SHREENIVASA ENGINEERINGCOLLEGE

(Approved by AICTE and Affiliated to Anna University,Chennai)

B.Pallipatti, Bommidi, Pappireddipatti (Tk), Dharmapuri Dt, Tamilnadu-635 301

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Bonafide Certificate

This is to certify that this is a Bonafide Record of Practical work done by……of… .....Semester of.....Department in the …….Laboratory During the year……

Head of the Department Staff Incharge

Submitted for the University Examination held on …….

Internal Examiner External Examiner

S.No.	Date	Name of the Experiment	Page No.	Marks	Signature

S.No.	Date	Name of the Experiment	Page No.	Marks	Signature

EX.No. : 1	INSTALLATION OF WINDOWS OPERATING SYSTEM
DATE :	

OBJECTIVE:

To Acquire Windows operating system installation skill.

OUTCOME:

Students can able to understand the step by step procedure of windows OS installation.

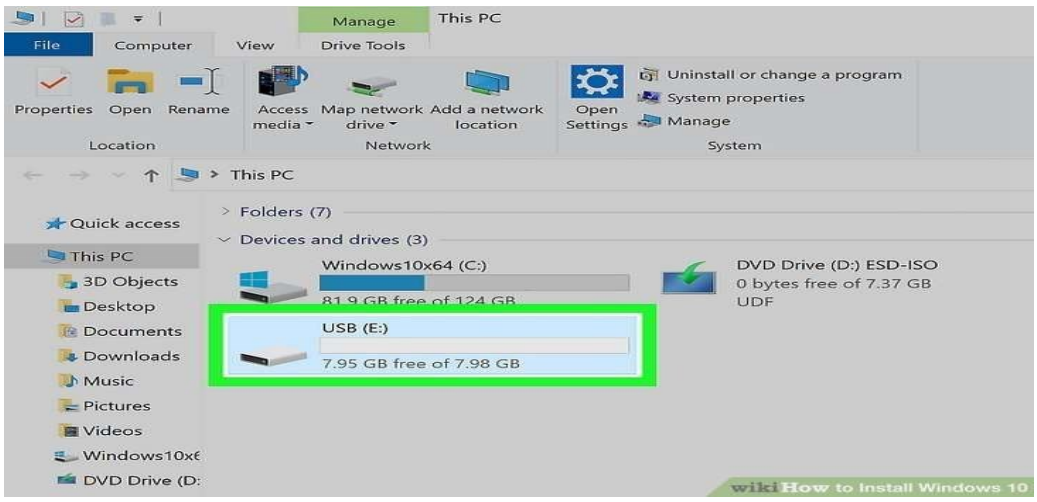
AIM:

To write a step by step process of windows operating system installation.

Installation processes:

Step1: Creating an installation Disc or Drive

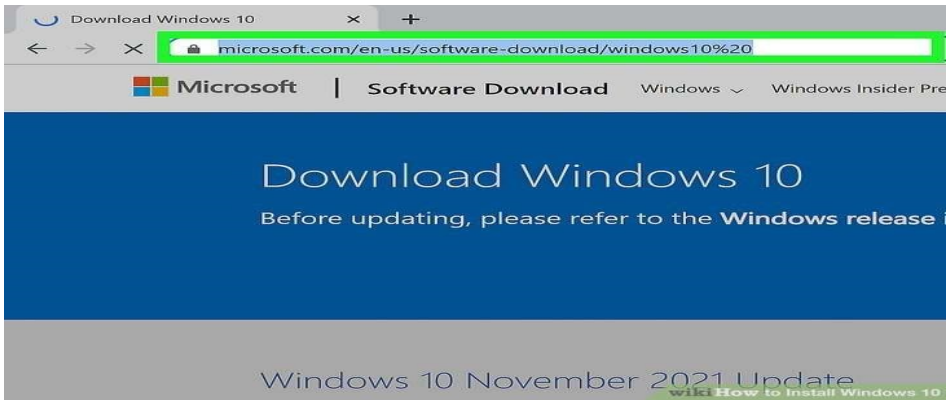
Connect a blank USB flash drive or insert a blank writable DVD. You can install Windows 10 by creating a bootable USB flash drive or DVD that contains the Windows 10 installation files. You'll need a USB flash drive that's at least 8GB, or any blank DVD to get started.



Step-2: Make sure you have the product key.

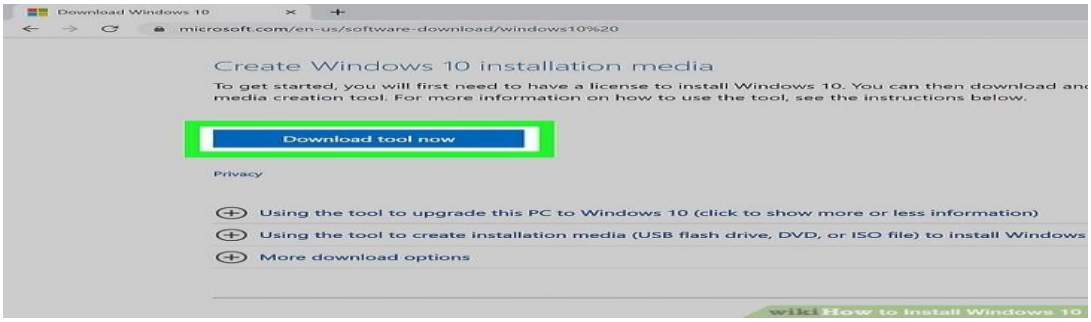
If you bought Windows 10 through Microsoft using your Microsoft account, your product key is already linked to your account. If you bought Windows 10 from another retailer, you'll have a 25character product key that you'll need to have handy to activate Windows

- If you don't have a product key or you're installing Windows 10 on a new hard drive, make sure you've linked your Windows 10 digital license to your Microsoft account before you start the installation.
- Head to Settings > Update & Security > Activation from the current installation—if the activation status says Windows is activated with a digital license,
- click Add an account and follow the on-screen instructions to link your Microsoft account.
- If you're upgrading from an earlier version and your PC qualifies for a free upgrade, you won't need a product key.



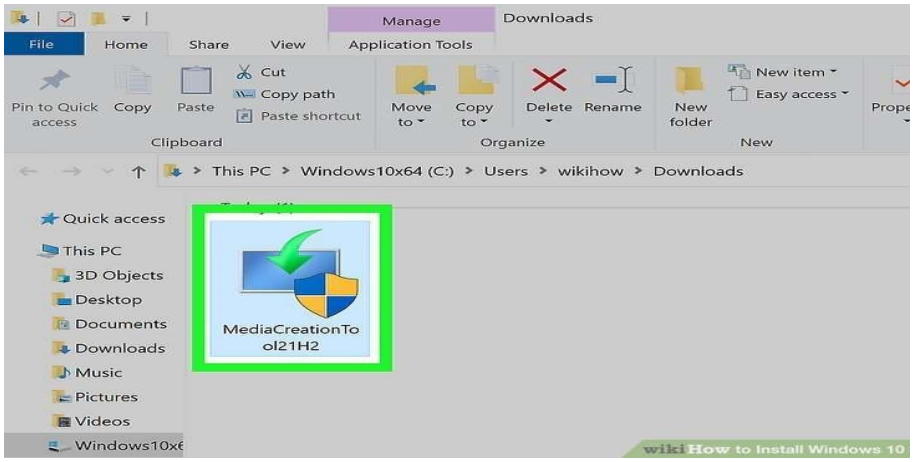
Step:3

Go to https://www.microsoft.com/en-us/software-download/windows10%20. This is the official download site for Windows 10.



Step:4

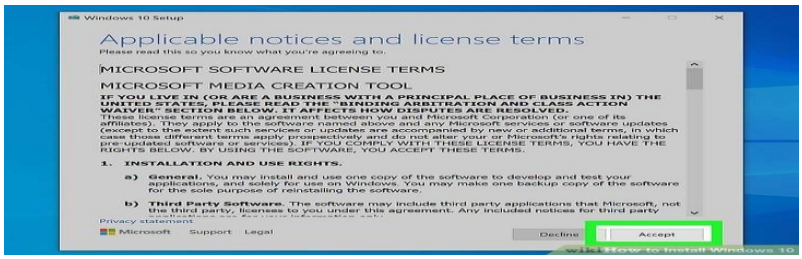
Click Download tool now. This is a blue button in the middle of the page. This downloads the Media Creation Tool, which you'll use to create your installation media (or start your upgrade).



Step:5

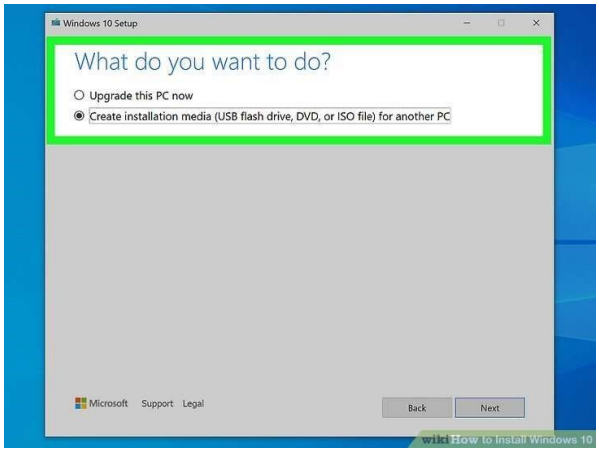
Double-click the downloaded file. Its name begins with "MediaCreationTool" and ends with ".exe." You'll find it in your default download folder, which is usually called Downloads.

- Click Yes when prompted to allow the installer to run.



Step:6

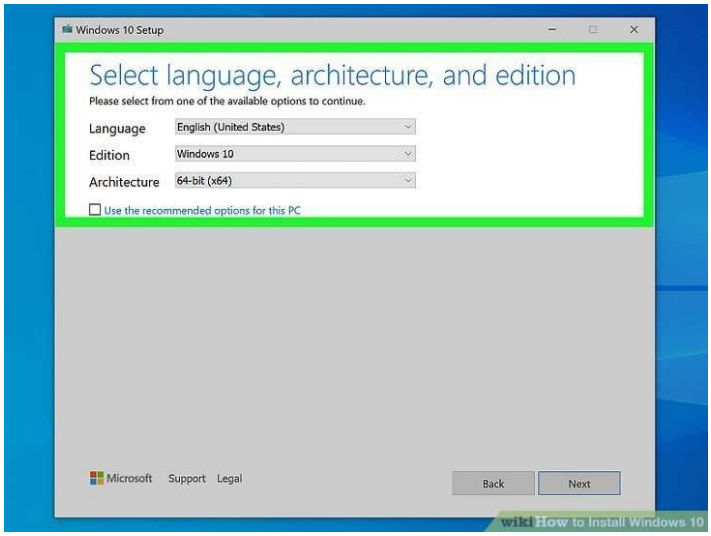
Click Accept to accept the license. It's in the bottom-right corner of the window.



Step-7:

Select "Create installation media" and click OK. This option lets you create a Windows installation disc or drive that will work on any compatible PC, not just the one you're using now.

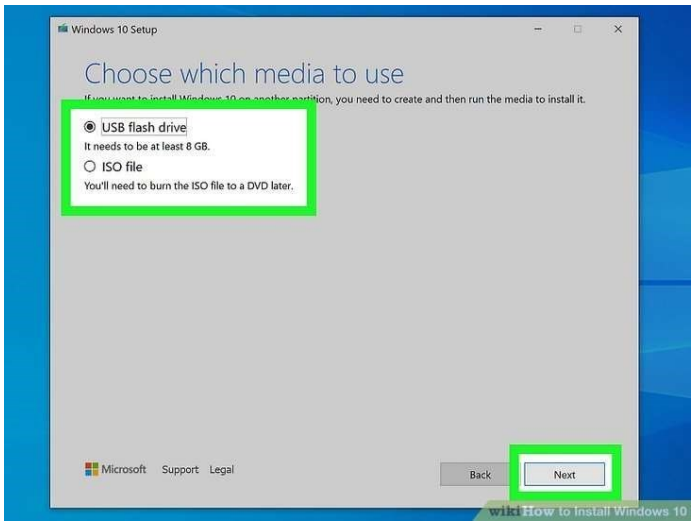
- If you're updating your PC from an earlier version of Windows, select Upgrade this PC now instead, and then follow the on-screen instructions to install Windows 10. You're done!



Step-08:

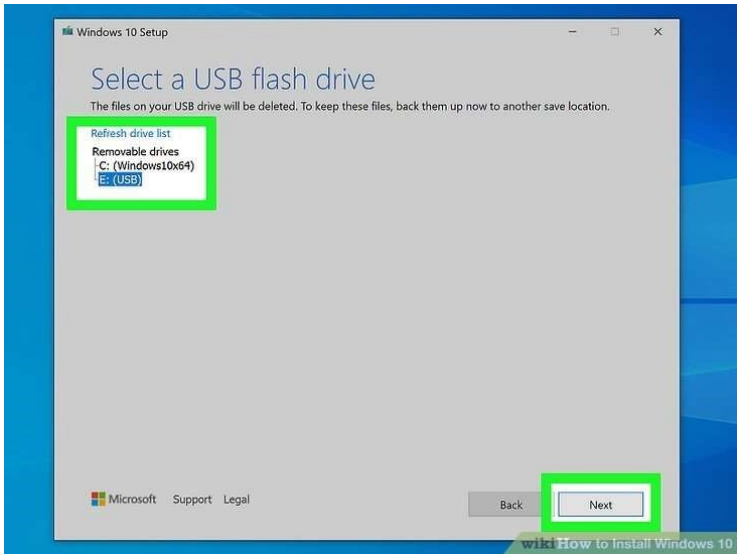
Select your preferences and click Next. If you're installing Windows on the current PC, you can keep the default options. If you need to install on a different PC, make sure you choose the language and edition for which you have a license, and select the architecture (64-bit or 32-bit) that matches the PC you're going to install on.

- If you're not sure about the architecture, you can choose Both from the menu.



Step-09:

Choose an installation type and click Next. An ISO file is a type of file that can be burned to a DVD, so choose that option if you plan to create a DVD. Otherwise, choose the USB flash drive option.



Step-10:

Create your installation media. The steps are a little different depending on what you're doing:

- Flash drive: Select your flash drive from the list, click Next, and wait for the installation files to install. When the process is complete, click Finish.
- DVD/ISO: Click Save to save the ISO file to your computer—it may take a while because the file is large and has to be downloaded. Once downloaded, you'll see a progress screen that monitors the download. When the download is complete, click Open DVD burner on the "Burn the ISO file to a DVD" screen, select your DVD burner, and then click Burn to create your DVD.

Step-11:

Booting from Windows 10 Installation Media

- Connect your Windows 10 installation media. If you created a flash drive, connect it to the PC on which you want to install Windows 10. If you made a DVD, insert it into the drive now.



Boot the PC into the BIOS. If your PC is not already set up to boot from your flash or optical drive, rebooting from your installation media won't work. You'll need to make a quick change in your BIOS to change the boot order. There are a few ways to get in:

- Windows 8.1 or 10: From Windows, open Settings, select Update & Recovery or Update & Security, and go to Recovery > Restart now > Troubleshoot > Advanced Options > UEFI Firmware Settings > Restart.
- Any PC: Reboot the PC and immediately start pressing (over and over again) the keyboard key required by your PC to enter "Setup," or the BIOS. The key varies by computer, but here are some of the most common keys:
- Acer and Asus: F2 or Del
- Dell: F2 or F12
- HP: ESC or F10
- Lenovo: F1, F2, or Fn + F2
- Lenovo ThinkPads: Enter + F1.
- MSI: DEL
- Microsoft Surface Tablets: Press and hold the volume-up button.
- Samsung and Toshiba: F2
- Sony: F1, F2, or F3



Step-12:

Go to the Boot tab. You'll use the arrow keys to select it.

The Boot tab may instead say Boot Options or Boot Order, depending on your computer's manufacturer



Step-13:

Select a device from which to boot. You have a couple of options here:

- For a USB flash drive, select the Removable Devices option.
- For a disc installation, select the CD-ROM Drive or Optical Drive option



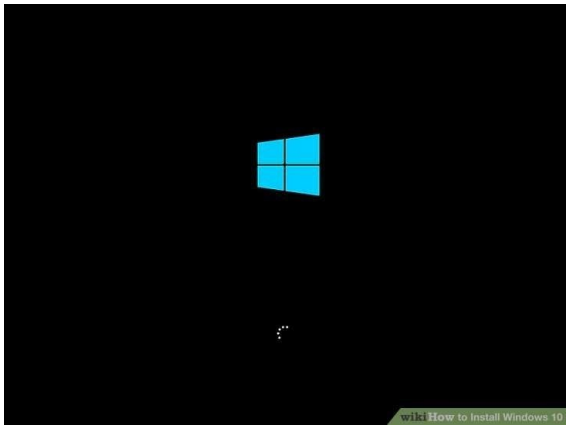
Step-14:

Press the + key until your boot option is first. Once either Removable Devices or CD-ROM Drive is at the top of the list, your computer will select your choice as its default boot option. On some computers, you'll instead press one of the function keys (e.g., F5 or the arrow keys to navigate an option up to the top of the menu. The key will be listed on the right side of the screen.



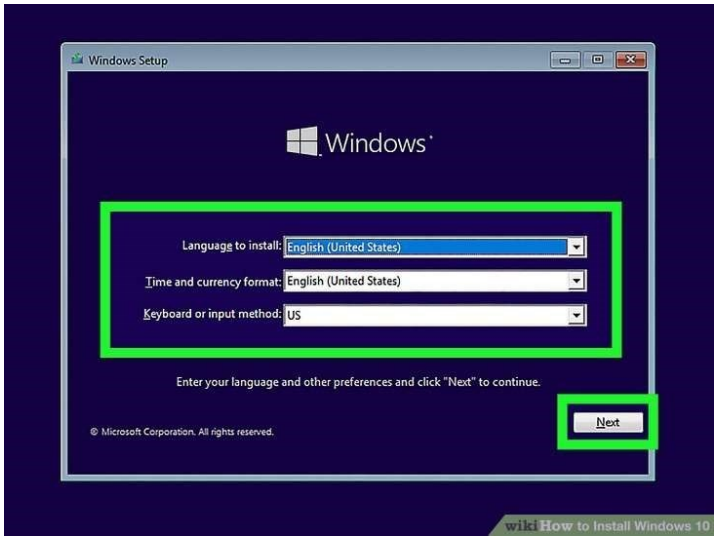
Step-15:

Save your settings. You should see a key prompt (e.g., F10 at the bottom of the screen that correlates to "Save and Exit". Pressing it will save your settings and restart your computer.



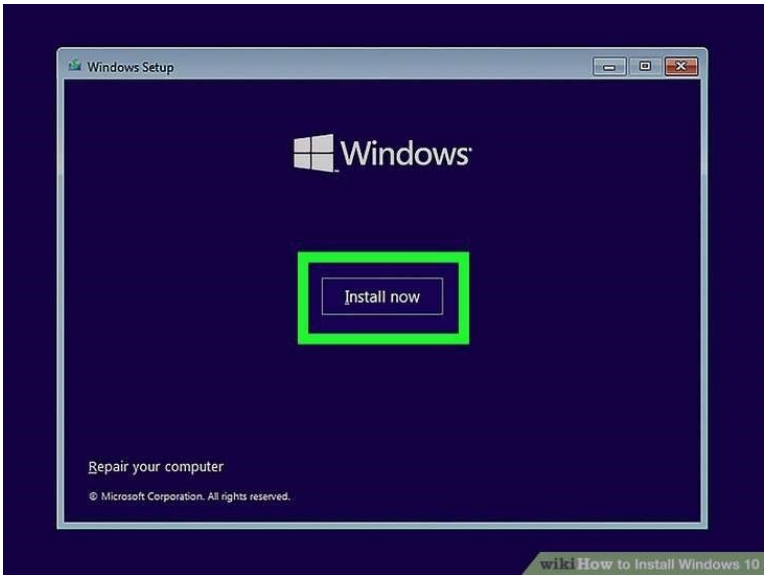
Step-16:

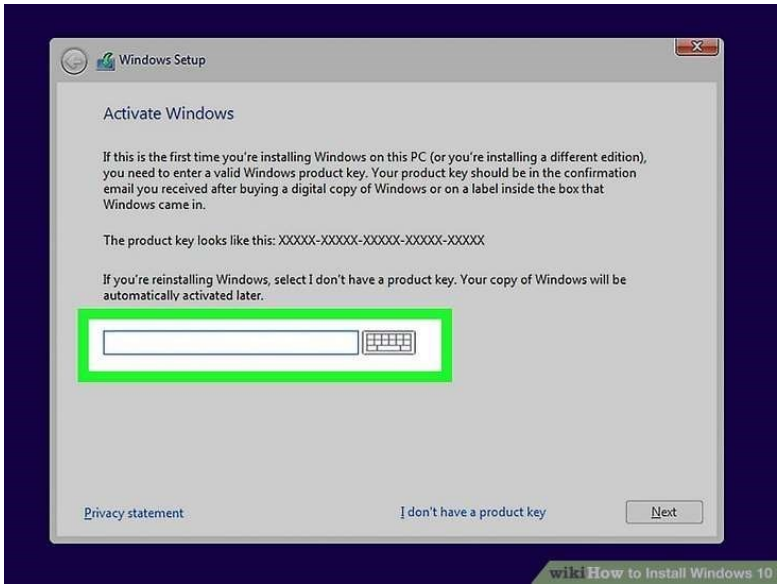
Wait for your computer to restart. Once your computer finishes restarting, you'll see a window here with your geographical data. You're now ready to begin setting up your Windows 10 installation.

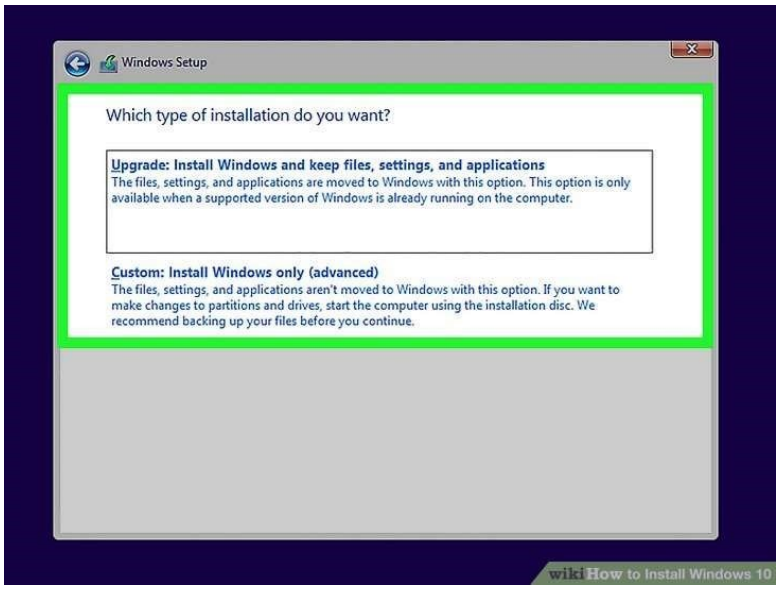


Step-17:

Click Next when prompted. You can also change the options on this page (e.g., the setup language) before continuing if need be







Step-18:

Follow the on-screen instructions to install Windows 10. You'll be asked to perform a few tasks, such as connecting to Wi-Fi and choosing some preferences. Once the installation is complete, you'll have a fresh new installation of Windows 10.

- If you're upgrading from an earlier version of Windows, you'll be asked if you want to upgrade the current operating system or do a custom install. If you choose Upgrade, you'll preserve existing apps and files.

Conclusion:

Thus the windows operating system was installed and configured successfully.

EX.No. : 2	INSTALLATION OF UNIX COMMAND & SHELL PROGRAMMING
DATE :	

OBJECTIVE:

Acquire operating system skills through UNIX Commands.

OUTCOME: Students can understand how to work with UNIX shell commands.

PRE-REQUEST: Shell- Basics.

SAMPLE PROGRAM TITLE:
Study of basic shell commands..

UNIX SHELL:

A UNIX shell is a command-line interpreter that provides a user interface for the UNIX operating system. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute or by creating text scripts of one or more such commands

GENERAL COMMANDS:

date	Used to display the current system date andtime.
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, Username
who am i	Used to display the login details of the user
tty	Used to display the terminal name (or) terminaltype
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

lp - file Allows the user to spool a job along with others in a print queue history

To display the commands used by the user since logon.
Exit Exit from a process. If shell is the only process then logs out
Clear Clear the screen

DIRECTORY COMMANDS:

Pwd Path of the present working directory
Mkdir dir A directory is created in the name under the current directory
Mkdir dir1 dir2 A number of sub-directories can be created under one stroke
Cd subdir Change Directory
Cd To switch to the home directory.
cd .. To move back to the parent directory rmdir subdir
Removes an empty sub-directory.

File commands:

t > filename
To create a file with some contents.
At the end press Ctrl+d. (> indicates redirecting output to a file.)
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
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.

rm * To delete all the files in the directory. rm –r *
Deletes all files and sub-directories rm –f *

To forcibly remove even write-protectedfiles ls Lists all files and subdirectories (blue colored) in sorted manner. ls name

name
To check whether a file or directory exists. ls name*
Short-hand notation to list out filenames of a specificpattern. ls –a

Lists all files including hidden files (files beginning with .)

Long listing showing file access rights (read/write/execute-rwx
for user/group/others-ugo).
cmp file1 file2

Used to compare two files. Displays nothing if files are empty wc

Produces the counts of lines (l), words(w), and characters(c). chmod

Perform file Changes permission for the specified file. who | wc –l

To terminate the unix session execute the command exit or logout head

Print the first N number of lines. tail

Print the last N number of lines. New

file Splits screen into multiple windows and opens thefile. Finger

Gathers and displays the information about theusers. Grep

Used to search and print specified patterns from a file
.
Fetches one copy of the redundant records writing them tothe standard output.

diff Difference between the two files

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.

To delete all the files in the directory. rm –r *
Deletes all files and sub-directories rm –f *

To forcibly remove even write-protectedfiles ls Lists all files and subdirectories (blue colored) in sorted manner. ls name

To check whether a file or directory exists. ls name*
Short-hand notation to list out filenames of a specificpattern. ls –a

Lists all files including hidden files (files beginning with .)

Long listing showing file access rights (read/write/execute-rwx
for user/group/others-ugo).
cmp file1 file2

Used to compare two files. Displays nothing if files are empty wc

Produces the counts of lines (l), words(w), and characters(c). chmod

Perform file Changes permission for the specified file. who | wc –l

To terminate the unix session execute the command exit or logout head

Print the first N number of lines. tail

Print the last N number of lines. new file

Splits screen into multiple windows and opens thefile. finger

Gathers and displays the information about theusers. grep

Used to search and print specified patterns from a file.

.

Fetches one copy of the redundant records writing them tothe standard output.

diff Difference between the two files

CONCLUSION:

Thus the basics of UNIX commands are studied by executing the commands

EX.No.:3	IMPLEMENTATION OF UNIX SYSTEM CALLS
DATE :	

IMPLEMENTATION OF UNIX SYSTEM CALLS
(getpid, waitpid)

OBJECTIVE
Acquire operating system skills through System calls & Commands.

OUTCOME
Students become expertise to write a C coding to simulate UNIX system calls.

PRE-REQUISITE
C- Basics.

AIM:
To write programs using the following system calls of UNIX operating system

fork, getpid, waitpid..

ALGORITHM:

Step 1: Start the program.

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

Step 3: Create a identical copy of parent process (child process) using fork ()
system call

Step 4: Process id is created for both child and parent process using getpid()
system call

Step 5: The Child process is made to execute.

Step 6: After the exiting of the child process parent process will getexit.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h
>
#include<unistd.h>
void parent(); void
child();  main()
{

int r; r=fork(); if(r<0)

{ printf("fork is empty");

 }  if(r>0)

parent(r);

else


child(r);


      }

void

child()     {

int i;

printf("\nchild starts
executing"); for(i=0;i<5;i++)

{ printf("\n child %d is executing %d time",getpid(),i);

sleep(i);

} printf("child %d exit

\n",getpid());

} void

parent(child_pid)

{

printf("\n Parent %d waiting for child %d to start the executing",
getpid(),child_pid); printf("\n");
```

```
waitpid(child_pid,NULL,0);
printf("Parent %d exit
\n",getpid());

}
```

OUTPUT:

[com39@localhost ~]$ cd os

[com39@localhost os]$ gcc ex1a.c

[com39@localhost os]$ ./a.out fork is empty

Parent 1769 waiting for child 1770 to start the executing fork is empty

child starts executing

child 1770 is executing 0 time

child 1770 is executing 1 time

child 1770 is executing 2 time

child 1770 is executing 3 time

child 1770 is executing 4 time

child 1770 exit

Parent 1769 exit [com39@localhost os]

CONCLUSION:

Thus the C program to simulate UNIX system calls fork, getpid, wait has been executed successfu

EX.No.:4	IMPLEMENTATION OF UNIX SYSTEM CALLS
DATE :	

OBJECTIVE:

Acquire operating system skills through System calls & Commands.

OUTCOME:

Students become expertise to write a Ccoding to simulate UNIX system calls.

PRE-REQUEST: C- Basics.

AIM:

To write programs using the following system calls of UNIX operating system opendir(), readdir(), closedir() and exit().

ALGORITHM:

Step 1: Start the program.

Step 2: Create a directory entry in the directory structure and a directory variable.

Step 3: Get the name of the directory to be opened.

Step 4: Using the system call opendir(),open the given directory

and store it in the directory variable.

Step 5: Read the contents of the directory using readdir()system call

and print the list of files in the directory.

Step 6: Close the directory after the read and write process.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>

 #include<dirent.h>

#include<stdlib.h>  struct

dirent *dptr; int main(int

argc,char *argv[])

{ char buff[256]; DIR *dirp; printf("\n \n enter

directory name");  scanf("%s",buff);

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

OUTPUT:

```
[com39@localhost ~]$ cd os

[com39@localhost os]$gcc ex1b.c

[com39@localhost os]$ ./a.out

ENTER A DIRECTORY NAME: oslabpgmsex5b.c

ex5b.c
ex1b.c
..
ex1a.c
ex4a.c
printf("erro)
while(dptrp)
{

printf("%se}
closedir(di}
 ex2.c
ex3a.c
ex1d.c
ex4b.c
ex3b.c
ex1c.c
ex5a.c
[com39@localhost os]$
```

CONCLUSION:

Thus the C program to simulate UNIX system calls opendir, readdir, closedir and exit has been executed successfull

EX.No.:5	CPU SCHEDULING ALGORITHMS - FIRST COME FIRST SERVER (FCFS)
DATE :	

OBJECTIVE:

Acquire operating system skills through CPU scheduling algorithms.

OUTCOME:

Students can understand how to schedule jobs using CPU algorithms.

PRE-REQUEST: C program

SAMPLE PROGRAM TITLE:

Implementation of First Come First Server Algorithm

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the variables.

Step 3: Get the number of process and their burst time of each process

Step 4: Calculate the waiting time and turnaround time of each of the processes

Step 5: Calculate average waiting and average turnaround time of all the process.

Step 6: Display waiting time, turnaround time, average waiting time and turnaround time of the processes

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>  #include<conio.h>
main() { int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg; clrscr(); printf("\nEnter
the number of processes -- ");  scanf("%d",
&n);

for(i=0;i<n;i++)
{ printf("\nEnter Burst Time for Process %d -- ",
i);  scanf("%d", &bt[i]);
} wt[0] = wtavg = 0; tat[0] = tatavg = bt[0];
for(i=1;i<n;i++) { wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i]; wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++) printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i,
bt[i], wt[i], tat[i]);  printf("\nAverage Waiting Time -- %f",
wtavg/n);  printf("\nAverage Turnaround Time -- %f",
tatavg/n); getch(); }
```

INPUT

Enter the number of processes -- 3

Enter Burst Time for Process 0 – 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 – 3

OUTPUT PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	24
P1	3	24	27
P2	3	27	30

Average Waiting Time-- 17.000000

Average Turnaround Time -- 27.000000

CONCLUSION:

Thus the C program using FCFS scheduling has been executed successfully

EX.No.:6	CPU SCHEDULING ALGORITHMS – SHORTEST JOB FIRST (SJF)
DATE :	

OBJECTIVE:
Acquire operating system skills through CPU scheduling algorithms.
OUTCOME:
Students can understand how to schedule jobs using CPU algorithms.

PRE-REQUEST: C program
SAMPLE PROGRAM TITLE:
Implementation of First Come First Server Algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the variables.
Step 3: Get the number of process and their burst time of each process.
Step 4: Calculate the waiting time and turnaround time of each of the processes Step 5: Calculate average waiting and average turnaround time of all the process. Step 6: Display waiting time, turnaround time, average waiting time and turnaround time of the processes Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

main()

{


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

temp;  float wtavg, tatavg; clrscr();

printf("\nEnter the number of processes --

");  scanf("%d", &n); for(i=0;i<n;i++)

{

scanf("

%d",

&bt[i]);

}
```

```
 for(i=0;i<n;i++)
for(k=i+1;k<n;k++
) if(bt[i]>bt[k])
{
temp=bt[i];

}  bt[i]=bt[k]; bt[k]=temp; temp=p[i]; p[i]=p[k];
p[k]=temp;  wt[0] = wtavg = 0; tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{ wt[i] = wt[i-1] +bt[i-1]; tat[i] = tat[i-1]
+bt[i]; wtavg = wtavg + wt[i]; tatavg =
tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND
TIME\n");
for(i=0;i<n;i++) printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i],
bt[i], wt[i], tat[i]);  printf("\nAverage Waiting Time -- %f",
wtavg/n);  printf("\nAverage Turnaround Time -- %f",
tatavg/n); getch();}
```

INPUT

Enter the number of processes -- 4

Enter Burst Time for

OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	6	3	9
P1	7	16	24
P2	7	9	16
P3	3	0	3

Average waiting time : 7.000000

Average turnaround time : 13.000000

CONCLUSION:

Thus the C program using SJF scheduling has been executed successfully.

EX.No.:7	CPU SCHEDULING ALGORITHMS – ROUND ROBIN SCHEDULING ALGORITHM
DATE :	

OBJECTIVE:

Acquire programming skills for application development in real-world problem solving.

OUTCOME:

Students become expertise to write a C coding to implement CPU scheduling algorithm.

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of Round Robin Scheduling.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the variables.

Step 3: Get the number of process and their burst time of each process.

Step 4: Get the quantum time.

Step 5: Schedule the CPU to the first process, after that quantum time is over schedule the second process. This will continue until all processes in the system complete their turn.

Step 6: Draw the chart of the processes as Round Robin manner.

Step 7: Calculate waiting time and turnaround time of each process.

Step 8: Calculate average waiting and average turnaround time of all the process.

Step 9: Display waiting time, turnaround time, average waiting time and turnaround time of the processes.

Step 10: Stop the program.

PROGRAM:

```
#include<stdio.h>  main() { int

pt[10][10],a[10][10],at[10],pname[10][10],i,j,n,k=0,q,sum=0;

float avg; printf("\n\n Enter the number of processes : ");

scanf("%d",&n);


for(i=0;i<10;i++)
```

```
{
for(j=0;j<10;j++)
{
pt[i][j]=0;
a[i][j]=0;
} } for(i=0;i<n;i++) { j=0; printf("\n\n Enter the process time for process %d :
",i+1); scanf("%d",&pt[i][j]);
} printf("\n\n Enter the time slice : ");
scanf("%d",&q); printf("\n\n"); for(j=0;j<10;j++)
{
for(i=0;i<n;i++)
{ a[2*j][i]=k;
if((pt[i][j]<=q)&&(pt[i][j]!=0))
{ pt[i][j+1]=0; printf(" %d P%d
%d\n",k,i+1,k+pt[i][j]); k+=pt[i][j]; a[2*j+1][i]=k;
} else
if(pt[i][j]!=0)
{ pt[i][j+1]=pt[i][j]-q; printf(" %d P%d
%d\n",k,i+1,(k+q)); k+=q; a[2*j+1][i]=k;
}
else {
a[2*j][i]=0;
a[2*j+1][i]=0;
} } }
for(i=0;i<n;i++
)
{
sum+=a[0][i];
}
for(i=0;i<n;i++
)
```

```
{
for(j=1;j<10;j++)
{ if((a[j][i]!=0)&&(a[j+1][i]!=0)&&((j+1)%2==0))
{ sum+=((a[j+1][i]-
a[j][i]));
} } } avg=(float)sum/n; printf("\n\n Average waiting time = %f
msec",avg); sum=avg=0; for(j=0;j<n;j++) { i=1;
while(a[i][j]!=0)
{ i+=1; } sum+=a[i-1][j]; } avg=(float)sum/n; printf("\n\n
Average turnaround time = %f msec\n\n",avg);
}
```

OUTPUT:

```
[com39@localhost os]$ gcc ex5b.c
 [com39@localhost os]$ ./a.out
Enter the number of processes : 3
Enter the process time for process 1 : 4
 Enter the process time for process 2 : 6
Enter the process time for process 3 : 9
Enter the time slice : 2
0 P1 2
2 P2 4
4 P3 6
6 P1 8
8 P2 10
10 P3 12
12 P2 14
14 P3 16
16 P3 18
18 P3 19
Average waiting time = 7.333333 msec
Average turnaround time = 13.666667
msec [com39@localhost os]$
```

CONCLUSION:

Thus the C program using Round Robin scheduling has been executed successfully.

EX.No.:8	INTER PROCESS COMMUNICATION
DATE :	

OBJECTIVE:

Learn about Shared Memory and Inter Process Communication

OUTCOME:

Students become expertise to write a C coding to implement Shared memory and Inter Process Communication.

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of Inter Process communication using PIPES.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a child process using fork().
Step 3: Now close the read end of the parent using close().
Step 4: Write the data in the pipe using write().
Step 5: Now close the read end of the child using close().
Step 6: Read the data in the pipe using read().
Step 7: Display the string.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>  int
main() { int fd[2],child;
char a[10];
printf("Enter the string to enter into the
pipe:\t");  scanf("%s", a); pipe(fd);
child=fork();  if(!child) { close(fd[0]);
write(fd[1],a,5);  wait(0); } else {

close(fd[1]); read(fd[0],a,5); printf("The string
received from the pipe is %s",a);
} return
0; }
```

OUTPUT:

```
[com39@localhost os]$ gcc ex6.c [com39@localhost os]$ ./a.out
Enter the string to enter into the pipe: welcome! The string received from the pipe is
welcome! [com39@localhost os]$
```

CONCLUSION:

Thus the C program for interprocess communication using pipes was executed and the output was verified.

EX.No.:9	SEMAPHORES
DATE :	

OBJECTIVE:

To write a C program to implement semaphores.

OUTCOME:

Students become expertise to write a C coding to implement semaphores.

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of semaphores.

ALGORITHM:

Step 1: Start the program

Step 2: Get the number of jobs from the user

Step3: When job1 is processing, job 2 is also starts processing

Step 4: When job 1 enters critical section, next job starts processing

Step 5: When job1 comes out of the critical section, the other job enters the critical section

Step 6: The above 3 steps are performed for various programs

Step 7: End the program.

PROGRAM:

```
#include<stdio.h> main()
{ int i,a=1,h=2,n; printf("\n Enter the no of
jobs"); scanf("%d",&n); for(i=0;i<n;i++)
{ if(a==1) { printf("processing
%d ! \n", i+1); a++;
} if(h>1)
{
```

```
 if(i+2<=n)

 {

printf("\n processing %d            ! \n",i+2);

 }

 printf("\n Process %d Enters Critical section", i+1); printf("\n Process %d Leaves Critical
 section", i+1);

 } h+1;

 }

 }
```

OUTPUT:

Enter the no of jobs 2

processing 1. !

processing 2. !

Process 1 Enters Critical section

Process 1 Leaves Critical section

Process 2 Enters Critical section

Process 2 Leaves Critical section

CONCLUSION:

Thus the program to implement semaphores in c was executed and the output was verified.

EX.No.:10	BANKERS AOR DEAD LOCK AVOIDANCE
DATE :	

OBJECTIVE:

To implement banker's algorithm for Dead Lock Avoidance.

OUTCOME:

Students become expertise to write a C coding to implement banker's algorithm

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Banker's algorithm for deadlock avoidance.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the values of resources and processes.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

Step 5: Check whether it's possible to allocate.

Step 6: If it is possible then the system is in safe state.

Step 7: Else system is not in safety state.

Step 8: If the new request comes then check that the system is insafety.

Step 9: Or not if we allow the request.

Step 10: Stop the program.

Step 11: End

PROGRAM:

```
#include< stdio.h>  #include< conio.h>  void main() { int
allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],fla g[15];
int pno,rno,i,j,prc,count,t,total; count=0 clrscr();

printf("\n Enter number of process:");
scanf("%d",&pno);  printf("\n Enter
number of resources:");
```

```
scanf("%d",&rno); for(i=1;i<
=pno;i++)
{ flag[i]=0; } printf("\n Enter total numbers of each resources:");
 for(i=1;i<= rno;i++)
 scanf("%d",&tres[i]); printf("\n Enter Max resources for each process:")
; for(i=1;i<= pno;i++)
{ printf("\n for process %d:",i); for(j=1;j<= rno;j++)
scanf("%d",&max[i][j
]); } printf("\n Enter allocated resources for each process:"); for(i=1;i<=
pno;i++)
{ printf("\n for process %d:",i); for(j=1;j<= rno;j++) scanf("%d",&allocate
d[i][j]);
} printf("\n available resources:\n"); for(j=1;j<=
rno;j++)
{ avail[j]=0; total=0; for(i=1;i<=
pno;i++)
{
total+=allocated[i][j];
} avail[j]=tres[j]-total; work[j]=avail[
j];
printf(" %d \t",work[j]); } do
```

```
printf("|");  for(j=1;j<= rno;j++) { printf("%4d",need[i][j]);
} } prc=0;
for(i=1;i<= pno;i++)
{ if(flag[i]==0) {
prc=i; for(j=1;j<=
rno;j++) {
if(work[j]<
need[i][j]) {
Prc=0; break;
}
} } if(prc!=0) break; } if(prc!=0) {
printf("\n Process %d completed",i);
count++; printf("\n Available matrix:");
for(j=1;j<= rno;j++)
{
work[j]+=allocate d[prc][j]; allocated[prc][j]= 0; max[prc][j]=0; flag[prc]=1;
printf(" %d",work[j]);
}
}
}while(count!=pno&&prc
!=0);  if(count==pno)  printf("\nThe system
is in a safe state!!");  else printf("\nThe
system is in an unsafe state!!");  getch();
}

printf("|"); for(j=1;j<= rno;j++)
{
printf("%4d",need[i][j]);
```

```
} } prc=0;
for(i=1;i<=
pno;i++)
{ if(flag[i]==0) {
prc=i; for(j=1;j<=
rno;j++)
{ if(work[j]<
need[i][j])
{
Prc=0; break;
}
} } if(prc!=0)
break;
} if(prc!=0) { printf("\n Process %d completed",i); count++; printf("\n Available matrix:");
for(j=1;j<= rno;j++)
{   work[j]+=allocate   d[prc][j];   allocated[prc][j]=   0;   max[prc][j]=0;
flag[prc]=1; printf(" %d",work[j]);
}
.}
}while(count!=pno&&prc
!=0); if(count==pno) printf("\nThe system is in a safe state!!"); else
printf("\nThe system is in an unsafe state!!"); getch();
}
```

OUTPUT:

Enter number of process: 5

Enter number of resources: 3

Enter total numbers of each resource: 10 5 7

Enter Max resources for each process: for process 1:7 5

3 for process 2:3 2 2 for process 3:9 0 2 for process 4:2

2 2 for process 5:4 3 3

Enter allocated resources for each process: for process 1:0 1 0 for process 2:3 0

2 for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

available resources: 2 Allocated matrix Max 3

need 0

0	1	0|	7	5	3|	7	4	3
3	0	2|	3	2	2|	0	2	0
3	0	2|	9	0	2|	6	0	0
2	1	1|	2	2	2|	0	1	1
0	0	2|	4	3	3|	4	3	1

Process 2 completed

Available matrix: 5 3 2 Allocated matrix Max need

0	1	0|	7	5	3|	7	4	3
0	0	0|	0	0	0|	0	0	0
3	0	2|	9	0	2|	6	0	0
2	1	1|	2	2	2|	0	1	1
0	0	2|	4	3	3|	4	3	1

Process 4 Complete 3

Available matrix: 7 4 3 Allocated matrix Max need

0	1	0|	7	5	3|	7	4	
0	0	0|	0	0	0|	0	0	0
3	0	2|	9	0	2|	6	0	0
0	0	0|	0	0	0|	0	0	0
0	0	2|	4	3	3|	4	3	1

Process 1 completed

Available matrix: 7 5 3 Allocat
matrix

0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
3	0	2|	9	0	2|	6	0	0
0	0	0|	0	0	0|	0	0	0
0	0	2|	4	3	3|	4	3	1

ted mat ix Max need

Process 3 completed

Available matrix: 10 5 5 Alloca

0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
0	0	2|	4	3	3|	4	3	1

Process 5 completed

Available matrix: 10 5 7 The system is in a safe state!!

CONCLUSION:

Thus the program to implement banker's algorithm for Dead Lock Avoidance was executed and the output was verified

EX.No.:11	DEADLOCK DETECTION ALGORITHM
DATE :	

OBJECTIVE:

To implement an algorithm for Dead Lock Detection

OUTCOME:

Students become expertise to write a C coding to implement algorithm for deadlock detection

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:Implementation of Deadlock Detection.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the values of resources and processes. Step 3: Get the avail value.

Step 4: After allocation find the need value. Step 5: Check whether it's possible to allocate.

Step 6: If it is possible then the system is in safe state. Step 7: Else system is not in safety state.

Step 8: If the new request comes then check that the system is insafety. Step 9: Or not if we allow the request.

Step 10: Stop the program. Step 11: End

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

int       max[100][100];

int       alloc[100][100];

int need[100][100]; int

avail[100]; int n,r; void input();

void show();  void cal();


int main()

{
```

```
int i,j;
printf("********** Deadlock Detection Algorithm ***********\n");
input();
show();
cal();
getch();
return 0; }
void
input() {
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);  printf("Enter the no of resource
instances\t");  scanf("%d",&r); printf("Enter the
Max Matrix\n");  for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
{
for(k=0;k<r;k++)
{ avail[k]+=alloc[i][j]; finish[i]=1;
flag=1;
}
//printf("\nP%d",i); if(finish[i]==1)
{
i=n;
}}}}} } j=0; flag=0;
for(i=0;i<n;i++)
```

```
{
if(finish[i]==0)
{ dead[j]=i; j++;
flag=1;
} } if(flag==1) { printf("\n\nSystem is in Deadlock and the Deadlock process
are\n"); for(i=0;i<n;i++)
{
printf("P%d\t",dead[i]);
} } else { printf("\nNo
Deadlock Occur");
}
}
}
```

PROGRAM 2

```
#include <bits/stdc++.h> using namespace std; int
arrmax[100][100]; int alloc[100][100]; int need[100][100]; int
avail[100]; int n, r;

void input()
{
   int i, j;
   cout << "Enter the no of Processes\t";     cin >> n;
 cout << "Enter the no of resource instances\t";     cin >> r;     cout << "Enter the Max
Matrix\n";
for (i = 0; i < n; i++)
   {        for (j = 0; j < r; j++)
```

```
    {           cin >> arrmax[i][j];
    }
  }
  cout << "Enter the Allocation Matrix\n";    for (i = 0; i < n; i++)
  {       for (j = 0; j < r; j++)           cin >> alloc[i][j];
#include <bits/stdc++.h> using namespace std; int
arrmax[100][100]; int alloc[100][100]; int need[100][100]; int
avail[100]; int n, r;

void input()
{
  int i, j;
  cout << "Enter the no of Processes\t";    cin >> n;    cout << "Enter the no of
resource instances\t";    cin >> r;    cout << "Enter the Max Matrix\n";    for (i = 0; i <
n; i++)
  {       for (j = 0; j < r; j++)
    {           cin >> arrmax[i][j];
    }
  }
  cout << "Enter the Allocation Matrix\n";    for (i = 0; i < n; i++)
  {       for (j = 0; j < r; j++)           cin >> alloc[i][j];

    }
  }
  cout << "Enter the available Resources\n";    for (j = 0; j < r; j++)
  {       cin >> avail[j];
  } } void show()
{
  int i, j;
  cout << "Process\t Allocation\t Max\t Available\t";    for (i = 0; i < n; i++)
```

```
{       cout << "\nP" << i + 1 << "\t ";        for (j = 0; j < r; j++)
    {           cout << alloc[i][j] << " ";
    }       cout << "\t\t";        for (j = 0; j < r; j++)
    {
       cout << arrmax[i][j] << " ";
    }       cout << "\t ";       if (i == 0)

       for (j = 0; j < r; j++)               cout << avail[j] << " ";
   }
  } } void cal() {    int finish[100], temp, need[100][100], flag = 1, k, c1 = 0;    int dead[100];
int safe[100];
  int i, j;    for (i = 0; i < n; i++)
  {       finish[i] = 0;
  }
  //find need matrix    for (i = 0; i < n; i++)
  {       for (j = 0; j < r; j++)
    {           need[i][j] = arrmax[i][j] - alloc[i][j];
    }    }    while (flag)
  {       flag = 0;       for (i = 0; i < n; i++)

        int c = 0;           for
(j = 0; j < r; j++)
        {
          if ((finish[i] == 0) && (need[i][j] <= avail[j]))
          {                c++;
if (c == r)                {
for (k = 0; k < r; k++)
                {
avail[k] += alloc[i][j];
            finish[i] = 1;
```

{

```
                int c = 0;
              for (j = 0; j < r;
              j++)
          {
            if ((finish[i] == 0) && (need[i][j] <= avail[j]))
            {                c++;
if (c == r)                {
for (k = 0; k < r; k++)
                {
avail[k] += alloc[i][j];
             finish[i] = 1;
flag = 1;
                }
                //cout<<"\nP%d",i;
if (finish[i] == 1)
                {
i = n;
                }
              }
            }
          }
      }   }   j = 0;
flag = 0;    for (i = 0; i
< n; i++)
   {
      if (finish[i] == 0)
      {
dead[j] = i;
```

```
    }   }   if
(flag == 1)
  {
    cout << "\n\nSystem is in Deadlock and the Deadlock process are\n";
for (i = 0; i < n; i++)
    {
      cout << "P" << dead[i] << "\t";
    }
}
else
  {
    cout << "\nNo Deadlock Occur";
  } } int
main()
{
  int i, j;
  cout << "********** Deadlock Detection Algorithm **********\n";
  input();
show();
cal();
return 0; }
```

OUTPUT:

********** Deadlock Detection Algorithm **********

Enter the no of Processes 3

Enter the no of resource instances 3

Enter the Max Matrix

3 6 8

4 3 3

3 4 4

Enter the Allocation Matrix

3 3 3

2 0 3

1 2 4

Enter the available Resources

1 2 0

Process	Allocation	Max	Available
P1	3 3 3	3 6 8	1 2 0
P2	2 0 3	4 3 3	
P3	1 2 4	3 4 4	

system is in Deadlock and the Deadlock process are

P0 P1 P2

CONCLUSION:

Thus the program to implement banker's algorithm for Dead Lock Detection was executed and the output was verified.

EX.No.:12	IMPLEMENTATION OF THREADING & SYNCHRONIZATON
DATE :	

OBJECTIVE

Learn about Threads and synchronization applications.

OUTCOME:

Students become expertise to write a C coding to implement threading applications.

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of threading applications.

ALGORITHM:

Step 1: Start the program.

Step 2: Create number of threads.

Step 3: Counter is maintained for the user.

Step 4: Enter the logs of the jobs to start.

Step 5: Stop the program.

PROGRAM:

```
#include<st dio.h>

#include<str ing.h>

#include<pt hread.h>

#include<st dlib.h>

#include<un istd.h> pthread_t tid[2];

int counter; void*

doSomeThing(void *arg)

{ unsigned long i = 0; counter += 1; printf("\n Job %d started\n",

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

finished\n", counter); return NULL;

}

int main(void) { int i=0; int err; while(i < 2) { err =

pthread_create(&(tid[i]), NULL, &doSomeThing, NULL); if (err != 0)

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

pthread_join(tid[0], NULL);  pthread_join(tid[1], NULL);  return 0;

}
```

OUTPUT:

Job 1 started

Job 1 finished

Job 2 started

Job 2 finished

CONCLUSION:
Thus the C program for threading and synchronization applications was created and executed successfully.

EX.No.:13	IMPLEMENTATION OF PAGING TECHNIQUE
DATE :	

OBJECTIVE:

Learn about paging techniques in memory management.

OUTCOME:

Students become expertise to write a C coding to implement paging techniques PRE-REQUEST:
C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of Paging Technique of Memory management

ALGORITHM

Step 1: Start the program.

Step 2: Input the pagesize and initialize memsize of page to 15.

Step 3: Compute number of pages by dividing memsize by pagesize.

Step 4: For the computed pages, give the page frames values.

Step 5: Get the logical address value. Compute frameno and offset.

Step 6: Using the page frameno and offset, Generate the physical address.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>  void
main()
{
int memsize=15;
int pagesize,nofpage;
int p[100]; int
frameno,offset;  int
logadd,phyadd;
 int i;
int choice=0; printf("\nYour memsize is %d
",memsize);    printf("\nEnter  page  size:");
scanf("%d",&pagesize);
nofpage=memsize/pagesize;
for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i+1);  scanf("%d",&p[i]);
}

do
{
printf("\nEnter a logical address:"); scanf("%d",&logadd);
 frameno=logadd/pagesize;
 offset=logadd%pagesize
; phyadd=(p[frameno]*pagesize)+offset; printf("\nPhysical address is:%d",phyadd)
; printf("\nDo you want to continue(1/0)?:");
 scanf("%d",&choice);
}while(choice==1);
}
```

Output:

Your memsize is 15 Enter page size: 5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address: 3

Physical address is: 13

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

Enter a logical address: 1

Physical address is: 11

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

CONCLUSION:

Thus the c program for implementing paging technique of memory management using first fit was executed and the output was verified.

EX.No.:14	MEMORY MANAGEMENT SCHEME USING FIRST FIT
DATE :	

OBJECTIVE:

Learn about Memory management schemes.

OUTCOME:

Students become expertise to write a C coding to implement memory management schemes

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of memory management using FirstFit

ALGORITHM:

Step 1: Include the necessary header files required. Step 2: Declare the variables needed. Step 3: Read the number of blocks and the size of each blocks. Step 4: Read the number of process and the size of each process.
Step 5: Check if the process size is less than or equal to block size.
Step 6: If yes, assign the corresponding block to the current process.
Step 7: Else print the current process is not allocated.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define max 25

void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;  static
int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit"); printf("\nEnter the
number of blocks:");
scanf("%d",&nb); printf("Enter the
number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");  for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
```

```
printf("Enter the size of the files :-\n"); for(i=1;i<=nf;i++)

{

printf("File %d:",i);

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

}

for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)

{
if(bf[j]!=1)         //if bf[j] is not allocated

{

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

if(temp>=0) if(highest<temp)

{
ff[i]=j; highest=temp;


}

}
frag[i]=highest; bf[ff[i]]=1; highest=0;
}

printf("\nFile_no:\tFile_size
:\tBlock_no:\tBlock_size:\tFragement"); for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],fra g[i]);
getch();
}
```

INPUT
Enter the number of blocks: 3 Enter the number of files: 2
Enter the size of the blocks:- Block 1: 5

Block 2: 2

Block 3: 7
Enter the size of the files:- File 1: 1
File 2: 4

OUTPUT

File	Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

CONCLUSION:

Thus the program to implement memory management using first fit was executed and the output was verified

EX.No.:15	MEMORY MANAGEMENT SCHEME USING WORST FIT
DATE :	

OBJECTIVE

Learn about Memory management schemes.

OUTCOME:

Students become expertise to write a C coding to implement memory management schemes

PRE-REQUEST: C- Basics.

SAMPLE PROGRAM TITLE:

Implementation of memory management using Worst Fit

ALGORITHM:

Step 1: Include the necessary header files required.

Step 2: Declare the variables needed.

Step 3: Read the number of blocks and the size of each blocks.

Step 4: Read the number of process and the size of each process.

Step 5: Check if the process size is less than or equal to block size.

Step 6: If yes, assign the corresponding block to the current process.

Step 7: Else print the current process is not allocated.

.

PROGRAM

```
#include<stdio.h>

#include<conio.h> #define max 25


void main() { int frag[max],b[max],f[max],i,j,nb,nf,temp; static int

bf[max],ff[max]; clrscr();

printf("\n\tMemory Management Scheme - First Fit"); printf("\nEnter the number of
blocks:"); scanf("%d",&nb);

printf("Enter the number of files:"); scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n"); for(i=1;i<=nb;i++)

{ printf("Block

%d:",i);

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

} printf("Enter the size of the files :-

\n"); for(i=1;i<=nf;i++)

{ printf("File

%d:",i);

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

}

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

{

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

{ if(bf[j]!=1) { temp=b[j]-

f[i]; if(temp>=0)

{ ff[i]=j;

break;

}
```

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

INPUT:

Enter the number of blocks: 3 Enter the number of files: 2

Enter the size of the blocks:- Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:- File 1: 1

File 2: {

```
printf("Block
%d:",i);
scanf("%d",&b[i]);
} printf("Enter the size of the files :-
\n"); for(i=1;i<=nf;i++)
{ printf("File
%d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{ if(bf[j]!=1) { temp=b[j]-
f[i]; if(temp>=0)
{ ff[i]=j;
break;
}
```

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

INPUT:

E nter the number of blocks: 3 Enter the number of files: 2

Enter the size of the blocks:- Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:- File 1: 1

File 2:

OUTPUT

File No	File Size	Block No	Block Size	fragment
1	1	1	5	4
4	3	7	3	7

CONCLUSION:

Thus the program to implement memory management using worst fit was executed and the output was verified.