has received f.

**STEP5:** In main() declared variable s as char.

**STEP6:** Now enter the state of the buffer and check condition if(s==’e’) then assign
n=0 else assign n=1

.**STEP7:** Inside while() check condition if(n==0) then call producer() and if(n==1)

then call consumer

**STEP8:** Check condition if(f%6==0) then print “do u want to continue(y/n):”

 if yes then theprocess is continued else the process is terminated

.**STEP 9:**Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int n=3,m,item,buf[3],i;

void wait()

{

m=1;

printf("\n Enter the critical section");

}

void signal()

{

m=0;

printf("\n Quitting the critical section");

}

void producer()

{

wait();

printf("Enter the data to produce:");

scanf("%d",&item);

buf[i]=item;

i++;

signal();

}

void consumer()

{

wait();

printf("\n The data to be consumed:%d",buf[i-1]);

i--;

signal();

}

void main()

{

int ch;

clrscr();

start:

printf("\n Menu:n\t1=producer\n\t2=consumer\n Enter your choice:");

scanf("%d",&ch);

if(ch==1)

{

if(1==n)

{

printf("\n Buffer full");

}

else

{

producer();

goto start;

}}

else if(ch==2)

{

if(i==0)

{

printf("\n no Item to consume");

}

else

{

consumer();

goto start;

}

}

else

{

exit(0);

}

getch();

**OUTPUT:**

MENU:

 1.Producer

 2.Consumer

Enter your choice: 1

Entering the critical section…….

Enter the data to produce: 6

Quitting the critical section…….

MENU:

 1.Producer

 2.Consumer

Enter your choice: 1

Entering the Critical section …….

Enter the data to produce: 5

Quitting the critical section………

MENU:

 1.Producer

 2.Consumer

Enter your choice: 2

Entering the critical section…….

The data has been Consumed : 5

Quitting the critical section………

MENU:

 1.Producer

 2.Consumer

Enter your choice: 2

Entering the critical section…..

The data has been Consumed: 6

Quitting the critical section…….

MENU:

 1.Producer

 2.Consumer

Enter your choice: 2

No item to Consume……..

**RESULT:**

Thus the C program for **producer consumer problem** is implemented and

executed and it is verified

**EX:NO:7b READER WRITER PROBLEM**

**AIM:**

To write a C program to find the solution for **reader writer problem**.

**ALGORITHM:**

**STEP1:**Start the program.

**STEP2:** In function write () assign initially r as 20 and by using the for()

decrement the r value

**STEP3:** In main ()declare the required variables and enter the number of process

required

.

**STEP4:**Get the process id and status of the process by using the for(). and initially

 rd[i]=0.

**STEP5:** Using for() check condition if(s[i]==1) then print process->p[i] is writing wait

 and increment the count value

**STEP6:**While (count!=0) then call write() and decrement the count value by one. and

print writing is completed.

**STEP7:**Else assign rd[i]=p[i].

**STEP8:**In for() check condition if(rd[i]!=0) then print there is no waiting process. and

 print the resultant list of the process.

**STEP9:**Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

void write()

{

int r=20,i;

for(i=1;i<=20;i++)

{

r--;

return;

}

}

void main()

{

clrscr();

int n,s[20],p[20],rd[20];

int count=0,i,a;

printf("\n\t\tREADERS WRITERS PROBLEM");

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

scanf("%d",&n);

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

{

printf("\nProcess\_ID:");

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

printf("Status(w=1/r=0):");

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

}

printf("\nProcess\_ID\tstatus");

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

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

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

rd[i]=0;

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

{

if(s[i]==1)

{

printf("\nprocess->%d is writing wait",p[i]);

count++;

while(count!=0)

{

write();

count--;

}

printf("\n\tWriting is completed:");

}

else

rd[i]=p[i];

}

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

{

if(rd[i]!=0)

{

printf("\nThere is no waiting process:");

printf("Readers List");

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

{

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

if(rd[i]!=0)

printf("\nProcess->%d",rd[i]);

}

}

}

if(count!=1)

printf("\n\tThere is no reading proccess:");

getch();

}

**OUTPUT:**

READER WRITER PROBLEM

Enter the no. of process: 4

Process\_ID:1

Status (w=1/r=0):1

Process\_ID:2

Status (w=1/r=0):0

Process\_ID:3

Status (w=1/r=0):1

Process\_ID:4

Status (w=1/r=0):0

Process\_ ID status

1 1

2 0

3 1

4 0

Process ->1 is writing wait

 Writing is completed:

Process ->3 is writing wait

 Writing is completed:

There is no waiting process: Readers List

Process->2

Process->4

 There is no reading process.

**RESULT:**

Thus the c program to implement **reader writer problem** is executed and

the output isverified successfully

**EX: NO: 7c DINING PHILOSOPHERS’ PROBLEM**

**AIM:**

To write a C program to implement the concept of **dining philosophers’ problem**

**ALGORITHM:**

**STEP 1:** Start the program

**STEP 2:** The status of the plate and chop stick is checked

**STEP 3:** If the plate and the adjacent chop stick is free philosopher sit

otherwise the next plate is checked

**STEP 4:** While the next philosopher is seated the previous philosopher seated plate is

freed and the step 3 is continued until all philosopher finishes

**STEP 5:** Stop the program

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

int a[5]={0,0,0,0,0},b[5]={0,0,0,0,0},g,x,y,z=0,i=0,j=0,k,n;

clrscr();

printf("\nEnter the no. of philosopher");

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

while(z<n)

{

if((a[(i%5)+1]==0)&&(b[(j%5)+1])&&(a[((i+1)%5)+1]==0))

{

a[(i%5)+1]=1;

b[(j%5)+1]=1;

a[((i+1)%5)+1]=1;

printf("\nPhilospher %d is eating\tChopstick

free:%d,%d\nplate:%d\n",++z,(i%5)+1,((i+1)%5)+1,(j%5)+1);

a[(x%5)+1]=0;

b[(y%5)+1]=0;

a[((x+1)%5)+1]=0;

x=i;

y=j;

}

 i=i+1;

j=j+1;

getch();

**OUTPUT:**

Enter the Number of the philosopher: 10

Philosopher 1 is eating

Chop Stick free: 1&2

Plate: 1

Philosopher 2 is eating

Chop Stick free: 3& 4

Plate: 3

Philosopher 3 is eating

Chop Stick free: 5&1

Plate: 5

Philosopher 4 is eating

Chop Stick free: 2&3

Plate: 2

Philosopher 5 is eating

Chop Stick free: 4&5

Plate: 4

Philosopher 6 is eating

Chop Stick free: 1&2

Plate: 1

Philosopher 7 is eating

Chop Stick free: 3&4

Plate: 3

Philosopher 8 is eating

Chop Stick free: 5&1

Plate: 5

Philosopher 9 is eating

Chop Stick free: 2&3

Plate: 2

Philosopher 10 is eating

Chop Stick free: 4&5

Plate: 4

**RESULT:**

Thus the **Dinning philosophers’ problem** is executed and the output was verified

**SIMULATION OFBASIC MEMORY MANAGEMENT SCHEMES**

**EX: NO: 8 STORAGE REPLACEMENT**

**AIM:**

To write a LINUX program to implement the concept of **storage replacement.**

**ALGORITHM:**

**STEP 1:**Start the program.

**STEP 2:**Declare the variables to be used.

**STEP 3:**The total number of blocks are got from the user.

**STEP 4:**First the blocks are allocated by getting the block number and block space.

**STEP 5:**The total number of jobs are got from the user.

**STEP 6:** The jobs are allocated by getting the job number and job space.

**STEP 7:**During the first fit the condition (job[j].space <= block[i].mspace) is checked,

 if the condition is true the job is allocated blocks.

**STEP 8:**During the best fit the condition (block[i].ospace > block[j].ospace) is checked

 if the condition is true the job is allocated blocks.

**STEP 9:**During the worst fit the condition (block[i].ospace < block[j].ospace) is

 checked, if the condition is true the job is allocated blocks.

**STEP 10:**Stop the program.

**PROGRAM:**

#include<stdlib.h>

#include<stdio.h>

#include<conio.h>

#define MAX 50

struct blockdetails

{

int bno;

int mspace;

int ospace;

}block[MAX];

struct jobdetails

{

int jno;

int space;

int status;

}job[MAX];

int m,n;

void allocateablock()

{

int i;

printf(“\n Enter the total number of blocks:”);

scanf(“%d”,&n);

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

{
block[i].bno=i+1;

printf(“\n Block number:%d \n”,block[i].bno);

printf(“\n Enter the block space:”);

scanf(“%d”,&block[i].ospace);

}

}

void allocatejob()

{

int i;

printf(“\n Enter the total number of jobs:”);

scanf(“%d”,&m);

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

{

job[i].jno=i+1;

printf(“\n Job number: %d”,job[i].jno);

printf(“\n Enter the job space:”);

scanf(“%d”,&job[i].space);

job[i].status=0;

}

clrscr();

printf(“\n Block number \t Block space”);

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

printf(“\n %d \t %d”,block[i].bno,block[i].ospace);

printf(“\n Job number \t Job space”);

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

printf(“\n %d \t %d”,job[i].jno,job[i].space);

}

void firstfit()

{

int i,j;

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

block[i].mspace=block[i].ospace;

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

{

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

{

if(job[j].space<=block[i].mspace)

{

printf(“\n Job %d is allocated in block %d”,job[j].jno,block[i].bno);

job[j].status=1;

block[i].mspace=block[i].mspace-job[j].space;

break;

}

}

}

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

{

if(job[j].status == 0)

printf(“\n Job %d is not allocated in any block %d”,job[j].jno);

}

}

void bestfit()

{

int i,j;

struct blockdetails temp;

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

{

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

{

if(block[i].ospace>block[j].ospace)

{

temp=block[i];

block[i]=block[j];

block[j]=temp;

}

}

}

void worstfit()

{

int i,j;

struct blockdetails temp;

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

{

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

{

if(block[i].ospace<block[j].ospace)

{

temp=block[i];

block[i]=block[j];

block[j]=temp;

}

}

}

firstfit();

}

void main()

{

clrscr();

allocateblock();

allocatejob();

printf(“\n First fit”);

firstfit();

printf(“\n Best fit”);

bestfit();

printf(“\n Worst fit”);

worstfit();

getch();

}

**OUTPUT:**

Enter the total number of blocks:3

Block number:1

Enter the block space:6

Block number:2

Enter the block space:4

Block number:3

Enter the block space:8

Enter the total number of jobs:3

Job number: 1

Enter the job space:4

Job number: 2

Enter the job space:6

Job number: 3

Enter the job space:8

Block number Block space

1. 6
2. 4
3. 8

Job number Job space

1. 4
2. 6
3. 8

First fit

Job 1 is allocated in Block 1

Job 2 is allocated in Block 3

Job 3 is not allocated in any Block

Best fit

Job 1 is allocated in Block 2

Job 2 is allocated in Block 1

Job 3 is allocated in Block 3

Worst fit

Job 1 is allocated in Block 3

Job 2 is allocated in Block 1

Job 3 is not allocated in any Block

**RESULT:**

Thus the LINUX program for performing **storage replacement**

was executed and output was verified successfully.

**SIMULATION OF VIRTUAL MEMORY MANAGEMENT SCHEMES**

**EX: NO: 9aFIFO PAGE REPLACEMENT**

**AIM:**

To write a C program to perform the operation of **FIFO page replacement**

**ALGORITHM:**

**STEP 1:**Start the program

**STEP 2:**Get the number of frames and number of elements in the reference string

**STEP 3:**Put each and every element into the frame

**STEP 4:**If the reference element is equal to the frame elements, replacement is not

 necessary

**STEP 5:**For each and every page replacement the page fault increases by one

**STEP 6:**Calculate and display the page fault

**STEP 7:**Stop the program

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int nf,no,hit,miss,ref[50],fr[50];

void getdata();

void fifo();

int search(int);

void getdata()

{

int i;

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

scanf("%d",&nf);

printf("\nEnter the number of elements in the reference string :");

scanf("%d",&no);

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

{

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

}

}

void fifo()

{

int i,j,temp=0,rear=-1;

miss=hit=0;

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

{

printf("\nThe number to be inserted is : %d",ref[i]);

printf("\t");

if(i>nf-1)

{

temp=search(ref[i]);

}

if(temp==0)

{rrtr

miss++;

if(rear==nf-1)

rear=0;

else if(rear<nf-1)

rear++;

fr[rear]=ref[i];

}

else

hit++;

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

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

}

printf("\n\nNumber of page faults =%d",miss);

printf("\n\nNumber of hits =%d",hit);

}

int search(int item)

{

int i;

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

{

if(fr[i]==item)

return(1);

}

return 0;

}

void main()

{

int i;

char c;

clrscr();

gotoxy(22,2);

printf("FIFO PAGE REPLACEMENT");

gotoxy(22,4);

getdata();

fifo();

getch();

}

**OUTPUT**

FIFO PAGE REPLACEMENT

Enter the number of frames: 3

Enter the number of elements in the reference string:20

7

0

1

2

0

3

0

4

2

3

0

3

2

1

2

0

1

7

0

1

The number to be inserted :7 700

The number to be inserted :0 700

The number to be inserted :1 701

The number to be inserted :2 201

The number to be inserted :0 201

The number to be inserted :3 231

The number to be inserted :0 230

The number to be inserted :4 430

The number to be inserted :2 420

The number to be inserted :3 423

The number to be inserted :0 023

The number to be inserted :3 023

The number to be inserted :2 023

The number to be inserted :1 013

The number to be inserted :2 012

The number to be inserted :0 012

The number to be inserted :1 012

The number to be inserted :7 712

The number to be inserted :0 702

The number to be inserted :1 701

Number of page faults =15

**RESULT**:

 Thus the program to perform **FIFO page replacement**was executed

and output is verified.

**EX:NO:9b**

**LRU PAGE REPLACEMENT**

**AIM:**

To write a C program to perform the operation of **LRU page replacement**

**ALGORITHM:**

**STEP 1:**Start the program

**STEP 2:**Get the number of frames and the number of reference string

**STEP 3:**Put the elements in the frame

**STEP 4:**If the frame is full, remove the element which is least recently

used from the left

**STEP 5:**For each and every replacement the page fault is increased by one

**STEP 6:**Display the page fault

**STEP 7:**Stop the program

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

int fr[3],nf;

void main()

{

void display();

int p[50],i,j,fs[3],no;

int ind,k,l,fl1=0,fl2=0,pf=0,frsize=3;

clrscr();

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

scanf("%d",&nf);

printf("\nEnter the number of elements in the reference string");

scanf("%d",&no);

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

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

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

fr[i]=-1;

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

{

fl1=0;

fl2=0;

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

{

if(fr[i]==p[j])

{

fl1=1;

fl2=1;

break;

}

}

if(fl1==0)

{

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

{

if(fr[i]==-1)

{

fr[i]=p[j];

fl2=1;

break;

}

}

}

if(fl2==0)

{

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

fs[i]=0;

for(k=j-1,l=1;l<=frsize;l++,k--)

{

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

{

if(fr[i]==p[k])

fs[i]=1;

}

}

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

{

if(fs[i]==0)

ind=i;

}

fr[ind]=p[j];

pf++;

}

display();

}

printf("\nNumber of page faults:%d",3+pf);

getch();

}

void display()

{

int i;

printf("\n");

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

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

printf("\n");

}

**OUTPUT:**

Enter the number of frames : 3

Enter the number of elements in the reference string : 20

7

0

1

2

0

3

0

4

2

3

0

3

2

1

2

0

1

7

0

1

7 -1 -1

7 0 -1

7 0 1

2 0 1

2 0 1

2 0 3

2 0 3

4 0 3

4 0 2

4 3 2

0 3 2

0 3 2

0 3 2

1 3 2

1 3 2

1 0 2

1 0 2

1 0 7

1 0 7

1 0 7

Number of page faults : 12

**RESULT:**

Thus the C program to perform **LRU page replacement** was executed

 successfully and the output was verified

**EX:NO:9** **OPTIMAL PAGE REPLACEMENT**

**AIM:**

To write a C program to perform the operation of **optimal page replacement**

**ALGORITHM:**

**STEP 1:**Start the program

**STEP 2:**Get the number of frames and the number of elements in the reference

String

**STEP 3:**Put each and every element into the frame

**STEP 4:**If the frame is full, remove the element from the frame which is least

 recently used from the right side

**STEP 5:**For each and every replacement the page fault is increased by 1

**STEP 6:**Display the page fault

**STEP 7:**Stop the program

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

int fr[3],nf,no;

void main()

{

void display(int);

int p[20],i,j,fs[3],frsize;

int max,f=0,lg[30],ind,r,l[3],k,fl1=0,fl2=2,pf=0;

frsize=3;

clrscr();

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

scanf("%d",&nf);

printf("\nEnter the number of elements in the reference string :");

scanf("%d",&no);

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

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

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

fr[i]=-1;

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

{

fl1=0;

fl2=0;

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

{

if(fr[i]==p[j])

{

fl1=1;

fl2=1;

break;

}

}

if(fl1==0)

{

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

{

if(fr[i]==-1)

{

fr[i]=p[j];

fl2=1;

break;

}

}

}

if(fl2==0)

{

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

l[i]=0;

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

{

for(k=j+1;k<no;k++)

{

if(fr[i]==p[k])

{

l[i]=k-j;

break;

}

}

}

f=0;

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

{

if(l[i]==0)

{

ind=i;

f=1;

break;

}

}

if(f==0)

{

max=l[0];

ind=0;

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

{

if(max<l[i])

{

max=l[i];

ind=i;

}

}

}

fr[ind]=p[j];

pf++;

}

display(nf);

}

printf("The no.of page faults :%d",3+pf);

getch();

}

void display(int nf1)

{

int i;

printf("\n");

for(i=0;i<nf1;i++)

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

printf("\n");

}

**OUTPUT:**

Enter the number of frames: 3

Enter the number of elements in the reference string: 20

7

0

1

2

0

3

0

4

2

3

0

3

2

1

2

0

1

7

0

1

7 -1 -1

7 0 -1

7 0 1

2 0 1

2 0 1

2 0 3

2 0 3

2 4 3

2 4 3

2 4 3

2 0 3

2 0 3

2 0 3

2 0 1

2 0 1

2 0 1

2 0 1

7 0 1

7 0 1

7 0 1

Number of page faults: 9

**RESULT:**

Thus the C program to perform the operation of **optimal page replacement** Was executed successfully and the output was verified

 **SIMULATION OF DISK SCHEDULING ALGORITHMS**

**EX:NO:10a FIRST COME FIRST SERVED**

**AIM:**

To write a program for disk scheduling using FCFS, one of the

responsibilities of operating system is to use hardware efficiently.

**FCFS:**

 In this mode the requests are served in the order in which they

 are received

**ALGORITHM:**

**STEP 1:**Start

**STEP 2:**Declare the variable.

**STEP 3:**Get the choice from the user through the menu and also number

of tracks and track position and curved track position.

**STEP 4:**In case of fcfs,calculate the displacement from current position.

**STEP 5:**Calculate the total head movement.

**STEP 6:** In case of the head arm moves to one extreme position and then

serve call along the ways of other end.

**STEP 7:**In case of the track position are sorted according the seek time .

**STEP 8:**Stop.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void space(int n)

{

int i;

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

{

printf(“--------“);

}

printf(“%d”,n);

}

void main()

{

clrscr();

int sec,i, pos[50],head,seek;

float avg,sum;

printf(“\n\t fcfs scheduling \n enter the no of data to read....:”);

scanf(“%d”,&sec);

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

{

printf(“\n\t enter the section position of data %d.....:”,i);

scanf(“%d”,&head);

clrscr();

seek=abs(pos[i]-head);

sum=seek;

printf(“\n\n FCFS SCHEDULING”);

space(head);

printf(“\n\n\n”);

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

{

space(pos[i]);

if(i==0)

{

seek=abs(pos[i]-head);

printf(“[seek time=%d]\n\n\n”,seek);

}

else

{

seek=abs(pos[i-1]-pos[i]);

sum=sum+seek;

printf(“[seek time=%d]\n\n\n”,seek);

}

}

avg=sum/sec;

printf(“\n\n average seek time=%f”,avg);

getch();

}

**OUTPUT:**

Enter the no of data to be read:5

Enter the sector position of data1:10

Enter the sector position of data2:15

Enter the sector position of data3:20

Enter the sector position of data4:25

Enter the sector position of data5:30

Enter the position of head:10

#  FCFS SCHEDULING

-----------10

-----------10[seek time=9]

-----------15[seek time=5]

-----------20[seek time=5]

-----------25[seek time=5]

-----------30[seek time=5]

Average seek time=5.8000000

**RESULT**:

 Thus the program to perform **FCFS disk scheduling** was executed

and output is verified.

**EX:NO:10b SHORTEST SEEK TIME FIRST**

**AIM:**

To write a program for disk scheduling using SSTF,one of the responsibilities of operating system is to use hardware efficiently.

**SSTF:**

This is a extension FCFS algorithm. The time is the important criteria.it selects the request with minimum seek time and served that process first and continue.

**ALGORITHM:**

**STEP 1:**Start

**STEP 2:**Declare the variable.

**STEP 3:**Get the choice from the user through the menu and also

number of tracks and track position and curved track position.

**STEP 4:**In case of sstf,calculate the displacement from current position.

**STEP 5:**Calculate the total head movement.

**STEP 6:**In case the head arm moves to one extreme position and then serve call

along the ways of other end.

**STEP 7:**The track position are sorted according the seek time .

**STEP 8:**Stop.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void space(int n)

{

int i;

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

{ printf(“--------“);}

printf(“%d”,n);

}

void main()

{

clrscr();

int sec, pos[50],head;

int min,cur,n;

float avg,sum=0;

printf(“\n\t sstf scheduling \n enter the no of data to read....:”);

scanf(“%d”,&sec);

for(int i=1;i<=sec;i++)

{

printf(“\n\t enter the section position of data %d.....:”,i);

scanf(“%d”,&pos[i]);

}

printf(“\n\n enter the position of head....:”);

scanf(“%d”,&head);

clrscr();

printf(“\n SSTF SCHEDULING \n”);

space(head);

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

{ n=i;

min=abs(pos[i]-head);

for(int j=2;j<=sec;j++)

{

if(min>abs(pos[j]-head))

{

min=abs(pos[i]-head);

n=j;

}}

space(pos[n]);

printf(“\n[seek time=%d]\n\n\n”,min);

sum=sum+min;

head=pos[n];

pos[n]=999;

}

avg=sum/sec;

printf(“\n\n average seek time=%f”,avg);

getch();

}

**OUTPUT:**

Enter the no of data to be read:5

Enter the sector position of data1:10

Enter the sector position of data2:15

Enter the sector position of data3:20

Enter the sector position of data4:25

Enter the sector position of data5:30

Enter the position of head:10

#  SSTF SCHEDULING

-----------10

-----------10[seek time=5]

-----------15[seek time=5]

-----------20[seek time=5]

-----------25[seek time=5]

-----------30[seek time=5]

Average seek time=4.0000000

**RESULT**:

 Thus the program to perform **SSTF disk scheduling** was executed

and output is verified.

**EX:NO:10c SCAN SCHEDULING**

**AIM:**

To write a program for disk scheduling using **SCAN scheduling**,one of the

 responsibilities of operating system is to use hardware efficiently.

**SCAN:**

 Here the disk arm starts at one end of the disk and moves towards the other end.

serving request as it reaches each cylinder with it gets to end of disk. At the other

end the direction of movement is reserved

**ALGORITHM:**

**STEP 1:**Start

**STEP 2:**Declare the variable.

**STEP 3:**Get the choice from the user through the menu and also

number of tracks and track position and curved track position

.**STEP 4:**In case of scan, calculate the displacement from current position.

**STEP 5:**Calculate the total head movement.

**STEP 6:**The head arm moves to one extreme position and then

serve calls along the ways of other end.

**STEP 7:** The track position are sorted according the seek time .

**STEP 8:**Stop.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

void space(int n)

{

int i;

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

{

printf(“--------“);

printf(“%d”,n);

}

void main()

{

clrscr();

int sec,i,pos[50],head,seek,j;

float avg;

printf(“\n\t scan scheduling \n enter the no of data to read....:”);

scanf(“%d”,&sec);

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

{

printf(“\n\t enter the section position of data %d.....:”,i-1);

scanf(“%d”,&pos[i]);

if(pos[i]>75||pos[i]<0)

{

 printf(“\n warning||sector position exceeded\n enter the valueb\w 0-75...:\n”);

i--;

}

}

label1:

printf(“\n\n enter the position of head....:”);

scanf(“%d”,&head);

if(head>75|| head<0)

{

printf(“\n warning||sector position exceeded\n enter the valueb\w 0-75...:\n”);

go to label1;

}

pos[0]=0;

pos[1]=head;

sec=sec+1;

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

{

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

{

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

{

int temp;

temp=pos[i];

pos[i]=pos[j];

pos[j]=temp;

}

}

}

int hpos;

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

{

if(pos[i]==head)

{

hpos=i;

}

}

float sum=0;

printf(“\t\t SCAN SCHEDULING”);

for(i=hpos;i>=0;i--)

{

printf(“\n\n\t”);

if(i!=hpos)

{

seek=abs(pos[i]-pos[i+1]);

printf(“\n [seek time=%d]”,seek);

sum = sum + seek;}}

for(i=hpos+1;i<=sec;i++){

printf(“\n\n\n”);

space(pos[i]);

if(i==hpos+1)

{

seek=pos[i];

sum=sum+seek;

printf(“\n [seek time=%d]”,seek);}}

avg=sum/(sec-1);

printf(“\n\n average seek time=%f”,avg);

getch();

}

**OUTPUT:**

#  SCAN SCHEDULING

Enter the no of process: 5

Enter the sector position of data1.....:20

Enter the sector position of data2.....:12

Enter the sector position of data3.....:34

Enter the sector position of data4.....:25

Enter the sector position of data5.....:03

Enter the position of record......:18`

[seek time=6]

[seek time =9]

[seek time=3]

[seek time=0]

---------------25[seek time=25]

-----------------34[seek time=9]

Average seek time=10.400000

**RESULT**:

 Thus the program to perform **SCAN disk scheduling** was executed

and output is verified.

**EX:NO:10d C-SCAN SCHEDULING**

**AIM:**

To write a program for disk scheduling using **C- SCAN**, one of the responsibilities of operating system is to use hardware efficiently.

**C-SCAN:**

In this method the head moves from one end of the disk to other end serving

request along the difference with scan technique is while it served its direction its deos not serve any request. Here the head comes to first position.

**ALGORITHM:**

**STEP 1:**Start

**STEP 2:**Declare the variable.

**STEP 3:**Get the choice from the user through the menu and also number

of tracks and track position and curved track position.

**STEP 4:**In case of c-scan,calculate the displacement from current position.

**STEP 5:**Calculate the total head movement.

**STEP 6:**In case of c-scan, the head arm moves to one extreme position and then

serve call along the ways of other end.

**STEP 7:**The track positions are sorted according the seek time .

**STEP 8:**Stop.

**PROGRAM**:

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void space(int n)

{int i;

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

{

printf(“-“);

}

printf(“%d”,n);

}void main()

{

int sec,i,j,pos[50],head,seek;

float avg;

printf(“\n \t \t C-SCAN DISK SCHEDULING \t\t\n”);

printf(“\n Enter the number of datas to be read:”);

scanf(“%d”,&sec);

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

{

printf(“\n Enter the sector position of data %d:”,i);

scanf(“%d”,&pos[i]);

if(pos[i]>75!!pos[i]<0)

{

printf(“\n Warning !! sector position execeeded \n enter the value between 0-75 \n”);

i--;}

}

label1:

printf(“\n Enter the position of head:”);

scanf(“%d”,&head);

if(head>75!!&head<0)

{printf(“\n Warning !! sector position exceeded \n enter the value between 0-75 \n”);

goto label1;}

Sec=sec+2;

pos[0]=head;

pos[1]=head;

pos[sec]=75;

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

{

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

{

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

{

int temp;

temp=pos[i];

pos[i]=pos[j];

pos[j]=temp;

}

}

}

int npos;

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

{

if(pos[i]==head)

{

npos=i;

}

}

float sum=0;

printf(“\n \t \t C- SCAN DISK SCHEDULING \t\t\n”);

for(i=npos;i<=sec;i++)

{

printf(“\n\n\n”);

space(pos[i]);

if(i!=npos)

{

seek=abs(pos[i]-pos[i-1]);

printf(“\n [seek time=%d]”,seek);

sum=sum+seek;

}

}

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

{

printf(“\n”);

space(pos[i]);

if(i==0)

{

seek=abs(pos[sec]-pos[0]);

sum=sum+seek;

printf(“\n [seek time=%d]”,seek);

}

else

{

seek=abs(pos[i]-pos[i-1]);

sum=sum+seek;

printf(“\n [seek time=%d]”,seek);

}

}

avg=sum/(sec-1);

printf(“\n Average seek time=%f”,avg);

}

**OUTPUT:**

enter the number of data to be read:5

enter the sector position of data1:12

enter the sector position of data2:30

enter the sector position of data3:25

enter the sector position of data4:20

enter the sector position of data5:8

enter the position of head:22

 C-SCAN DISK SCHEDULING

------22

------25(seek time=3)

------30(seek time=5)

------75(seek time=45)

------8(seek time=8)

------20(seek time=12)

Average seek time=24.666

**RESULT**:

 Thus the program to perform **C-SCAN disk scheduling** was executed

and output is verified.

**EX:NO:10e C-LOOK SCHEDULING**

**AIM:**

To write a program for disk scheduling using **C-LOOK**,One of the responsibilities of operating system is to use hardware efficiently.

**C-LOOK:**

Here the disk arm starts at one end of the disk and moves towards the other end serving request as it reaches each cylinder with it served its direction its does not serve any request. Till it reaches the final given disk position.

**ALGORITHM:**

**STEP 1:**Start

**STEP 2:**Declare the variable.

**STEP 3:**Get the choice from the user through the menu and also number

of tracks and track position and curved track position.

**STEP 4:**In case ofc-look,calculate the displacement from current position.

**STEP 5:**Calculate the total head movement.

**STEP 6:**In case of the head arm moves to one extreme position and then serve call

along the ways of other end.

**STEP 7:**In case of the track position are sorted according the seek time .

**STEP 8:**Stop.

**PROGRAM**:

#include<stdio.h>

#include<stdlib.h>

void space(int n)

{

int i;

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

{

printf(“--------“);

printf(“%d”,n);

}

void main()

{

clrscr();

int sec,i,pos[50],head,seek;

float avg;

printf(“\n\t enter the no of data to read....:”);

scanf(“%d”,&sec);

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

{

printf(“\n\t enter the section position of data %d.....:”,i);

scanf(“%d”,&pos[i]);

if(pos[i]>75||pos[i]<0)

{

 printf(“\n warning||sector position exceeded\n enter the valueb\w 0-75...:\n”);

i--;

}

}

label1:

printf(“\n\n enter the position of head....:”);

scanf(“%d”,&head);

printf(“\n warning||sector position exceeded\n enter the valueb\w 0-75...:\n”);

go to label1;

pos[0]=head;

clrscr();

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

{

for(int j=i+1;j<=sec;j++)

{

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

{

int temp;

temp=pos[i];

pos[i]=pos[j];

pos[j]=temp;

}

}

}

int npos;

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

{

if(pos[i]=head)

{

npos=i;

}

}

float num=0;

printf(“\t\t c look scheduling”);

for(i=npos;i<=sec;i++)

{

printf(“\n\n\n”);

space(pos[i]);

if(i!=npos)

{

seek=abs(pos[i]-pos[i-1]);

printf(“\n [seek time=%d]”,seek);

sum=sum+seek;}}

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

printf(“\n\n\n”);

space(pos[i]);

if(i==0){

seek=abs(pos[sec]-pos[0]);

sum=sum+seek;

printf(“\n [seek time=%d]”,seek);}

else{

seek=abs(pos[i]-pos[i-1]);

sum=sum+seek;

printf(“\n [seek time=%d]”,seek);}}

avg=sum/(sec-1);

printf(“\n\n average seek time=%f”,avg);

getch();

}

**OUTPUT:**

#

Enter the no of process: 5

Enter the sector position of data1.....:12

Enter the sector position of data2.....:35

Enter the sector position of data3.....:20

Enter the sector position of data4.....:08

Enter the sector position of data5.....:25

Enter the position of record......:18

C LOOK SCHEDULING

----------18

-----------20[seek time =2]

------------25[seek time=5]

-------------35[seek time=10]

---------------8[seek time=27]

-----------------12[seek time=4]

Average seek time=12.000000

**RESULT**:

 Thus the program to perform **C-LOOK disk scheduling** was executed

and output is verified.

**EX: NO: 11 SIMULATION OF FILE SYSTEMS**

**AIM:**

 To write a C program for performing simulation of files systems.

**ALGORITHM:**

**STEP1:** Start the program.

**STEP2:**Declare the needed variables.

**STEP3:**Create a file with extension

**STEP4:** Add an attributes in the created file.

**STEP5:** Perform write, read, delete operations in the created file using switch case.

**STEP6:** Perform sequencial or random access methods in the file using switch case.

**STEP7:** Displaythe start and end position for read or write operation.

**STEP8:**Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char fname[20], str[100], ch=0, ch1;

FILE \*fp;

int handle;

int m, pos2, k, pos1;

void fcreate()

{

 printf("\n\nEnter the File Name with extension: ");

 scanf("%s", fname);

 fp = fopen(fname, "w");

 fflush(stdin);

 printf("\nEnter the Data:\n");

 gets(str);

 fprintf(fp, "%s", str);

 handle = fileno(fp);

int len=strlen(fname);

 int flag=0;

 printf("\n File created Sucessfully\n\n");

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

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

{

if(fname[i]=='.')

{

flag = i;

}

}

printf("\nFile Name :");

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

{

 printf("%c", fname[i]);

}

printf("\nFile Type :");

for(i=flag+1;i<=len;i++)

{

 printf("%c", fname[i]);

}

printf("\nFile Identifier:%d", handle);

fclose(fp);

}

void fwrite()

{

fp=fopen(fname, "a");

printf("\nFile Opened for writing:\n");

fflush(stdin);

printf("\nEnter the Data:\n");

gets(str);

fprintf(fp, "%s", str);

fclose(fp);

}

void fread()

{

fp=fopen(fname, "r");

printf("\nEnter your choice of Access Methods:\n\t1: Sequential \n\t2: Random :");

scanf("%d", &m);

 if(m==1)

{

printf("\n Enter the Position till you want to read:");

scanf("%d", &pos2);

rewind(fp);

while(pos2!=ftell(fp))

{

char c=getc(fp);

printf("%c", c);

}

}

else if(m==2)

{

 printf("\n Enter the Start Position you want to read:");

scanf("%d", &pos1);

fseek(fp,pos1, 0);

printf("\n Enter the Position till you want to read:");

scanf("%d", &pos2);

while(pos2!=ftell(fp))

{

char c=getc(fp);

printf("%c", c);

}

}

fclose(fp);

}

void delfile()

{

if (remove(fname) == 0)

{

printf("Removed %s.\n",fname);

}

else

{

 perror("remove");

}

}

void main()

{

clrscr();

printf("\n\tFile Creation");

fcreate();

getch();

clrscr();

do

{

if(ch==3)

{

fcreate();

}

printf("\n\nFile Operations\n\n");

fflush(stdin);

printf("\n\n1:Write Data\n2:Read Data\n3:Delete File");

printf("\nEnter your Choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

fwrite();

break;

case 2:

fread();

break;

case 3:

delfile();

break;

default:

printf("\nWrng Choice");

}

fflush(stdin);

printf("\nDo you Want to Continue(y/n):");

scanf("%c", &ch1);

}while(ch1=='y');

getch();

}

**OUTPUT:**

File Creation

Enter the File Name with extension: file1.txt

Enter the Data:Hello world

 File created Sucessfully

File Attributes

File Name :file1

File Type :txt

File Identifier:5

File Operations

1:Write Data

2:Read Data

3:Delete File

Enter your Choice: 1

File Opened for writing:

Enter the Data: How are you

Do you Want to Continue(y/n):y

File Operations

1:Write Data

2:Read Data

3:Delete File

Enter your Choice: 2

Enter your choice of Access Methods:

 1: Sequential

 2: Random :2

 Enter the Start Position you want to read:5

 Enter the Position till you want to read:10

 worldhow

Do you Want to Continue(y/n):y

File Operations

1:Write Data

2:Read Data

3:Delete File

Enter your Choice:3

Removed file1.txt.

Do you Want to Continue(y/n):n

**RESULT:**

Thus the C program to **simulate the system files** was executed and the

system files was executed and the output was verified