



SRM VALLIAMMAI ENGINEERING
COLLEGE



SRM Nagar, Kattankulathur-603203.

Lab Manual

IT3464 OPERATING SYSTEMS LABORATORY

(IV semester)

PREPARED BY

G.SANGEETHA ,ASSISTANT PROFESSOR /CSE

IT3464 OPERATING SYSTEMS LABORATORY**L T P C****0 0 3 1.5****OBJECTIVES:**

- To understand the basics of Unix command and shell programming.
- To implement various CPU scheduling algorithms.
- To implement Deadlock Avoidance and Deadlock Detection Algorithms
- To implement Page Replacement Algorithms
- To implement various memory allocation methods and File Organization, File Allocation Strategies.

LIST OF EXPERIMENTS:

1. UNIX commands and Basic Shell Programming
2. Process Management using System Calls : Fork, Exit, Getpid, Wait, Close
3. Write C programs to implement the various CPU Scheduling Algorithms
4. Implement mutual exclusion by Semaphore
5. Write C programs to avoid Deadlock using Banker's Algorithm
6. Write a C program to Implement Deadlock Detection Algorithm
7. Write C program to implement Threading
8. Write C program to Implement the paging Technique.
9. Write C programs to implement the following Memory Allocation Methods
 - a. First Fit
 - b. Worst Fit
 - c. Best Fit
- 10 Write C programs to implement the various Page Replacement Algorithms
11. Write C programs to Implement the various File Organization Techniques
12. Implement the following File Allocation Strategies using C programs
 - a. Sequential
 - b. Indexed
 - c. Linked
13. Write C programs for the implementation of various disk scheduling algorithms

TOTAL: 45 PERIODS

Software Requirements:

Standalone desktops with C / C++ / Java / Equivalent compiler 30 Nos.

(or) Server with C / C++ / Java / Equivalent compiler supporting 30 terminals or more

COURSE OUTCOMES:

At the end of this course, the students will be able to:

- Define and implement UNIX Commands.
- Compare the performance of various CPU Scheduling Algorithms.
- Compare and contrast various Memory Allocation Methods.
- Define File Organization and File Allocation Strategies.
- Implement various Disk Scheduling Algorithms

CO – PO – PSO Mapping

CO	PO												PSO			
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4
1	3	1	3	1	1	-	-	-	-	-	-	-	-	2	-	-
2	3	1	1	2	2	-	-	-	-	-	-	-	-	2	-	-
3	3	3	2	1	2	-	-	-	-	-	-	-	-	2	-	-
4	1	2	2	3	2	-	-	-	-	-	-	-	-	2	-	-
5	2	2	1	1	3	-	-	-	-	-	-	-	-	2	-	-
Avg	2.4	1.8	1.8	1.6	2.0	-	-	-	-	-	-	-	-	2.0	-	-

EX NO: 1

BASICS OF UNIX COMMANDS

Aim

To study and execute Unix commands.

Unix is security conscious, and can be used only by those persons who have an account.

Telnet (Telephone Network) is a Terminal emulator program for TCP/IP networks that enables users to log on to remote servers.

To logon, type **telnet** server_ipaddress in **run** window.

User has to authenticate himself by providing username and password. Once verified, a greeting and \$ prompt appears. The shell is now ready to receive commands from the user.

Options suffixed with a hyphen (-) and arguments are separated by space.

General commands

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

Directory commands

Command	Function
pwd	Path of the present working directory

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

File commands

Command	Function
cat >filename	To create a file with some contents. To end typing press Ctrl+d . The >symbol means redirecting output to a file. (<for input)
cat filename	Displays the file contents.
cat >>filename	Used to append contents to a file
cp src des	Copy files to given location. If already exists, it will be overwritten
cp -i src des	Warns the user prior to overwriting the destination file
cp -r src des	Copies the entire directory, all its sub-directories and files.
mv old new	To rename an existing file or directory. -i option can also be used
mv f1 f2 f3 dir	To move a group of files to a directory.
mv -v old new	Display name of each file as it is moved.
rmfile	Used to delete a file or group of files. -i option can also be used
rm *	To delete all the files in the directory.
rm -r *	Deletes all files and sub-directories
rm -f *	To forcibly remove even write-protected files
ls	Lists all files and subdirectories (blue colored) in sorted manner.
lsname	To check whether a file or directory exists.
lsname*	Short-hand notation to list out filenames of a specific pattern.
ls -a	Lists all files including hidden files (files beginning with .)
ls -x dirname	To have specific listing of a directory.
ls -R	Recursive listing of all files in the subdirectories
ls -l	Long listing showing file access rights (read/write/execute- rw x for user/group/others- ugo).
cmpfile1 file2	Used to compare two files. Displays nothing if files are identical.
wcfile	It produces a statistics of lines (l), words(w), and characters(c).
chmodperm file	Changes permission for the specified file. (r=4, w=2, x=1) chmod 740 file sets all rights for user, read only for groups and no rights for others

Output:**GENERAL COMMANDS**

```
[student@vecit ~]$ date
Mon Dec 15 10:12:47 AQT 2014
[student@vecit ~]$ date +%D
12/15/14
[student@vecit ~]$ date +%T
10:13:11
[student@vecit ~]$ date +%Y
2014
[student@vecit ~]$ date +%H
10
```

```
[sandhya@vecit ~]$ cal
December 2014
Su Mo Tu We Th Fr Sa
  1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

```
[sandhya@vecit ~]$ cal 2014
```

```
2014
```

```

January          February          March
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
19 20 21 22 23 24 25 26 27 28 29 30 31
26 27 28 29 30 31

April           May           June
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
20 21 22 23 24 25 26 27 28 29 30 31
27 28 29 30 31

July           August           September
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
20 21 22 23 24 25 26 27 28 29 30 31
27 28 29 30 31

October        November        December
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
19 20 21 22 23 24 25 26 27 28 29 30 31
26 27 28 29 30 31
```

```
[sandhya@vecit ~]$ cal 5 2014
```

```
May 2014
Su Mo Tu We Th Fr Sa
  4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

```
[student@vecit ~]$ who
```

```
studentpts/1    Dec 15 10:05 (172.16.1.14)
```

```
[student@vecit ~]$ who am i
```

```
studentpts/1    Dec 15 10:05 (172.16.1.14)
```

```
[student@vecit ~]$ tty
```

```
/dev/pts/1
```

```
[student@vecit ~]$ uname
```

```
Linux
```

```
[student@vecit ~]$ echo "hello"
```

```
hello
```

```
[student@vecit ~]$ echo $HOME
```

```
/home/student
```

```
[student@vecit ~]$ bc
```

```
bc 1.06
```

```
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
```

```
This is free software with ABSOLUTELY NO WARRANTY.
```

```
For details type `warranty'.
```

```
[student@vecit ~]$ man lp
```

```
lp(1)                Easy Software Products                lp(1)
```

NAME

lp - print files

cancel - cancel jobs

SYNOPSIS

```
lp [ -E ] [ -c ] [ -d destination ] [ -h server ] [ -m ] [ -n num-
copies [ -o option ] [ -q priority ] [ -s ] [ -t title ] [ -H handling
] [ -P page-list ] [ file(s) ]
```

```
lp [ -E ] [ -c ] [ -h server ] [ -i job-id ] [ -n num-copies [ -o
option ] [ -q priority ] [ -t title ] [ -H handling ] [ -P page-list ]
cancel [ -a ] [ -h server ] [ -u username ] [ id ] [ destination ] [
destination-id ]
```

DESCRIPTION

lp submits files for printing or alters a pending job. Use a filename of "-" to force printing from the standard input.

cancel cancels existing print jobs. The -a option will remove all jobs from the specified destination.

OPTIONS

The following options are recognized by lp:

```
[student@vecit ~]$ history
```

```
1 ls
```

```
2 date
```

```
3 date +%D
```

```
4 date +%T
```

```
5 date +%Y
```

```
6 date +%H
```

```
7 cal
```

```
8 cal 2014
```

```
9 cal 5 2014
10 who
11 who am i
12 tty
13 uname
14 uname -r
15 uname -n
16 echo "hi"
17 echo $HOME
18 bc
19 manlp
20 history
21 PWD
22 pwd
23 mkdir san
24 mkdir san s
25 mkdir s1 s2
26 ls
27 cd s1
28 cd
29 cd /
30 cd..
31 cd . .
32 rmdir s
33 rmdir s1
34 ls
35 cd
36 ls
37 rmdir s1
38 ls
39 CAT>TEST
40 cat>test
41 cat test
42 cat>>test
43 cat test
44 cat test1
45 cat>test1
46 cp test test1
47 cat test1
48 exit
49 vi
50 vi swap.sh
51 sh swap
52 cc swap.sh
53 cc swap
54 sh swap.sh
55 vi swap.sh
56 sh swap.sh
57 vi swap.sh
```



```
58 sh swap.sh
59 vi temp.sh
60 sh temp.sh
61 vi temp.sh
62 sh temp.sh
63 vi temp.sh
64 sh temp.sh
65 vi temp.sh
66 sh temp.sh
67 exit
68 date
69 date +%D
70 date +%D
71 date +%T
72 date +% Y
73 exit
74 date
75 date +%D
76 date +%T
77 date +%Y
78 date +%H
79 cal
80 cal 2014
81 cal may 2014
82 cal 5 2014
83 who
84 who am i
85 tty
86 uname
87 echo "hello"
88 echo $HOME
89 bc
90 lp swap
91 manlp
92 history
```

DIRECTORY COMMANDS

```
[student@vecit ~]$ pwd
/home/student
[student@vecit ~]$ mkdir san
[student@vecit ~]$ mkdir s1 s2
[student@vecit ~]$ ls
s s1 s2 san
[student@vecit ~]$ cd s1
[student@vecit s1]$ cd
[student@vecit ~]$ cd /
[student@vecit /]$ cd ..
[student@vecit /]$ rmdir s1
```

```
[student@vecit ~]$ ls
s s2 san
[student@vecit ~]$
```

FILE COMMANDS

```
[student@vecit ~]$ cat>test
hi welcome operating systems lab
[student@vecit ~]$ cat test
hi welcome operating systems lab
[student@vecit ~]$ cat>>test
fourth semester
[student@vecit ~]$ cat test
hi welcome operating systems lab fourth semester
[student@vecit ~]$ cat>test1
[student@vecit ~]$ cp test test1
[student@vecit ~]$ cat test1
hi welcome operating systems lab fourth semester
[student@vecit ~]$ cp -i test test1
cp: overwrite `test1'? y
[student@vecit ~]$ cp -r test test1
[student@vecit ~]$ ls
s s2 san swap.sh temp.sh test TEST test1
[student@vecit ~]$ mv san san1
[student@vecit ~]$ ls
s s2 san1 swap.sh temp.sh test TEST test1
[student@vecit ~]$ mv test test1 san1
[student@vecit ~]$ mv -v san1 sannew
`san1' -> `sannew'
[student@vecit ~]$ ls
s s2sannew swap.sh temp.sh TEST
[student@vecit ~]$ cmp test test1
cmp: test: No such file or directory
```

Result

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

Viva Questions

1. What is the use of cat commands?
2. Define Operating Systems?
3. What is the use of filter/grep/pipe commands?
4. How is unix different from windows
5. What is unix?
6. What is the file structure of unix?
7. What is a kernel?
8. What is the difference between multi-user and multi-tasking?
9. Differentiate relative path from absolute path.
10. What are the differences among a system call, a library function, and a UNIX command?

Ex. No: 1A

SIMPLE SHELL PROGRAMS

AIM

To write simple shell scripts using shell programming fundamentals.

The activities of a shell are not restricted to command interpretation alone. The shell also has Rudimentary programming features. When a group of commands has to be executed regularly, they are stored in a file (with extension **.sh**). All such files are called shell scripts or shell programs. Shell programs run in interpretive mode.

The original UNIX came with the Bourne shell (**sh**) and it is universal even today. Then came a plethora of shells offering new features. Two of them, C shell (**cs**) and Korn shell (**ks**) has been well accepted by the UNIX fraternity. Linux offers Bash shell (**ba**) as a superior alternative to Bourne shell.

Preliminaries

1. Comments in shell script start with **#**. It can be placed anywhere in a line; the shell ignores contents to its right. Comments are recommended but not mandatory
2. Shell variables are loosely typed i.e. not declared. Their type depends on the value assigned. Variables when used in an expression or output must be prefixed by **\$**.
3. The **read** statement is shell's internal tool for making scripts interactive.
4. Output is displayed using **echo** statement. Any text should be within quotes. Escape sequence should be used with **-e** option.
5. Commands are always enclosed with **` `** (back quotes).
6. Expressions are computed using the **expr** command. Arithmetic operators are + - * / %. Meta characters * () should be escaped with a ****.
7. Multiple statements can be written in a single line separated by **;**
8. The shell scripts are executed using the **sh** command (shfilename).

Swapping values of two variables

Algorithm

- Step 1 : Start
- Step 2 : Read the values of a and b
- Step 3 : Interchange the values of a and b using another variable t as follows:
 t = a
 a = b
 b = t
- Step 4 : Print a and b
- Step 5 : Stop

Program (swap.sh)

```
# Swapping values
echo -n "Enter value for A : "
read a
echo -n "Enter value for B : "
read b
t=$a
a=$b
b=$t
echo "Values after Swapping"
echo "A Value is $a"
echo "B Value is $b"
```

Output

```
[student@vecit ~]$ sh swap.sh
Enter Value for A:5
Enter Value for B:6
Values after Swapping
A value is 6
B values is 5
[student@vecit ~]$
```

Fahrenheit to Centigrade Conversion**Algorithm**

Step 1 : Start
 Step 2 : Read fahrenheitvalue
 Step 3 : Convert fahrenheitto centigrade using the formulae: $(\text{fahrenheit} - 32) \times 5/9$
 Step 4 : Print centigrade
 Step 5 : Stop

Program

```
# Degree conversion
echo -n "Enter Fahrenheit : "
read f
c=`expr \( $f - 32 \) \* 5 / 9`
echo "Centigrade is : $c"
```

Output

```
[student@vecit ~]$ sh temp.sh
Enter Fahrenheit:4
Centrigrade is: -15
[student@vecit ~]$
```

Result

Thus using programming basics, simple shell scripts were executed

Ex no 1B**CONDITIONAL CONSTRUCTS****Aim**

To write shell scripts using decision-making constructs.

Shell supports decision-making using **if** statement. The **if** statement like its counterpart in programming languages has the following formats. The first construct executes the statements when the condition is true. The second construct adds an optional **else** to the first one that has different set of statements to be executed depending on whether the condition is true or false. The last one is an **elif** ladder, in which conditions are tested in sequence, but only one set of statements is executed.

<pre>if [condition] then statements fi</pre>	<pre>if [condition] then statements else statements fi</pre>	<pre>if [condition] then statements elif [condition] then statements . . . else statements fi</pre>
--	--	--

The set of relational and logical operators used in conditional expression is given below. The numeric comparison in the shell is confined to integer values only.

Operator	Description
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal to
-a	Logical AND
-o	Logical OR
!	Logical NOT

Odd or even**Algorithm**

Step 1 : Start

Step 2 : Read number

Step 3 : If number divisible by 2 then

Print "Number is Even"

Step 3.1 : else

Print "Number is Odd"

Step 4 : Stop

Program

```
# Odd or even using if-else
echo -n "Enter a non-zero number : "
read num
rem=`expr $num % 2`
if [ $rem -eq 0 ]
then
echo "$num is Even"
else
echo "$num is Odd"
fi
```

Output

```
[student@vecit ~]$ sh oddeven.sh
```

```
Enter a non-zero number : 12
```

```
12 is Even
```

String comparison**Algorithm**

Step 1 : Start

Step 2 : Read strings str1 and str2

Step 3 : If str1 = str2 then

Print "Strings are the same"

Step 3.1 : else

Print "Strings are distinct"

Step 4 : Stop

Program

```
echo -n "Enter the first string : "
read s1
echo -n "Enter the second string : "
read s2
if [ $s1 == $s2 ]
then
echo "Strings are the same"
else
echo "Strings are distinct"
fi
```

Output

```
[student@vecit ~]$ sh strcomp.sh
Enter the first string :ece-a
Enter the second string : ECE-A
Strings are distinct
```

Result

Thus using if statement scripts with conditional expressions were executed

Ex No 1C**MULTI-WAY BRANCHING****Aim**

To write shell scripts using case construct to match patterns.

The `case` statement is used to compare a variables value against a set of constants (integer, character, string, range). If it matches a constant, then the set of statements followed after `)` is executed till a `;;` is encountered. The optional default block is indicated by `*`. Multiple constants can be specified in a single pattern separated by `|`.

```
casevariablein
constant1)
statements ;;
constant2)
statements ;;
...
constantN)
statements ;;
*)
statements
esac
```

Simple Calculator**Algorithm**

```
Step 1 : Start
Step 2 : Read operands a and b
Step 3 : Display operation menu
Step 4 : Read option
Step 5 : If option = 1 then
Calculate c = a + b
Step 5.1 : else if option = 2 then
Calculate c = a - b
Step 5.2 : else if option = 3 then
Calculate c = a * b
Step 5.3 : else if option = 4 then
Calculate c = a / b
Step 5.4 : else if option = 5 then
```

Calculate $c = a \% b$
 Step 5.5 : else
 Print "Invalid option"
 Step 6 : Print c
 Step 7 : Stop

Program

```
# Arithmetic operations--multiple statements in a block
echo -n "Enter the two numbers : "
read a b
echo " 1. Addition"
echo " 2. Subtraction"
echo " 3. Multiplication"
echo " 4. Division"
echo " 5. Modulo Division"
echo -n "Enter the option : "
read option
case $option in
1) c=`expr $a + $b`
echo "$a + $b = $c";;
2) c=`expr $a - $b`
echo "$a - $b = $c";;
3) c=`expr $a \* $b`
echo "$a * $b = $c";;
4) c=`expr $a / $b`
echo "$a / $b = $c";;
5) c=`expr $a % $b`
echo "$a % $b = $c";;
*) echo "Invalid Option"
esac
```

Output

```
[student@vecit ~]$ shsimplecal.sh
Enter the two numbers : 2 4
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Modulo Division
Enter the option : 1
2 + 4 = 6
```

Result

Thus using case statement, shell scripts were executed

Ex No 1D**LOOPING****Aim**

To write shell scripts using looping statements.

Shell supports a set of loops such as **for**, **while** and **until** to execute a set of statements repeatedly. The body of the loop is contained between **do** and **done** statement.

The **for** loop doesn't test a condition, but uses a list instead.

```
for variable in list
do
statements
done
```

The **while** loop executes the statements as long as the condition remains true.

```
while [ condition ]
do
statements
done
```

The **until** loop complements the while construct in the sense that the statements are executed as long as the condition remains false.

```
until [ condition ]
do
statements
done
```

Armstrong Number**Algorithm**

Step 1 : Start

Step 2 : Read number

Step 3 : Initialize 0 to sum and number to num

Step 4 : Extract last digit by computing number modulo 10

Step 5 : Cube the last digit and add it to sum

Step 6 : Divide number by 10

Step 7: Repeat steps 4–6 until number > 0

Step 8 : If sum = number then

Print "Armstrong number"

Step 8.1 : else

Print "Not an Armstrong number"

Step 9 : Stop

Program (armstrong.sh)

```
# Armstrong number using while loop
echo -n "Enter a number : "
read n
a=$n
s=0
while [ $n -gt 0 ]
```

```
do
r=`expr $n % 10`
s=`expr $s + \( $r \* $r \* $r \)`
n=`expr $n / 10`
done
if [ $a -eq $s ]
then
echo "Armstrong Number"
else
echo -n "Not an Armstrong number"
fi
```

Output

```
[student@vecit ~]$ sh armstrong.sh
```

```
Enter a number : 370
```

```
Armstrong Number
```

Result

Thus using loops, iterative scripts were executed

Viva Questions

1. What is Shell?
2. What are some common shells and what are their indicators?
3. Briefly describe the Shell's responsibilities
4. What are shell variables?
5. What is Bash Shell?
6. Differentiate cat command from more command.
7. What does this command do? cat food 1 > kitty
8. What's the conditional statement in shell scripting?
9. How do you do number comparison in shell scripts?
10. How do you define a function in a shell script?

Ex No: 3A

IMPLEMENTATION OF ROUND ROBIN SCHEDULING ALGORITHM

Aim

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

Algorithm:

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

Program:

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
intct=0,y[30],j=0,bt[10],cwt=0;
inttq,i,max=0,n,wt[10],t[10],at[10],tt[10],b[10];
float a=0.0,s=0.0;
char p[10][10];
clrscr();
printf("\n enter the no of process:");
scanf("%d",&n);
printf("\nenter the time quantum");
scanf("%d",&tq);
printf("\nenter the process name,bursttime,arrival time");
for(i=0;i<n;i++)
{
scanf("%s",p[i]);
scanf("%d",&bt[i]);
scanf("%d",&at[i]);
wt[i]=t[i]=0;
b[i]=bt[i];
}
printf("\n\t\tGANTT CHART");
printf("\n-----\n");
for(i=0;i<n;i++)
{
if(max<bt[i])
max=bt[i];
}
while(max!=0)
```

```

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

if(bt[i]>0)
{
if(ct==0)
wt[i]=wt[i]+cwt;
else
wt[i]=wt[i]+(cwt-t[i]);
}
if(bt[i]==0)
cwt=cwt+0;
else if(bt[i]==max)
{
if(bt[i]>tq)
{
cwt=cwt+tq;
bt[i]=bt[i]-tq;
max=max-tq;
}
else
{
cwt=cwt+bt[i];
bt[i]=0;
max=0;
}
printf("\t%s",p[i]);
y[j]=cwt;
j++;
}
else if(bt[i]<tq)
{
cwt=cwt+bt[i];
bt[i]=0;
printf("\t%s",p[i]);
y[j]=cwt;
j++;
}
else if(bt[i]>tq)
{
cwt=cwt+tq;
bt[i]=bt[i]-tq;
printf("\t%s",p[i]);
y[j]=cwt;
j++;
}
}
}

```

```

}
else if(bt[i]==tq)
{
cwt=cwt+bt[i];
printf("\t%s",p[i]);
bt[i]=0;
y[j]=cwt;
j++;
}
t[i]=cwt;
}
ct=ct+1;
}
for(i=0;i<n;i++)
{
wt[i]=wt[i]-at[i];
a=a+wt[i];
tt[i]=wt[i]+b[i];
s=s+tt[i];
}
a=a/n;
s=s/n;
printf("\n-----");
printf("\n0");
for(i=0;i<j;i++)
printf("\t%d",y[i]);
printf("\n");
printf("\n-----");
printf("\n\t\tROUND ROBIN\n");
printf("\n Process Burst-time Arrival-time Waiting-time Turnaround-time\n");
for(i=0;i<n;i++)
printf("\n\n %d%s \t %d\t\t %d \t\t %d\t\t %d",i+1,p[i],b[i],at[i],wt[i],tt[i]);
printf("\n\nAvg waiting time=%f",a);
printf("\n\nAvgturn around time=%f",s);
getch();
}

```

Output:

```

enter the no of process:3
enter the time quantum2
enter the process name,bursttime,arrival time
p1 2 0
p2 3 1
p3 4 2

```

GANTT CHART

```

-----
|  p1|  p2|  p3|  p2|  p3
-----
0   2   4   6   7   9
-----

```

ROUND ROBIN

Process	Burst-time	Arrival-time	Waiting-time	Turnaround-time
1p1	2	0	0	2
2p2	3	1	3	5
3p3	4	2	3	5

Avg Waiting Time=2.000000
 Avg Turnaround Time=4.000000

Result

The Round Robin scheduling algorithm has been implemented in C.

Ex No: 3B**IMPLEMENTATION OF SJF SCHEDULING ALGORITHM****Aim**

To write a C program to implement shortest job first (non-pre-emptive) scheduling algorithm.

Description:

Assume the next burst time of each process is known. SJF selects process which has the shortest burst time. This is an Optimal algorithm because it has the shortest average waiting time, however it is impossible to know in advance. OS knows the past burst times - make a prediction using an average. It can be either nonpreemptive or preemptive scheduling.. It processes shortest-remaining-time-first. The process interrupts running process if a new process enters the queue. The new process must have shorter burst than remaining time.

Algorithm:

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on burst time.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
int i,j,n,bt[30],at[30],st[30],ft[30],wat[30],wt[30],temp,temp1,tot,tt[30];
floatawt,att;
int p[15];
clrscr();
wat[1]=0;
printf("ENTER THE NO.OF PROCESS");
scanf("%d",&n);
printf("\nENTER THE PROCESS NUMBER,BURST TIME AND ARRIVAL TIME");
for(i=1;i<=n;i++)
{
scanf("%d\t %d\t %d",&p[i],&bt[i],&at[i]);
}
printf("\nPROCESS\tBURSTTIME\tARRIVALTIME");
for(i=1;i<=n;i++)
{
printf("\np%d\t%d\t%d",p[i],bt[i],at[i]);
}
for(i=1;i<=n;i++)
```

```

{
for(j=i+1;j<=n;j++)
{
if(bt[i]>bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
temp1=p[i];
p[i]=p[j];
p[j]=temp1;
}
}
if(i==1)
{
st[1]=0;
ft[1]=bt[1];
wt[1]=0;
}
else
{
st[i]=ft[i-1];
ft[i]=st[i]+bt[i];
wt[i]=st[i];
}
}
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n-----\n");
for(i=1;i<=n;i++)
printf("\tp%d\t",p[i]);
printf("\t\n");
printf("\n-----\n");
printf("\n");
for(i=1;i<=n;i++)
printf("%d \t\t",wt[i]);
printf("%d",wt[n]+bt[n]);
printf("\n-----\n");
for(i=2;i<=n;i++)
wat[i]=wt[i]-at[i];
for(i=1;i<=n;i++)
tt[i]=wat[i]+bt[i];
printf("\nPROCESS\tBURSTTIME\tARRIVALTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=1;i<=n;i++)
{
printf("\np%d %5d %15d %15d %15d",p[i],bt[i],at[i],wat[i],tt[i]);

```



```

}
for(i=1,tot=0;i<=n;i++)
tot+=wt[i];
awt=(float)tot/n;
printf("\n\n AVERAGE WAITING TIME=%f",awt);
for(i=1,tot=0;i<=n;i++)
tot+=tt[i];
att=(float)tot/n;
printf("\n\n AVERAGE TURNAROUND TIME=%f",att);
getch();
}

```

Output:

```

enter the no.of process3
enter the process number,burst time and arrival time1 8 1
2 5 1
3 3 1

```

PROCESS	BURSTTIME	ARRIVALTIME
p1	8	1
p2	5	1
p3	3	1

GANTT CHART

```

-----
|  p3  |  p2  |  p1  |
-----
0      3      8      16
-----

```

PROCESS	BURSTTIME	ARRIVALTIME	WAITINGTIME	TURNAROUNDTIME
p3	3	1	0	2
p2	5	1	2	6
p1	8	1	7	14

AVERAGE WAITING TIME=3.666667

AVERAGE TURNAROUND TIME=7.333333

Result

The SJF scheduling algorithm has been implemented in C.

Ex No: 3C**3 C. IMPLEMENTATION OF FCFS SCHEDULING ALGORITHM****Aim**

To write a C program to implement First Come First Serve scheduling algorithm.

Description:

This is a non-preemptive technique. A single queue of ready processes is maintained, and the dispatcher always picks the first one. This method does not emphasise throughput, since long processes are allowed to monopolise CPU. The response time with FCFS can be high (with respect to execution time), especially if there is a high variance in process execution times.

This method penalises short processes following long ones, though no starvation is possible.

Jobs are processed in the order in which they arrive at a machine or work center

Algorithm:

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on order in which it requests CPU.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
intbt[50],wt[80],at[80],wat[30],ft[80],tat[80];
inti,n;
floatawt,att,sum=0,sum1=0;
char p[10][5];
clrscr();
printf("\nenter the number of process....");
scanf("%d",&n);
printf("\nEnter the process name and burst-time:");
for(i=0;i<n;i++)
scanf("%s%d",p[i],&bt[i]);
printf("\nEnter the arrival-time:");
for(i=0;i<n;i++)
scanf("%d",&at[i]);
wt[0]=0;
for(i=1;i<=n;i++)
```

```

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

ft[0]=bt[0];
for(i=1;i<=n;i++)
ft[i]=ft[i-1]+bt[i];
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n-----\n");
for(i=0;i<n;i++)
printf("\t%s\t",p[i]);
printf("\t\n");
printf("\n-----\n");
printf("\n");
for(i=0;i<n;i++)
printf("%d\t",wt[i]);
printf("%d",wt[n]+bt[n]);
printf("\n-----\n");
printf("\n");
for(i=0;i<n;i++)
wat[i]=wt[i]-at[i];
for(i=0;i<n;i++)
tat[i]=wat[i]+bt[i];
printf("\n FIRST COME FIRST SERVE\n");
printf("\n Process Burst-time Arrival-time Waiting-time Finish-time Turnaround-time\n");
for(i=0;i<n;i++)
printf("\n %d%s \t %d\t\t %d \t\t %d\t\t %d \t\t %d",i+1,p[i],bt[i],at[i],wat[i],ft[i],tat[i]);
for(i=0;i<n;i++)
sum=sum+wat[i];
awt=sum/n;
for(i=0;i<n;i++)
sum1=sum1+bt[i]+wt[i];
att=sum1/n;
printf("\n\nAverage waiting time:%f",awt);
printf("\n\nAverage turnaround time:%f",att);
getch();
}

```

Output:

```

enter the number of process....3
Enter the process name and burst-time:p1 2
p2 3
p3 4
Enter the arrival-time:0 1 2

```

GANTT CHART

p1	p2	p3
0	2	5
0	2	9

FIRST COME FIRST SERVE

Process	Burst-time	Arrival-time	Waiting-time	Finish-time	Turnaround-time
1p1	2	0	0	2	2
2p2	3	1	1	5	4
3p3	4	2	3	9	7

Average waiting time:1.333333

Average turnaround time:5.333333

Result:

The FCFS scheduling algorithm has been implemented in C.

Ex No: 3D

IMPLEMENTATION OF PRIORITY SCHEDULING ALGORITHM

Aim

To write a C program to implement Priority Scheduling algorithm.

Algorithm:

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on priority.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

Program:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
    int bt[30],pr[30],np;
    int wt[30],tat[30],wat[30],at[30],ft[30];
    int i,j,x,z,t;
    float sum1=0,sum=0,awt,att;
    char p[5][9],y[9];
    clrscr();
    printf("\nEnter the number of process");
```

```

scanf("%d",&np);
printf("\nEnter the process,burst-time and priority:");
for(i=0;i<np;i++)
scanf("%s%d%d",p[i],&bt[i],&pr[i]);
printf("\nEnter the arrival-time:");
for(i=0;i<np;i++)
scanf("%d",&at[i]);
for(i=0;i<np;i++)
for(j=i+1;j<np;j++)
{
if(pr[i]>pr[j])
{
x=pr[j];
pr[j]=pr[i];
pr[i]=x;
strcpy(y,p[j]);
strcpy(p[j],p[i]);
strcpy(p[i],y);
z=bt[j];
bt[j]=bt[i];
bt[i]=z;

}
}
wt[0]=0;
for(i=1;i<=np;i++)
wt[i]=wt[i-1]+bt[i-1];
ft[0]=bt[0];
for(i=1;i<np;i++)
ft[i]=ft[i-1]+bt[i];
printf("\n\n\tGANTT CHART\n");
printf("\n-----\n");
for(i=0;i<np;i++)
printf("\t%s\t",p[i]);
printf("\t\n");
printf("\n-----\n");
printf("\n");
for(i=0;i<=np;i++)
printf("%d\t",wt[i]);
printf("\n-----\n");
printf("\n");
for(i=0;i<np;i++)
wat[i]=wt[i]-at[i];
for(i=0;i<np;i++)
tat[i]=wat[i]+bt[i];
printf("\n\nPRIORITY SCHEDULING:\n");

```

```

printf("\nProcess Priority Burst-time Arrival-time Waiting-time Turnaround-time");
for(i=0;i<np;i++)
printf("\n\n%d%s\t%d\t%d\t%d\t%d\t%d",i+1,p[i],pr[i],bt[i],at[i],wt[i],tat[i]);
for(i=0;i<np;i++)
sum=sum+wat[i];
awt=sum/np;
for(i=0;i<np;i++)
sum1=sum1+tat[i];
att=sum1/np;
printf("\n\nAverage waiting time:%f",awt);
printf("\n\nAverageturn around time is:%f",att);
getch();
}

```

Output:

Enter the number of process3
Enter the process, burst-time and priority:
p1 3 3
p2 4 2
p3 5 1
Enter the arrival-time: 0 1 2

GANTT CHART

GANTT CHART					

	p3	p2	p1		

0	5	9	12		

PRIORITY SCHEDULING:

Process Priority Burst-time Arrival-time Waiting-time Turnaround-time

1p3	1	5	0	0	0
2p2	2	4	1	5	3
3p1	3	3	2	9	5

Average waiting time: 3.666667

Average turnaround time is: 2.666667

Result

The Priority scheduling algorithm has been implemented in C.

Viva Questions

1. Define process.
2. What is meant by burst time, waiting time and turnaround time?

3. What are the various scheduling algorithms?
4. Explain FCFS scheduling.
5. Explain SJF scheduling.
6. Explain Priority scheduling.
7. Can priority scheduling cause starvation?
8. Explain Round Robin scheduling.
9. What is meant by response time and slice time?
10. What is pre-emptive and non-preemptive scheduling?

Ex No: 4

PRODUCER CONSUMER PROBLEM USING SEMAPHORES

Aim:

To write a C program to implement producer consumer problem using semaphore

Algorithm:

Step 1: Declare the buffer size variable.

Step 2: Initialize empty, full and the value of mutex.

Step 3: For case 1, produce the item till the buffer is full after performing the wait operation of variables empty and mutex.

Step 4: Then perform signal of full.

Step 5: For case 2, consume the item till the buffer is empty. Perform the wait and signal operation..

Step 6: print the contents of the buffer.

Program:

```
#include<stdio.h>
#include<string.h>
#define SIZE 3
struct process
{
char a[10];
}buffer[SIZE];
intmutex=1,full=0,empty=SIZE,f=0;
int main()
{
//intmutex=1,full=0,empty=SIZE,f=0;
intch,i;
printf("\tProdecer Consumer Problem:\n");
while(1)
{
printf("\n The choices are\n");
printf("1.Producer Routine\n2.Consumer Routine\n3.Buffer Contents\n4.Exit\n");
```

```

printf("Enter your Choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
empty=wait(empty);
mutex=wait(mutex);
if(f==0)
{
printf("\nEnter the item to be added:\n");
scanf("%s",&buffer[full]);
full=signal(full);
printf("\nItem Produced Successfully:\n");
}
else
{
printf("\nBuffer is full\n");
f=0;
}
mutex=signal(mutex);
break;
case 2:
full=wait(full);
mutex=wait(mutex);
if(f==0)
{
printf("\nOne Item is Consumed:\n");
printf("\nConsumed item is: %s\n",buffer[0].a);
for(i=0;i<SIZE;i++)
strcpy(buffer[i].a,buffer[i+1].a);
empty=signal(empty);
}
else
{
printf("\nBuffer is empty\n");
f=0;
}
mutex=signal(mutex);
break;
case 3:
if(full!=0)
{
for(i=0;i<full;i++)
printf("\n%s\n",buffer[i].a);
}
else
printf("\nBuffer is empty\n");
break;
case 4:
exit(0);
default:

```



```

printf("Enter Correct Option");
break;
}
}
}
int wait(int s)
{
if(s==0)
f=1;
else
s--;
return s;
}
int signal(int s)
{
s++;
return s;
}

```

Output:

The choices are

- 1.Producer Routine
- 2.Consumer Routine
- 3.Buffer Contents
- 4.Exit

Enter your Choice:1

Enter the item to be added:

1

Item Produced Successfully:

The choices are

- 1.Producer Routine
- 2.Consumer Routine
- 3.Buffer Contents
- 4.Exit

Enter your Choice:1

Enter the item to be added:

2

Item Produced Successfully:

The choices are

- 1.Producer Routine
- 2.Consumer Routine
- 3.Buffer Contents
- 4.Exit

Enter your Choice:1

Enter the item to be added:3

Item Produced Successfully:

The choices are

- 1.Producer Routine
- 2.Consumer Routine

3.Buffer Contents

4.Exit

Enter your Choice:1

Buffer is full

The choices are

1.Producer Routine

2.Consumer Routine

3.Buffer Contents

4.Exit

Enter your Choice:2

One Item is Consumed:

Consumed item is: 1

The choices are

1.Producer Routine

2.Consumer Routine

3.Buffer Contents

4.Exit

Enter your Choice:

Result

Thus the producer consumer problem using semaphores has been implemented in C.

Viva Questions

1. What is a semaphore?
2. What is bounded-buffer problem?
3. What is readers-writers problem?
4. What is dining philosophers' problem?
5. What is process synchronization?
6. What is critical section problem?
7. What are the two operations of a semaphore? Explain.
8. What is deadlock and starvation?
9. What do you mean by monitors?
10. What is the advantage of monitors over semaphores?

ExNo:5 BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

Aim:

To write a program to implement bankers algorithm for deadlock avoidance.

Algorithm:

1. Start the program.
2. Get the number of process, resource, allocation matrix, Maximum matrix and available matrix.
3. Calculate need matrix using Maximum matrix and allocation matrix i.e.,
 $NEED[i,j]=MAX[i,j]-ALLOC[i,j]$. Also initialize work matrix i.e., $WORK[j]=AVAIL[j]$.
4. Now find safety sequence of the system using work matrix, need matrix.
5. Now compare NEED and WORK matrix, if $NEED[i,j] \leq WORK[j]$ then
 Execute the corresponding process also calculate new work matrix i.e.,
 $WORK[j] = WORK[j] + ALLOC[i,j]$.
6. If condition doesn't satisfy, move to next process and repeat Step 5, until completion of all processes and print the safety sequence.
7. If any process request for resource, get the process id and request matrix from the user.
8. Check the following conditions for the respective process using Request matrix, Need matrix and Available matrix i.e., $REQ[j] \leq NEED[k,j]$ and
 $REQ[j] \leq AVAIL[j]$.
9. If the above condition satisfies, request can be granted and proceed next step otherwise display error as "Request cannot be granted".
10. Now calculate new Available matrix, Allocation matrix, Need matrix and Work matrix i.e., $AVAIL[j] = AVAIL[j] - REQ[j]$,
 $WORK[j] = AVAIL[j]$, $ALLOC[k,j] = ALLOC[k,j] + REQ[j]$,
 $NEED[k,j] = NEED[k,j] - REQ[j]$.
11. To find safety sequence of the system, repeat Step 5 and Step 6 for all processes.
12. Then print the safety sequence of execution of processes.
13. Stop the program.

Program:

```
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

void main()

{

int alloc[20][20],max[20][20],avail[20],need[20][20],work[20]={0};

int newavail[20],req[20]={0},check=0,check2=0,cond,p;

int i=0,j=0,m=0,n=0,x=0,c[20]={0},count,count2,a[20],b;

int x2=0,c2[20];

clrscr();

printf("Enter the no. of resources\n");

scanf("%d",&m);

printf("Enter the no. of process");

scanf("%d",&n);

printf("\n Enter the resources for available\n");

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

{

printf("Enter the %d resources of avail",j);

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

work[j]=avail[j];

}

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

```
{
for(j=1;j<=m;j++)
{
printf("Enter the %d resources of %d alloc",j,i);
scanf("%d", &alloc[i][j]);
}
for(j=1;j<=m;j++)
{
printf("\n Enter the %d resource of %d max",j,i);
scanf("%d",&max[i][j]);
need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n Allocation    max    need\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=m;j++)
printf("%2d",alloc[i][j]);
printf("\t");
for(j=1;j<=m;j++)
printf("%2d",max[i][j]);
printf("\t");
for(j=1;j<=m;j++)
```

```
printf("%2d",need[i][j]);
printf("\n");
}
printf("\n Process executes in this order\n");
do{
for(i=1;i<=n;i++)
{
count=0;
if(c[i]!=i+1)
{
for(j=1;j<=m;j++)
{
if(need[i][j]<=work[j])
count = count+1;
}
if(count == m)
{
printf("p%d\t",i);
c[i]=i+1;
x = x+1;
for(j=1;j<=n;j++)
work[j]=work[j]+alloc[i][j];
}
```

```
}  
}  
check = check +1;  
}  
while(x<n && check <=n);  
if(x==n)  
printf("\n system is in saftey\n");  
else  
printf("\n System is not in saftey");  
printf("\n Checking the bankers algorithm after the request");  
printf("\n Enter the request process number");  
scanf("%d",&p);  
printf("\n Enter the values");  
for(j=1;j<=m;j++)  
{  
scanf("%d",&req[j]);  
}  
for(j=1;j<=m;j++)  
{  
if(req[j]<=avail[j]&& req[j]<=need[p][j])  
cond=cond+1;  
}  
if(cond != m)
```

```
{
for(j=1;j<=m;j++)
{
alloc[p][j]=alloc[p][j]+req[j];
avail[j]=avail[j]-req[j];
need[p][j]=need[p][j]-req[j];
}
}
else
{
printf("req is not satisfied");
exit(0);
}
printf("\n Execution of process after request");

do
{
for(i=1;i<=n;i++)
{
count2=0;
if(c2[i]!=i+1)
{
for(j=1;j<=m;j++)
```



```
{
if(need[i][j]<=avail[j])
count2 = count2+1;
}
if(count2 == m)
{
printf("p%d\t",i);
c2[i]=i+1;
x2 = x2+1;
for(j=1;j<=n;j++)
avail[j]=avail[j]+alloc[i][j];
}
}
}
check2 = check2 +1;
}
while(x2<n && check2 <=n);
if(x2==n)
printf("\n system is in saftey\n");
else
printf("\n System is not in saftey");
getch();
```

}

Output

Enter the no. of resources

3

Enter the no. of process3

Enter the resources for available

Enter the 1 resources of avail1

Enter the 2 resources of avail2

Enter the 3 resources of Avail3

Enter the 1 resources of 1 alloc3

Enter the 2 resources of 1 alloc2

Enter the 3 resources of 1 alloc1

Enter the 1 resource of 1 max1

Enter the 2 resource of 1 max2

Enter the 3 resource of 1 max3

Enter the 1 resources of 2 alloc2

Enter the 2 resources of 2 alloc2

Enter the 3 resources of 2 alloc2

Enter the 1 resource of 2 max3

Enter the 2 resource of 2 max3

Enter the 3 resource of 2 max3

Enter the 1 resources of 3 alloc2

Enter the 2 resources of 3 alloc2

Enter the 3 resources of 3 alloc2

Enter the 1 resource of 3 max1

Enter the 2 resource of 3 max2

Enter the 3 resource of 3 max3

Allocation max need

3 2 1 1 2 3 -2 0 2

2 2 2 3 3 3 1 1 1

2 2 2 1 2 3 -1 0 1

Process executes in this order

p1 p2 p3

system is in safety

Checking the bankers algorithm after the request

Enter the request process number2

Enter the values1

1

1

execution after the request

p1 p2 p3

the system is in safe state.

Result:

The program for banker's algorithm for deadlock avoidance was implemented and hence verified.

Viva Questions:

1. Explain Bankers algorithm?
2. What is system model?
3. What is meant by deadlock?
4. What are conditions under which a deadlock situation may arise?
5. What are the methods for Handling Deadlocks?
6. What is mutual exclusion?
7. What is dead lock avoidance?
8. What are the dead lock avoidance methods?
9. What is meant by starvation?
10. When a system is said to be in safe state?

Ex No: 6 IMPLEMENTATION OF DEADLOCK DETECTION

Aim:

To write a program to implement Deadlock detection algorithm.

Algorithm:

- Step 1:** Start the program.
- Step 2:** Get the values of resources and processes.
- Step 3:** Get the allocation and request matrix value.
- Step 4:** Get the available instances for each resources
- Step 5:** Check whether its possible to allocate using below condition.
- Step 6:** If $Need \leq Work$ update work as $work = work + allocation$
- Step 7:** Else proceed with next process.
- Step 8:** If it is possible to allocate then the system is in safe state.
- Step 9:** Else system is not in safe state.
- Step 10:** Stop the program.

Program:

```
#include <stdio.h>
main()
{
    int found,flag,l,p[5][6],tp,tr,c[5][6],i,j,k=1,m[6],r[6],a[6],temp[6],sum=0;
    printf("Enter total no of processes");
    scanf("%d",&tp);
    printf("Enter total no of resources");
    scanf("%d",&tr);
    printf("Enter claim (Max. Need) matrix\n");
    for(i=1;i<=tp;i++)
    {
        printf("process %d:\n",i);
        for(j=1;j<=tr;j++)
            scanf("%d",&c[i][j]);
    }
    printf("Enter allocation matrix\n");
    for(i=1;i<=tp;i++)
```

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

```

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

```

Output :

Enter total no. of processes 4

Enter total no. of resources 5

Enter claim (Max. Need) matrix :0 1 0 0 1

0 0 1 0 1

0 0 0 1

1 0 1 0 1

Enter allocation matrix : 1 0 1 1 0

1 1 0 0 0

0 0 0 1 0

0 0 0 0 0

Enter resource vector (Total resources) :

2 1 1 2 1

Enter availability vector (available resources) :

0 0 0 0 1

deadlock causing processes are : 2 3

Result:

The program for deadlock detection algorithm was implemented and hence verified

Viva Questions:

1. What is deadlock, starvation? Explain their difference.
2. How can we detect deadlocks?
3. What is deadlock prevention? Explain its methods.
4. What is meant by finite state?
5. What is resource allocation graph?
6. Define Request Edge and Assignment Edge.
7. What are the different types of deadlock detection used?
8. Explain How deadlocks are corrected.
9. List advantages of Deadlock Prevention.
10. List advantages and disadvantages of deadlock detection

Ex No: 7. THREADING AND SYNCHRONIZATION APPLICATIONS

Aim:

To write a program to implement the application of threading and synchronization.

Algorithm:

- Step 1:** Start the process
- Step 2:** Initialize mutex lock ,thread id and required other variables
- Step 3:** Create POSIX threads using pthread_create
- Step 4:** Define thread task by creating new function.
- Step 5:** Synchronize threads using mutex lock and unlock variables
- Step 6:** Using pthread_join allow threads to wait until parent process completes
- Step 7:** Destroy the synchronization variable mutex using pthread_mutex_destroy
- Step 8:** Stop the process

Program

//Threading Application Program

```
#include <pthread.h>
#include <stdio.h>
/* this function is run by the second thread */
void *inc_x(void *x_void_ptr)
{
/* increment x to 100 */
int *x_ptr = (int *)x_void_ptr;
while(++(*x_ptr) < 100);
printf("x increment finished\n");
return NULL;
}
int main()
{
int x = 0, y = 0;
/* show the initial values of x and y */
printf("x: %d, y: %d\n", x, y);
/* this variable is our reference to the second thread */
```



```

pthread_t inc_x_thread;
/* create a second thread which executes inc_x(&x) */
if(pthread_create(&inc_x_thread, NULL, inc_x, &x)) {
    fprintf(stderr, "Error creating thread\n");
    return 1;
}
/* increment y to 100 in the first thread */
while(++y < 100);
printf("y increment finished\n");
/* wait for the second thread to finish */
if(pthread_join(inc_x_thread, NULL)) {
    fprintf(stderr, "Error joining thread\n");
    return 2;
}
/* show the results - x is now 100 thanks to the second thread */
printf("x: %d, y: %d\n", x, y);
return 0;
}

```

Output

```
[student@vecit ~]$ gcc final.c -o final -pthread
```

```
[student@vecit ~]$ ./a.out
```

```
x: 0, y: 0
```

```
y increment finished
```

```
x increment finished
```

```
x: 100, y: 100
```

//Synchronization Application Program

```

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];

```

```
int counter;
pthread_mutex_t lock;
void* doSomething(void *arg)
{
    pthread_mutex_lock(&lock);
    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d started\n", counter);
    for(i=0; i<(0xFFFFFFFF);i++);
    printf("\n Job %d finished\n", counter);
    pthread_mutex_unlock(&lock);
    return NULL;
}
int main(void)
{
    int i = 0;
    int err;
    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n mutex init failed\n");
        return 1;
    }
    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &doSomething, NULL);
        if (err != 0)
            printf("\ncan't create thread :[%s]", strerror(err));
        i++;
    }
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Output

```
[student@vecit ~]$ synthread.c -o synthread -pthread
```

```
[student@vecit ~]$ ./a.out
```

Job 1 started

Job 1 finished

Job 2 started

Job 2 finished

Result:

The program for threading and synchronization was implemented and hence verified.

Viva Questions:

1. Define threads?
2. List the types of threads
3. What is multithreading?
4. What is posix thread?
5. How thread is different from process.
6. Compare User Threads and Kernel Threads.
7. Define synchronization.
8. How threads can be used for synchronization.
9. Define Thread Cancellation & Target Thread.
10. What are the different ways in which a Thread can be cancelled?

Ex No: 8 PAGING TECHNIQUE OF MEMORY MANAGEMENT

Aim:

To implement the Memory management using Paging technique.

Algorithm

1. Start the process
2. Declare the size and data variables
3. Get the number of processes to be inserted
4. Get the value
5. Start and identify the process with process id
6. Make the variables and arguments to be a global while in communication
7. Write a separate routine for creating memory delete memory in separate region
8. Give permission that a process can kill paging
9. Display the values
10. Stop the process

Program

```
#include<stdio.h>
int main()
{
int p[50],f[20],m,ps,n,i,pg,tf,off,pg1,off1,pa;
printf("\n Enter the memory size:");
scanf("%d",&m);
printf("\n Enter the process size:");
scanf("%d",&ps);
printf("\n Enter the page size:");
scanf("%d",&pg);
n=ps/pg;
tf=m/pg;
for(i=0;i<n;i++){
printf("Enter the free frame number:");
scanf("%d",&f[i]);
if(f[i]>tf){
```

```

printf("\n Invalid frame number\n");
i--;
}}printf("\n\t\t Page table");
printf("\n\t Page no \t Frame no");
for(i=0;i<pg;i++)
printf("\n\t %d\t\t%d",i,f[i]);
printf("\n Enter the logical address:");
scanf("%d",&off);
pgl=off/pg;
printf("\n The off set no:%d",off1);
pa=(f[pgl]*pg)+off1;
printf("\n The physical address:%d",pa);
}

```

Output

Enter the memory size:32

Enter the process size:16

Enter the page size:4

Enter the free frame number:1

Enter the free frame number:3

Enter the free frame number:4

Enter the free frame number:5

Page table

Page no	Frame no
0	1
1	3
2	4
3	5

Enter the logical address:6

Page no:1

The off set no:2

The physical address:10

Result:

The program for memory management using paging technique was implemented and hence verified.

Viva Questions:

1. Explain paging technique.
2. Define pages, frames, offset in paging technique.
3. What is meant by logical and physical address?
4. What is thrashing?
5. What is meant by page hit, page fault?
6. What is the formula to calculate physical address?
7. What is Demand paging?
8. What is virtual memory?
9. What is Locality of Reference?
10. What is Fragmentation?

Ex.No: 9. IMPLEMENTATION OF MEMORY MANAGEMENT SCHEME**Aim**

To write a program to implement memory management scheme.

A. First fit Algorithm**Algorithm**

1. Start the process
2. Declare the size
3. Get the number of processes to be inserted
4. Allocate the first hole that is big enough searching
5. Start at the beginning of the set of holes
6. If not start at the hole that is sharing the pervious first fit search end
7. Compare the hole

8. if large enough then stop searching in the procedure
9. Display the values
10. Stop the process

Program

```
#include<stdio.h>void main()
{
int i,j,temp,f[10],fp[10];
int no,p[15],part[15],pno,pr[15],prmem[15];
printf("\n*****");
printf("\n Implementation of first-fit algorithm");
printf("\n*****");
printf("\n Enter the number of partitions:");
scanf("%d",&no);
for(i=1;i<=no;i++)
{
p[i]=i;
printf("\n Enter the memory size for partition:%d:\t",i);
scanf("%d",&part[i]);
}
printf("\n Enter the number of process:");
scanf("%d",&pno);
for(i=1;i<=pno;i++)
{
pr[i]=i;
printf("\n Enter the size for process %d: \t",i);
scanf("%d",&prmem[i]);
}
printf("\n-----\n");
printf("process\t Partition\t Free_memory\n");
printf("\n-----\n");
j=1;
for(i=1;i<=no;i++)
{
```

```

f[i]=0;
fp[i]=0;
}
while(j<=pno)
{
for(i=1;i<=no;i++)
{
if((part[i]>=prmem[j])&&(f[i]==0))
{
part[i]=part[i]-prmem[j];
fp[j]=1;
f[i]=1;
printf("%d\t\t%d\t\t%d\n",pr[j],p[i],part[i]);
goto l1;
}
}
l1:
j++;
}
for(i=1;i<=no;i++)
{
if(f[i]==0)
{
printf ("%d\t\t %d \t\t \n",p[i],part[i]);
}
}
printf("\n The following process is not allocated:");
for(i=1;i<=pno;i++)
{
if(fp[i]==0)
{
printf("%d",pr[i]);
}
}
}

```


}

Output

Implementation of first-fit algorithm

Enter the number of partitions:3

Enter the memory size for partition:1: 60

Enter the memory size for partition:2: 80

Enter the memory size for partition:3: 100

Enter the number of process:3

Enter the size for process 1: 55

Enter the size for process 2: 75

Enter the size for process 3: 90

process| Partition| Free_memory|-----
1 1 5
2 2 5
3 3 10

The following process is not allocated:0

Result

The program for first fit was implemented and hence verified

B.Best Fit Algorithm

Algorithm

1. Start the process
2. Declare the size
3. Get the number of processes to be inserted
4. Allocate the best hole that is small enough searching
5. Start at the best of the set of holes
6. If not start at the hole that is sharing the pervious best fit search end

7. Compare the hole
8. If small enough then stop searching in the procedure
9. Display the values
10. Stop the process

Program

```

#include<stdio.h>
main()
{
int i,j,temp,f[10],fp[10];
int no,p[15],part[15],pno,pr[15],prmem[15];
printf("\n*****");
printf("\n implementation of best-fit algorithm");
printf("\n*****");
printf("\n Enter the number of partitions:");
scanf("%d",&no);
for(i=1;i<=no;i++)
{
p[i]=i;
printf("\n Enter the memory size for partition %d:\t",i);
scanf("%d",&part[i]);
}
for(i=1;i<=no;i++)
{
for(j=1;j<=i;j++)
{
if(part[j]>part[i])
{
temp=part[i];
part[i]=part[j];
part[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
}

```

```

}
}
}
printf("\n Enter the number of process:");
scanf("%d",&pno);
for(i=1;i<=pno;i++)
{
pr[i]=i;
printf("\n Enter the size for process %d:\t",i);
scanf("%d",&prmem[i]);
}
printf("\n-----\n");
printf("PROCESS\t PARTITION\t FREE_MEMORY\n");
printf("\n-----\n");
j=1;
for(i=1;i<=no;i++)
{
f[i]=0;
fp[j]=0;
}
while(j<=pno)
{
for(i=1;i<=no;i++)
{
if((part[i]>=prmem[j])&&(f[i]==0))
{
part[i]=part[i]-prmem[j];
fp[j]=1;
f[i]=1;
printf("%d \t\t %d \t\t %d \n",pr[j],p[i],part[i]);
goto l1;
}
}
}
l1:

```

```

j++;
}
for(i=1;i<=no;i++)
{
if(f[i]==0)
{
printf("%d \t\t %d \t\t \n",p[i],part[i]);
}
}
printf("\n The following process is not allocated:");
for(i=1;i<=pno;i++)
{
if(fp[i]==0)
{
printf("%d",pr[i]);
}
}
}
}

```

Output

implementation of best-fit algorithm

Enter the number of partitions:3

Enter the memory size for partition 1: 2

Enter the memory size for partition 2: 6

Enter the memory size for partition 3: 9

Enter the number of process:4

Enter the size for process 1: 2

Enter the size for process 2: 3

Enter the size for process 3: 3

Enter the size for process 4: 6

PROCESS	PARTITION	FREE_MEMORY
---------	-----------	-------------

1	1	0
2	2	3
3	3	6

The following process is not allocated:4

Result

The program for best fit was implemented and hence verified

C.Worst Fit Algorithm

Algorithm

1. Start the process
2. Declare the size
3. Get the number of processes to be inserted
4. Allocate the best hole that is small enough searching
5. Start at the best of the set of holes
6. If not start at the hole that is sharing the pervious best fit search end
7. Compare the hole
8. If small enough then stop searching in the procedure
9. Display the values
10. Stop the process

Program

```
#include<stdio.h>
main()
{
int i,j,temp,f[10],fp[10];
int no,p[15],prmem[15],part[15],pr[15],pno;
printf("\n*****\n");
printf("enter no of partitions:\n");
scanf("%d",&no);
for(i=1;i<=no;i++)
```

```
{
p[i]=i;
printf("enter the memory for partition %d\t",i);
scanf("%d",&part[i]);
}
for(i=1;i<=no;i++)
{
for(j=1;j<=i;j++)
{
if(part[j]<part[i])
{
temp=part[i];
part[i]=part[j];
part[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
printf("\n free memory");
for(i=1;i<=no;i++)
{
printf("\n partition %d:\t %d",p[i],part[i]);
}
printf("\n enter the number of process:");
scanf("%d",&pno);
for(i=1;i<=pno;i++)
{
pr[i]=i;
printf("\n enter the size of process %d:\t",i);
scanf("%d",&prmem[i]);
}
printf("\n-----\n");
```

```

printf("process| \t partition \t free memory\n");
printf("\n-----\n");
j=1;
for(i=1;i<=no;i++)
{
f[i]=0;
fp[j]=0;
}
while(j<=pno)
{
for(i=1;i<=no;i++){
if((part[i]>=prmem[j])&&(f[i]==0))
{
part[i]=part[i]-prmem[j];
fp[j]=1;
f[i]=1;
printf("%d \t %d \t %d \n",pr[j],p[j],part[i]);
goto l1;
}
}
l1:
j++;
}for(i=1;i<=no;i++)
{
if(f[i]==0)
{
printf("\t %d \t %d \n",p[i],part[i]);
}
}
printf("\n the following process is not allocated:");
for(i=1;i<=pno;i++)
{
if(fp[i]==0)
{

```

```
printf("%d",pr[i]);
}
}
}
```

Output

enter no of partitions:

3

enter the memory for partition 1 20

enter the memory for partition 2 30

enter the memory for partition 3 10

free memory

partition 2: 30

partition 1: 20

partition 3: 10

enter the number of process:3

enter the size of process 1: 12

enter the size of process 2: 22

enter the size of process 3: 7

```
-----
process|   partition   free memory|
```

```
-----
1       2           18
```

```
3       3           13
```

```
        3           10
```

the following process is not allocated:2

Result

The program for worst fit was implemented and hence verified.

Viva Questions:

1. What do you mean by First Fit?
2. What do you mean by Best Fit?

3. What do you mean by Worst Fit?
4. How is memory protected in a paged environment?
5. What is External Fragmentation?
6. What do you mean by Compaction?
7. Define Secondary Memory
8. What is called memory mapping?
9. Define Effective Access Time.
10. What is the use of Valid-Invalid Bits in Paging?

Ex No: 10

PAGE REPLACEMENT ALGORITHMS

Ex. No 10 ,A,

IMPLEMENTATION OF FIFO ALGORITHM

Aim:

To implement First in First Out page replacement technique.

Algorithm:

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

Program:

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
```

```

scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
    for(i=0;i<no;i++)
frame[i]= -1;
j=0;
printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
{
printf("%d\t\t",a[i]);
avail=0;
for(k=0;k<no;k++)
if(frame[k]==a[i])
avail=1;
if (avail==0)
{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf("Page Fault Is %d",count);
return 0;
}

```

Output

```

ENTER THE NUMBER OF PAGES: 20
ENTER THE PAGE NUMBER :   7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
ENTER THE NUMBER OF FRAMES :3

```

<u>ref string</u>	<u>page frames</u>		
7	7	-1	-1
0	7	0	-1
1	7	0	1
2	2	0	1
0			
3	2	3	1
0	2	3	0
4	4	3	0
2	4	2	0
3	4	2	3
0	0	2	3
3			
2			
1	0	1	3
2	0	1	2
0			
1			
7	7	1	2
0	7	0	2
1	7	0	1

Page Fault Is 15

Result:

The program for First in First Out page replacement technique was implemented and hence verified.

Ex No: 10B IMPLEMENTATION OF LRU ALGORITHM

Aim:

To implement Least Recently Used page replacement technique.

Algorithm:

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

Program:

```
#include<stdio.h>
main()
{
    int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
    printf("Enter no of pages:");
    scanf("%d",&n);
    printf("Enter the reference string:");
    for(i=0;i<n;i++)
    scanf("%d",&p[i]);
    printf("Enter no of frames:");
    scanf("%d",&f);
    q[k]=p[k];
    printf("\n\t%d\n",q[k]);
    c++;
    k++;
    for(i=1;i<n;i++)
    {
        c1=0;
```

```
for(j=0;j<f;j++)
{
if(p[i]!=q[j])
c1++;
}
if(c1==f)
{
c++;
if(k<f)
{
q[k]=p[i];
k++;
for(j=0;j<k;j++)
printf("\t%d",q[j]);
printf("\n");
}
else
{
for(r=0;r<f;r++)
{
c2[r]=0;
for(j=i-1;j<n;j--)
{
if(q[r]!=p[j])
c2[r]++;
else
break;
}
}
for(r=0;r<f;r++)
b[r]=c2[r];
for(r=0;r<f;r++)
{
for(j=r;j<f;j++)
```

```

{
if(b[r]<b[j])
{
t=b[r];
b[r]=b[j];
b[j]=t;
}
}
}
for(r=0;r<f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
printf("\t%d",q[r]);
}
printf("\n");
}
}
}
printf("\nThe no of page faults is %d",c);
}

```

Output:

Enter no of pages:10

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

Enter no of frames:3

```

7
7 5
7 5 9
4 5 9
4 3 9
4 3 7
9 3 7
9 6 7
9 6 2
1 6 2

```

The no of page faults is 10

Result:

The program for Least Recently Used page replacement technique was implemented and hence verified .

Ex No: 10C IMPLEMENTATION OF LFU ALGORITHM**Aim:**

To implement Least Frequently Used page replacement technique.

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>
int main()
{
    int f,p;
    int pages[50],frame[10],hit=0,count[50],time[50];
    int i,j,page,flag,least,minTime,temp;
    printf("Enter no of frames : ");
    scanf("%d",&f);
    printf("Enter no of pages : ");
    scanf("%d",&p);

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

```
    frame[i]=-1;
}
for(i=0;i<50;i++)
{
    count[i]=0;
}
printf("Enter page no : \n");
for(i=0;i<p;i++)
{
    scanf("%d",&pages[i]);
}
printf("\n");
for(i=0;i<p;i++)
{
    count[pages[i]]++;
    time[pages[i]]=i;
    flag=1;
    least=frame[0];
    for(j=0;j<f;j++)
    {
        if(frame[j]==-1 || frame[j]==pages[i])
        {
            if(frame[j]!=-1)
            {
                hit++;
            }
            flag=0;
            frame[j]=pages[i];
            break;
        }
    }
    if(count[least]>count[frame[j]])
    {
        least=frame[j];
    }
}
```



```

}
if(flag)
{
    minTime=50;
    for(j=0;j<f;j++)
    {
        if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
        {
            temp=j;
            minTime=time[frame[j]];
        }
    }
    count[frame[temp]]=0;
    frame[temp]=pages[i];
}
for(j=0;j<f;j++)
{
    printf("%d ",frame[j]);
}
printf("\n");
}
printf("Page hit = %d",hit);
return 0;
}

```

Output

Enter no of frames: 3

Enter no of pages:10

Enter page no:

2

3

4

2

1

3

7

5

4

3
 2 -1 -1
 2 3 -1
 2 3 4
 2 3 4
 2 1 4
 2 1 3
 2 7 3
 2 7 5
 2 4 5
 2 4 3

Page hit=1

Result:

The program for Least Frequently Used page replacement technique was implemented and hence verified

EXP:NO:10 D) OPTIMAL PAGE REPLACEMENT ALGORITHM

AIM:

TO write the c program to implement the optimal page replacement algorithm.

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

main()

{

int fr[5],i,j,k,t[5],p=1,flag=0,page[25],psz,nf,t1,u[5];

clrscr();

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

scanf("%d",&nf);

printf("\n enter the page size");

scanf("%d",&psz);

printf("\nenter the page sequence:");

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

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

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

fr[i]=-1;

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

{

if(full(fr,nf)==1)

break;

else

{
```

```
flag=0;

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

{

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

{

flag=1;

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

break;

}

}

if(flag==0)

{

fr[p]=page[i];

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

p++;

}

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

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

printf("\n");

}
```

```
p=0;

for(; i<=psz; i++)

{

flag=0;

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

{

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

{

flag=1;

break;

}

}

if(flag==0)

{

p++;

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

{

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

{

if(fr[j]==page[k])
```

```
u[j]=k;
break;
}
else
u[j]=21;
}
}
for(j=1; j<=nf; j++)
t[j]=u[j];
for(j=1; j<=nf; j++)
{
for(k=j+1; k<=nf; k++)
{
if(t[j]<t[k])
{
t1=t[j];
t[j]=t[k];
t[k]=t1;
}
}
}
for(j=1; j<=nf; j++)
```

```
{  
if(t[1]==u[j])  
  
{  
fr[j]=page[i];  
u[j]=i;  
}  
}  
printf("page fault\t");  
}  
else  
printf("\t");  
printf("%d:\t",page[i]);  
for(j=1; j<=nf; j++)  
printf(" %d ",fr[j]);  
printf("\n");  
}  
printf("\ntotal page faults: %d",p+3);  
getch();  
}  
int full(int a[],int n)  
{
```

```

int k;

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

{

if(a[k]==-1)

return 0;

}

return 1;

}

```

OUTPUT:

```

enter the number of frames:3

enter the page size10

enter the page sequence:2 3 2 4 2 1 3 7 5 4
      2:      2  -1  -1
      3:      2  3  -1
      2:      2  3  -1
      4:      2  3  4
      2:      2  3  4
page fault  1:      1  3  4
            3:      1  3  4
page fault  7:      7  7  4
page fault  5:      5  5  4
            4:      5  5  4

total page faults: 6

```

Viva Questions:

1. What is page and frame in OS?
2. Explain FIFO technique?
3. What is meant by Belady's Anamoly
4. What are the other page replacement techniques?
5. Explain LRU technique?
6. Advantages of LRU over FIFO

7. Disadvantages of LRU
8. Explain LFU technique?
9. Advantages of LFU over FIFO,LRU.
10. Which is the best page replacement technique?

Ex No: 11A

IMPLEMENTATION OF SINGLE LEVEL DIRECTORY

Aim:

To write a program to implement Single Level Directory structure.

Algorithm:

- Step 1:** Read the number of directories to be created.
Step 2: Input the number of files for each directory.
Step 3: Give the file names for the files.
Step 4: Display the single level directory containing files in each.

Program:

```
#include<stdio.h>
main()
{
intmaster,s[20];
char f[20][20][20];
char d[20][20];
inti,j;
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("*****\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
```

```

for(j=0;j<s[i];j++)
printf("%s\n\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
}

```

Output:

```

enter number of directorios:2
enter names of directories:dir1 dir2
enter size of directories:2 1
enter the file names :
f1
f2
f3

```

```

directory  size  filenames
*****
dir1      2   f1
          f2

dir2      1   f3

```

Result

Thus the single level directory has been implemented in C.

Ex No: 11B

IMPLEMENTATION OF TWO LEVEL DIRECTORY

Aim:

To write a program to implement two level directory structure.

Algorithm:

- Step 1:** Read the number of directories to be created.
- Step 2:** Input the number of subdirectories and names of the subdirectories for each directory.
- Step 3:** Give the file names for the files under the subdirectories.
- Step 4:** Display the two level structure for each directory created.

Program:

```

#include<stdio.h>
#include<conio.h>
structst
{
chardname[10];
charsdname[10][10];
charfname[10][10][10];
intds,sds[10];
}dir[10];
void main()
{
inti,j,k,n;
clrscr();
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");
}
getch();
}

```

Output:

```

Enter the number of directories: 1
Enter directory1 name: Dir1
Enter size of directories: 2
Enter subdirectory name and size: Sub1 1
Enter file name: File1
Enter subdirectory name and size: Sub2 1
Enter file name: File2
dirname      size  subdirname  size  files
*****
Dir1         2    Sub1        1    File1
           Sub2        1    File2

```

Result:

Thus the two level directory has been implemented in C.

Ex No: 11C**IMPLEMENTATION OF HIERARCHY LEVEL DIRECTORY****Aim:**

To write a program to implement Hierarchy level directory.

Algorithm:

Step 1: Read the Directory for which the hierarchy structure to be displayed.

Step 2: Write commands to access through the subdirectories and files in the given directory.

Step 3: If subdirectories found print them as next level of the directory

Step 4: If file found print them as next sub level of the sub directory

Program:

```

#define _XOPEN_SOURCE 500
#include<ftw.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdint.h>
#include<limits.h>
#define TRUE 1
#define FALSE 0

```

```

#define MAXDEPTH 20
staticintdisplay_info(const char *fpath, conststruct stat *sb, inttflag, struct FTW *ftwbuf)
{
char blanks[PATH_MAX];
const char *filename=fpath+ftwbuf->base;
int width = 4*ftwbuf->level;
memset(blanks, ' ', width);
blanks[width] = '\0';
printf("%s%s", blanks, filename);
if(tflag==FTW_DNR)
printf("(unreadabledirectory)");
else if(tflag==FTW_SL)
printf("(symboliclink)");
else if(tflag==FTW_SLN)
printf("(brokensymboliclink)");
else if(tflag==FTW_NS)
printf("statfailed");
printf("\n");
return 0;
}
int main(intargc, char *argv[])
{
int flags=0;
int status;
intch;
char options[]=":cdpm";
opterr=0;
while(TRUE)
{
ch=getopt(argc, argv, options);
/**itreturns-1whenitfindsnomoreoptions */
if(-1 == ch)
break;
switch(ch)
{
case 'c':
flags|=FTW_CHDIR;
break;
case 'd':
flags|=FTW_DEPTH;
break;
case 'p':
flags|=FTW_PHYS;
break;
case 'm':
flags|=FTW_MOUNT;
break;
default:
fprintf(stderr, "Badoptionfound.\n");
return 1;
}
}

```

```

}
if(optind<argc)
{
while(optind<argc)
{
status=nftw(argv[optind],display_info,MAXDEPTH,flags);
if(-1 == status)
{
perror("nftw");
exit(EXIT_FAILURE);
}
else
optind++;
} }
else
{
status=nftw(".",display_info,MAXDEPTH,flags);
if(-1==status)
{
perror("nftw");
exit(EXIT_FAILURE);
} }
exit(EXIT_SUCCESS);
}

```

Output:

```

enter name of dir/file(under root):Root
enter 1 for dir/ 2 for file:1
no of sub directories /files (for Root):2
enter name of dir/file(under Root):Java
enter 1 for dir/ 2 for file:1
no of sub directories /files (for Java):1
enter name of dir/file(under Java):JP
enter 1 for dir/ 2 for file:2
enter name of dir/file(under Root):HTML
enter 1 for dir/ 2 for file:1
no of sub directories /files (for HTML):1
enter name of dir/file(under HTML):DHTML
enter 1 for dir/ 2 for file:1
no of sub directories /files (for DHTML):1
enter name of dir/file(under DHTML):XML
enter 1 for dir/ 2 for file:1
no of sub directories /files (for XML):1
enter name of dir/file(under XML):ADo
enter 1 for dir/ 2 for file:1
no of sub directories /files (for ADo):AD
enter name of dir/file(under ADo):AD
enter 1 for dir/ 2 for file:2

```

Result:

Thus the hierarchy level directory has been implemented in C.

Viva Questions:

1. Explain Single Level Directory (SLD)?
2. What are the various File directory structures?
3. Advantages and Disadvantages of SLD
4. Give a real time example of SLD
5. Explain Two Level Directory (TLD)?
6. Advantages of TLD over SLD
7. Explain Hierarchical Level Directory (HLD)?
8. What is Directed acyclic graph?
9. What are the other file organization methods?
10. Draw a sample DAG?

Ex No: 12A**IMPLEMENTATION OF SEQUENTIAL FILE ALLOCATION****Aim:**

To write a program to implement sequential file allocation.

Algorithm:

Step 1: Create a disk space using array.

Step 2: Give the file length and starting block number.

Step 3: Allocate each block of the file sequentially in disk.

Step 4: The disk space is checked to ensure the block is allocated or not.

Program

```
#include<stdio.h>
#include<conio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
```

```

{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
return 0;}

```

Output

Enter the starting block & length of file4 10

4->1

5->1

6->1

7->1

8->1

9->1

10->1

11->1

12->1

13->1

the file is allocated to disk

if u want to enter more files?(y-1/n-0)

RESULT

Thus sequential file allocation has been implemented in C.

Ex No: 12B

IMPLEMENTATION OF INDEXED FILE ALLOCATION

Aim:

To write a C program to implement indexed file allocation.

Algorithm:

Step 1: Enter the number of files along with their file names.

Step 2: Give the file size and block size for each file.

Step 3: Enter the block numbers for each block in file.

Step 4: Display the indexed file allocation of each file.

Program

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{
int a[100],ind,i,k,j,index[100],n,c,count=0,block;
charfname[50];
clrscr();
for(i=0;i<100;i++)
a[i]=0;
X:
printf("\nenter the parent file name\n");
scanf("%s",fname);
printf("enter the index block\n");
scanf("%d",&ind);
if(a[ind]!=1)
{
a[ind]=1;
printf("\nenter no of files on index\n");
scanf("%d",&n);
printf("\n\t-----\n");
printf("\tfile name\tindex block\n");
printf("\t%s\t%d",fname,ind);
printf("\n\t-----\n");
printf("\nenter the files....");
}
y:
for(i=0;i<n;i++)
{
scanf("%d",&index[i]);
}
for(i=0;i<n;i++)
```

```

{
if(a[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
if(ind<index[j]||ind>index[j])
a[index[j]]=1;
printf("\nfile is allocated");
printf("\nfile is indexed");
for(k=0;k<n;k++)
printf("\n%d---->%d:%d",ind,index[k],a[index[k]]);
}
else
{
exit(0);
}}

```

Output:

```

enter the parent file name  srinivasan
enter the index block  14
enter no of files on index  5

```

```

-----
file name      index block
srinivasan    14
-----

```

```

enter the files....2 3 4 5 20
file is allocated
file is indexed
14->2:1
14->3:1
14->4:1
14->5:1
14->20:1

```

Result

Thus indexed file allocation has been implemented in C.

Ex No: 12C

IMPLEMENTATION OF LINKED FILE ALLOCATION

Aim:

To write a C program to implement linked file allocation.

Algorithm:

Step 1: Enter the number of files along with their file names.

Step 2: Give the file size and block size for each file.

Step 3: Enter the block numbers for each block in file.

Step 4: Display the linked file allocation of each file using their block delimiters.

Program:

```
#include<stdio.h>
struct file
{
charfname[50];
intstart,size,block[10];
}f[50];
main()
{
inti,j,n;
printf("Enter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter file name:");
scanf("%s",&f[i].fname);
printf("Enter starting block:");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("Enter no.of blocks:");
scanf("%d",&f[i].size);
printf("Enter block numbers:");
for(j=1;j<=f[i].size;j++)
{
scanf("%d",&f[i].block[j]);
}
}
printf("\n-----");
printf("\n|File\t\tstart\t\tsize\t\tblock\n");
printf("\n-----\n");

for(i=0;i<n;i++)
{
printf("%s\t\t%d\t\t%d\t\t",f[i].fname,f[i].start,f[i].size);
printf("%d",f[i].start);
for(j=0;j<=f[i].size-1;j++)
printf("--->%d",f[i].block[j+1]);
printf("\n");
printf("\n-----\n");
}
}
```

}

Output:

Enter no. of files:2

Enter file name:srinivasan

Enter starting block:3

Enter no.of blocks:3

Enter block numbers:1 2 4

Enter file name:seenu

Enter starting block:6

Enter no.of blocks:3

Enter block numbers:10 20 12

```

-----
|File |   start |   size |   block
-----
srinivasan  3     3     3--->1--->2--->4
-----
seenu       6     3     6--->10--->20--->12
-----

```

Result

Thus linked file allocation has been implemented in C.

Viva Questions

1. What is a file? What are the different file types?
2. What are the file access methods?
3. Explain Sequential File allocation?
4. Give some real time examples of sequential file allocation?
5. What is the need for efficient file allocation strategies?
6. Explain Indexed File allocation?
7. Advantages of indexed file allocation.
8. Explain Linked File allocation?
9. Advantages of linked file allocation?
10. What is directory?

Ex. no. 13. IMPLEMENTATION OF VARIOUS DISK SCHEDULING ALGORITHMS

Ex.No.13.a. FCFS (First-Come, First-Served)

```
#include <stdio.h>

void FCFS(int requests[], int n, int head) {
    int total_seek = 0;
    printf("Seek Sequence: %d", head);
    for (int i = 0; i < n; i++) {
        total_seek += abs(requests[i] - head);
        head = requests[i];
        printf(" -> %d", head);
    }
    printf("\nTotal Seek Time: %d\n", total_seek);
}

int main() {
    int requests[] = {98, 183, 37, 122, 14, 124, 65, 67};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head = 53;
    FCFS(requests, n, head);
    return 0;
}
```

Ex.No 13. B. SSTF (Shortest Seek Time First)

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

```
#include <stdlib.h>

#include <stdbool.h>

void SSTF(int requests[], int n, int head) {
    int total_seek = 0;
    bool visited[n];
    for (int i = 0; i < n; i++) visited[i] = false;

    printf("Seek Sequence: %d", head);
    for (int i = 0; i < n; i++) {
        int min_distance = __INT_MAX__;
        int index = -1;
        for (int j = 0; j < n; j++) {
            if (!visited[j] && abs(requests[j] - head) < min_distance) {
                min_distance = abs(requests[j] - head);
                index = j;
            }
        }
        visited[index] = true;
        total_seek += min_distance;
        head = requests[index];
        printf(" -> %d", head);
    }
    printf("\nTotal Seek Time: %d\n", total_seek);
}
```

```

int main() {
    int requests[] = {98, 183, 37, 122, 14, 124, 65, 67};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head = 53;
    SSTF(requests, n, head);
    return 0;
}

```

Ex. No 13 . SCAN Algorithm

```

#include <stdio.h>
#include <stdlib.h>

void SCAN(int requests[], int n, int head, int disk_size, int direction) {
    int total_seek = 0;
    int size = n + 2;
    int seek_sequence[size];
    int index = 0;

    requests[n] = 0;
    requests[n + 1] = disk_size - 1;
    n += 2;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (requests[j] > requests[j + 1]) {
                int temp = requests[j];

```

```
        requests[j] = requests[j + 1];
        requests[j + 1] = temp;
    }
}
}

int pos;
for (int i = 0; i < n; i++) {
    if (head < requests[i]) {
        pos = i;
        break;
    }
}

if (direction == 1) {
    for (int i = pos; i < n; i++) {
        seek_sequence[index++] = requests[i];
    }
    for (int i = pos - 1; i >= 0; i--) {
        seek_sequence[index++] = requests[i];
    }
} else {
    for (int i = pos - 1; i >= 0; i--) {
        seek_sequence[index++] = requests[i];
    }
    for (int i = pos; i < n; i++) {
```



```
        seek_sequence[index++] = requests[i];
    }
}

printf("Seek Sequence: %d", head);
for (int i = 0; i < size; i++) {
    total_seek += abs(seek_sequence[i] - head);
    head = seek_sequence[i];
    printf(" -> %d", head);
}
printf("\nTotal Seek Time: %d\n", total_seek);
}

int main() {
    int requests[] = {98, 183, 37, 122, 14, 124, 65, 67};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head = 53;

    SSTF(requests, n, head);

    return 0;
}
```