Ex: NO 1 INSTALLATION OF WINDOWS XP OPERATING SYSTEM

Aim:

To install Windows XP Operating System.

Description

Windows XP is a versatile O.S. which took over various laggings in the earlier O.S. given by Microsoft. One major component which really improves the performance and working is the availability of various device drivers which really make device operation as simple as plug-n-play. If the minimum system requirement is not fulfilled then you will not be able to install this O.S. on to your machine, however, Windows 98 might serve the purpose. Windows XP is in huge demand globally, let us learn how to load it before making us to work on it.

System Requirements

System Requirements	Minimum	Recommended
Processor	233MHz	300 MHz or higher
Memory	64 MB RAM	128 MB RAM or higher
Video adapter and Monitor	Super VGA (800 x 600) or higher resolution	
Hard drive disk free space	1.5 GB or higher	
Drives	CD-ROM drive or DVD drive	
Input Devices	Keyborad, Microsoft Mouse	
Sound	Speakers. Sound card, Head Phones	

Installation Steps

The following step by step procedure will help you to install Windows XP. The installation procedure is shown with the figure appears on your screen after doing a step.

1) Insert the Windows XP CD into your computer and restart.

2) If prompted to start from the CD, press SPACEBAR. If you miss the prompt (it only appears for a few seconds), restart your computer to try again.

3) Windows XP Setup begins. During this portion of setup, your mouse will not work, so you must use the keyboard. On the Welcome to Setup page, press ENTER.

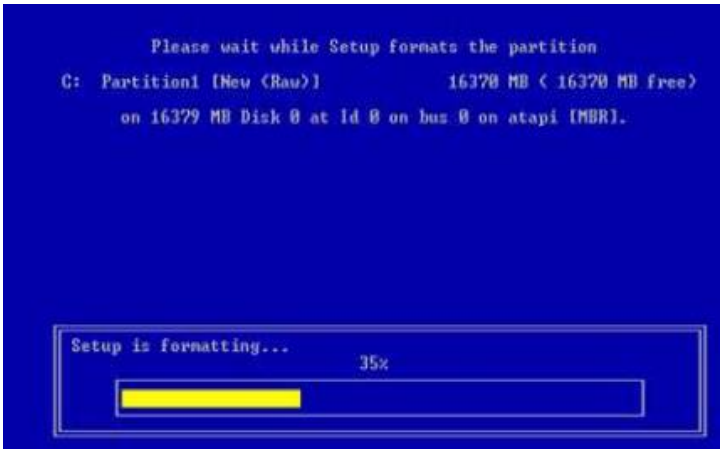
4) On the Windows XP Licensing Agreement page, read the licensing agreement. Press the PAGE DOWN key to scroll to the bottom of the agreement. Then press F8.

5) Next page enables you to select the hard disk drive on which Windows XP will be installed. Once you complete this step, all data on your hard disk drive will be removed and cannot be recovered. It is extremely important that you have a recent backup copy of your files before continuing. When you have a backup copy, press D, and then press L when prompted. This deletes your existing data.

6) Press ENTER to select Unpartitioned space, which appears by default.

7) Press ENTER again to select Format the partition using the NTFS file system, which appears by default.

8) Windows XP erases your hard disk drive using a process called formatting and then copies the setup files. You can leave your computer and return in 20 to 30 minutes



9) Windows XP restarts and then continues with the installation process. From this point forward, you can use your mouse. Eventually, the Regional and Language Options page appears. Click Next to accept the default settings. If you are multilingual or prefer a language other than English, you can change language settings after setup is complete.

10) On the Personalize Your Software page, type your name and your organization name. Some programs use this information to automatically fill in your name when required. Then, click Next.

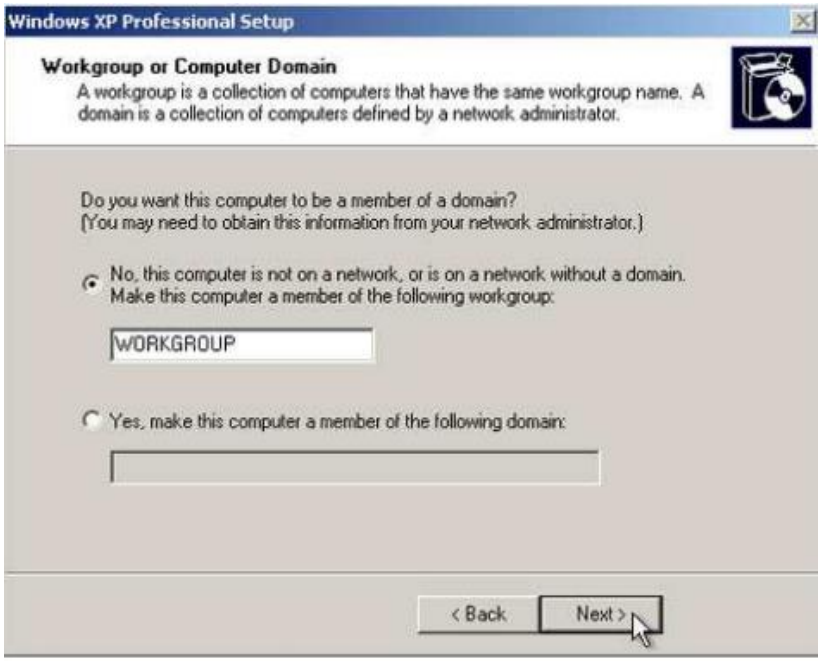
11) On the Your Product Key page, type your product key as it appears on your Windows XP CD case. The product key is unique for every Windows XP installation. Then, click Next.

12) On the Computer Name and Administrator Password page, in the Computer name box, type a name that uniquely identifies your computer in your house, such as FAMILYROOM or TOMS. You cannot use spaces or punctuation. If you connect your computer to a network, you will use this computer name to find shared files and printers. Type a strong password that you can remember in the Administrator password box, and then retype it in the Confirm password box. Write the password down and store it in a secure place. Click Next.

13) On the Date and Time Settings page, set your computer's clock. Then, click the Time Zone down arrow, and select your time zone. Click Next.

14) Windows XP will spend about a minute configuring your computer. On the Networking Settings page, click Next.

15) On the Workgroup or Computer Domain page, click Next.



16) Windows XP will spend 20 or 30 minutes configuring your computer and will automatically restart when finished. When the Display Settings dialog appears, click OK.

17) When the Monitor Settings dialog box appears, click OK.

18) The final stage of setup begins. On the Welcome to Microsoft Windows page, click Next.

19) On the Help protect your PC page, click Help protect my PC by turning on Automatic Updates now. Then, click Next

20) Windows XP will then check if you are connected to the Internet:

If you are connected to the Internet, select the choice that describes your network connection on the Will this computer connect to the Internet directly, or through a network? page. If you're not sure, accept the default selection, and click Next.

21) If you use dial-up Internet access, or if Windows XP cannot connect to the Internet, you can connect to the Internet after setup is complete. On the How will this computer connect to the Internet? page, click Skip.

22) Windows XP Setup displays the Ready to activate Windows? page. If you are connected to the Internet, click Yes, and then click Next. If you are not yet connected to the Internet, click No, click Next, and then skip to step 24. After setup is complete, Windows XP will

automatically remind you to activate and register your copy of Windows XP.

23) On the Ready to register with Microsoft? page, click Yes, and then click Next

24) On the Collecting Registration Information page, complete the form. Then, click Next.

25) On the Who will use this computer? page, type the name of each person who will use the computer. You can use first names only, nicknames, or full names. Then click Next. To add users after setup is complete or to specify a password to keep your account private, read Create and customize user accounts.

26) On the Thank you! page, click Finish.

27) Congratulations! Windows XP setup is complete. You can log on by clicking your name on the logon screen. If you've installed Windows XP on a new computer or new hard disk drive, you can now use the File and Settings Transfer Wizard to copy your important data to your computer or hard disk drive.

RESULT:

Thus the Windows Operating System is installed and Executed successfully.

Ex. No. : 2.a **UNIX COMMANDS AND SHELL PROGRAMMING**
DATE :

Aim:

To study the basic commands in Linux.

1. TASK **:** To display the system date and time.
COMMAND **:** date.
SYNTAX **:** date.
EXPLANATION: This command displays the current system date and time on the screen.
OUTPUT **:** Tue Jun 19 11:37:17 GMT 2007.

2. TASK **:** To display the current month.
COMMAND **:** date.
SYNTAX **:** date +%m.
EXPLANATION: This command displays the current month on the screen.
OUTPUT **:** 06.

3. TASK **:** To display the name of the current month.
COMMAND **:** date.
SYNTAX **:** date +%h.
EXPLANATION: This command displays the name of the current month on the screen.
OUTPUT **:** Jun.

4. TASK **:** To display the current system date.
COMMAND **:** date.
SYNTAX **:** date +%d.
EXPLANATION: This command displays the current system date on the screen.
OUTPUT **:** 19.

5. TASK **:** To display the current system date (year).
COMMAND **:** date.
SYNTAX **:** date +%y.
EXPLANATION: This command displays the current year on the screen.
OUTPUT **:** 07.

6. TASK **:** To display the current system time.
COMMAND **:** date.
SYNTAX **:** date +%H.
EXPLANATION: This command displays the current system time (in hours) on the screen.

OUTPUT **:** 11.

7. TASK **:** To display the current system time.
COMMAND **:** date.
SYNTAX **:** date +%M.
EXPLANATION: This command displays the current system time (in minutes) on the screen.
OUTPUT **:** 43.

8. TASK **:** To display the current system time.
COMMAND **:** date.
SYNTAX **:** date +%S.
EXPLANATION: This command displays the current system time (in seconds) on the screen.
OUTPUT **:** 15.

9. TASK **:** To display the calendar of the current month.
COMMAND **:** calendar.
SYNTAX **:** cal.
EXPLANATION: This command displays the calendar of the current month on the screen.
OUTPUT **:**

Jun 07

S	M	T	W	T	F	S
					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

10. TASK **:** To display user-defined message.
COMMAND **:** echo.
SYNTAX **:** echo “message”.
EXPLANATION: This command displays on the screen the argument of the echo command.
OUTPUT **:** echo “OS”.
➔ **OS**

11. TASK **:** To display the details of all users.
COMMAND **:** who.
SYNTAX **:** who.
EXPLANATION : This command lists the information about all the users who have logged on to that system.
OUTPUT **:**

12. TASK **:** To display the user detail.
COMMAND **:** who.
SYNTAX **:** whoami.
EXPLANATION : This command displays information about the current user of the

system on the screen.
OUTPUT **:** root.

13. TASK **:** To create a directory.
COMMAND **:** make directory.
SYNTAX **:** mkdir.
EXPLANATION : This command is used to create a new directory with the specified name.
EXAMPLE **:** mkdir student.
OUTPUT **:** The directory "**student**" is created.

14. TASK **:** To change directory.
COMMAND **:** change directory.
SYNTAX **: cd** directory name.
EXPLANATION : This command is used to switch from one directory to another.
EXAMPLE **:** cd staff.
OUTPUT **:** The directory "**staff**" is switched onto.

15. TASK **:** To delete a directory.
COMMAND **:** remove directory.
SYNTAX **: rmdir** directory name
EXPLANATION **:** This command is used to delete the specified directory.
EXAMPLE **:** rmdir student.
OUTPUT **:** The "**student**" directory is deleted.

16. TASK **:** To come out of a sub-directory.
COMMAND **:** change directory.
SYNTAX **: cd ..**
EXPLANATION : This command helps in switching to the main directory.
OUTPUT **:**

17. TASK **:** To list all the files and directories.
COMMAND **:** list.
SYNTAX **: ls**.
EXPLANATION : This command displays all the files and directories of the system.
OUTPUT **:**

18. TASK **:** To create a file.
COMMAND **:** cat.
SYNTAX **: cat>** file name.
EXPLANATION : This command leads to the creation of a new file with the specified file name and contents.
EXAMPLE **:** cat> wind.
OUTPUT **:** A null file called "**wind**" is created.

19. TASK **:** To view a file.
COMMAND **:** cat.
SYNTAX **: cat** file name.

EXPLANATION : This command displays the contents of the specified file.
EXAMPLE : cat wind.
OUTPUT : Contents of the file called "**wind**" will be displayed on the screen.

20. TASK : To copy a file.
COMMAND : copy.
SYNTAX : cp sourcefile destinationfile.
EXPLANATION : This command produces a copy of the source file and is stored in the specified destination file by overwriting its previous contents.
EXAMPLE : cp sun moon.
OUTPUT : The contents of "**sun**" file will be copied to the "**moon**" file.

21. TASK : To move a file.
COMMAND : move.
SYNTAX : mv sourcefile destinationfile.
EXPLANATION : After moving the contents of the source file into destination file, the source file is deleted.
EXAMPLE : mv sun moon.
OUTPUT : After copying contents from the "**sun**" file to "**moon**" file, the "**sun**" file is deleted.

22. TASK : To display / cut a column from a file.
COMMAND : cut.
SYNTAX : cut –c no. file name.
EXPLANATION : This command displays the characters of a particular column in the specified file.
EXAMPLE : cut –c3 moon.
OUTPUT : Those characters occurring in the 3rd column of the file called "**moon**" are displayed.

23. TASK : To delete a file.
COMMAND : remove.
SYNTAX : rm file name.
EXPLANATION : This command deletes the specified file from the directory.
EXAMPLE : rm sun.
OUTPUT : The file called "**sun**" will be deleted.

24. TASK : To retrieve a part of a file.
COMMAND : head.
SYNTAX : head -no. of rows file name.
EXPLANATION : This command displays the specified no. of rows form the top of the specified file.
EXAMPLE : head -1 sun.
OUTPUT : The first row of the file called "**sun**" is displayed.

25. TASK : To retrieve a file.
COMMAND : tail.
SYNTAX : **tail** -no. of rows file name.
EXPLANATION : This command displays the specified no. of rows form the bottom of the specified file.
EXAMPLE : tail -1 moon.
OUTPUT : The last row of the file called "**moon**" is displayed.

26. TASK : To sort the contents of a file.
COMMAND : sort.
SYNTAX : **sort** file name.
EXPLANATION : This command helps in sorting the contents of a file in ascending order.
EXAMPLE : sort win.
OUTPUT : The contents of the file "**win**" are displayed on the screen in a sorted order.

27. TASK : To display the no. of characters in a file.
COMMAND : word count.
SYNTAX : **wc** file name.
EXPLANATION : This command displays on the screen the no. of rows, words, and the sum of no**.** of characters and words.
EXAMPLE : wc ball.
OUTPUT : The no. of rows, words, and no. of characters present in the file "**ball**" are displayed.

28. TASK : To display the calendar of a year.
COMMAND : cal.
SYNTAX : **cal** year.
EXPLANATION : This command displays on the screen the calendar of the specified year.
EXAMPLE : cal 2007.
OUTPUT : The calendar of the year **2007** will be displayed.

29. TASK : To search for particular pattern .
COMMAND : grep.
SYNTAX : grep pattern filename.
EXPLANATION : This command searches and displays the specified content
EXAMPLE : grep null wind
OUTPUT : The desired pattern will be displayed.

30. TASK : To display list of files.
COMMAND : ls
SYNTAX : **ls**.
EXPLANATION : This command displays the list of files
EXAMPLE : ls
OUTPUT : The list of files will be displayed.

Result

Thus the Linux commands was studied.

Ex.no: 2b (i) **Simulation of ls command**

Date: **Aim:**

To simulate ls command using UNIX system calls.

Algorithm:

1.Store path of current working directory using getcwd system call.

2. Scan directory of the stored path using scandir system call and sort the resultant array of structure.

3. Display dname member for all entries if it is not a hidden file.

4. Stop.

Program:-

```
#include <stdio.h>
#include <dirent.h>
main()
{
        struct dirent **namelist;
        int n,i;
        char pathname[100];
        getcwd(pathname);
        n = scandir(pathname, &namelist, 0, alphasort);
        if(n < 0)
                printf("Error\n");
        else
                for(i=0; i<n; i++)
        if(namelist[i]->d_name[0] != '.')
                printf("%-20s", namelist[i]->d_name);
}
```

Result

Thus the filenames/subdirectories are listed, similar to ls command

Ex.no: 2b(ii) Simulation of grep command

Date:

Aim:

To simulate grep command using UNIX system call.

Algorithm:

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

Program:-

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main(int argc,char *argv[])
{
    FILE *fd;
    char str[100];
    char c;
    int i, flag, j, m, k;
    char temp[30];
    if(argc != 3)
    {
```

```
        printf("Usage: gcc mygrep.c –o mygrep\n");
        printf("Usage: ./mygrep <search_text> <filename>\n");
        exit(-1);
}
fd = fopen(argv[2],"r");
if(fd == NULL)
{
        printf("%s is not exist\n",argv[2]);
        exit(-1);
}
while(!feof(fd))
{
        i = 0;
        while(1)
        {
                c = fgetc(fd);
                if(feof(fd))
                {
                        str[i++] = '\0';
                        break;
                }
                if(c == '\n')
                {
                        str[i++] = '\0';
                        break;
                }
                str[i++] = c;
        }
        if(strlen(str) >= strlen(argv[1]))
        for(k=0; k<=strlen(str)-strlen(argv[1]); k++)
        {
```

```
                for(m=0; m<strlen(argv[1]); m++)
                temp[m] = str[k+m];
                temp[m] = '\0';
                if(strcmp(temp,argv[1]) == 0)
                {
                        printf("%s\n",str);
                        break;
                }
        }
    }
}
```

Result

Thus the program simulates grep command by listing lines containing the search text.

Ex.no:2b(iii) **Simulation of cp command**

Date:

Aim

To simulate cp command using UNIX system call.

Algorithm

1. Get source and destination *filename* as command-line argument.
2. Declare a buffer of size 1KB
3. Open the source file in readonly mode using open system call.
4. If file does not exist, then stop.
5. Create the destination file using creat system call.
6. If file cannot be created, then stop.
7. File copy is achieved as follows:
    a. Read 1KB data from source file and store onto buffer using read system call.
    b. Write the buffer contents onto destination file using write system call.
    c. If end-of-file then step 8 else step 7a.
8. Close source and destination file using close system call.
9. Stop.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#define SIZE 1024
main(int argc, char *argv[])
{
    int src, dst, nread;
    char buf[SIZE];
    if (argc != 3)
    {
```

```
            printf("Usage: gcc copy.c -o copy\n");
            printf("Usage: ./copy <filename> <newfile> \n");
            exit(-1);
        }
        if ((src = open(argv[1], O_RDONLY)) == -1)
        {
            perror(argv[1]);
            exit(-1);
        }
        if ((dst = creat(argv[2], 0644)) == -1)
        {
            perror(argv[1]);
            exit(-1);
        }
        while ((nread = read(src, buf, SIZE)) > 0)
        {
            if (write(dst, buf, nread) == -1)
            {
                printf("can't write\n");
                exit(-1);
            }
        }
        close(src);
        close(dst);
}
```

Result:

Thus a file is copied using file I/O. The cmp command can be used to verify that contents of both file are same

Ex.no: 3a **PROCESS MANAGEMENT USING SYSTEM CALLS**

Date: **Program using system call fork()**

Aim : To write the program to create a Child Process using system call fork().

Algorithm :

1. Declare the variable pid.
2. Get the pid value using system call fork().
3. If pid value is less than zero then print as “Fork failed”.
4. Else if pid value is equal to zero include the new process in the system‟s Fil. using execlp system call.
5. Else if pid is greater than zero then it is the parent process and it waits till the child completes using the system call wait()
6. Then print “Child complete”.

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc,char *arg[])
{
int pid;
pid=fork();
if(pid<0)
{
        printf("fork failed");
        exit(1);
```

```
}
else if(pid==0)
{
        execlp("whoami","ls",NULL);
        exit(0);
}
else
{
        printf("\n Process id is -%d\n",getpid());
        wait(NULL);
        exit(0);
}
}
```

Output:

```
[cse6@localhost Pgm]$ cc prog4a.c

[cse6@localhost Pgm]$ ./a.out
```

Result:

Thus the program was executed and verified successfully

Ex.no : 3b Program using system calls getpid() & getppid()

Date:

Aim :

To write the program to implement the system calls getpid() and getppid().

Algorithm :

1. Declare the variables pid , parent pid , child id and grand chil id.
2 .Get the child id value using system call fork().
3.If child id value is less than zero then print as “error at fork() child”.
4 .If child id !=0 then using getpid() system call get the process id.
5. Print “I am parent” and print the process id.
6 .Get the grand child id value using system call fork().
7 .If the grand child id value is less than zero then print as “error at fork() grand child”.
8. If the grand child id !=0 then using getpid system call get the process id.
9 .Assign the value of pid to my pid.
10 .Print “I am child” and print the value of my pid.
11 .Get my parent pid value using system call getppid().
12. Print “My parent’s process id” and its value.
13. Else print “I am the grand child”.
14 .Get the grand child’s process id using getpid() and print it as “my process id”.
15. Get the grand child‟s parent process id using getppid() and print it as “my parent’s process id

<u>System calls used</u> :

1.getpid( )

Each process is identified by its id value**.** This function is used to get the id value of a

particular process.

2.getppid( )

Used to get particular process parent‟s id value.

3.perror( )

Indicate the process error.

Program :

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main( )
{
int pid;
pid=fork( ); if(pid==-1)
{
        perror(“fork failed”); exit(0);
}
if(pid==0)
{
        printf(“\n Child process is under execution”);
        printf(“\n Process id of the child process is %d”, getpid()); printf(“\n
        Process id of the parent process is %d”, getppid());
}
else
{
        printf(“\n Parent process is under execution”);
        printf(“\n Process id of the parent process is %d”, getpid());
        printf(“\n Process id of the child process in parent is %d”, pid());
        printf(“\n Process id of the parent of parent is %d”, getppid());
}
return(0);
```

}

Output:

Child process is under execution

Process id of the child process is 9314

Process id of the parent process is 9313 Parent

process is under execution

Process id of the parent process is 9313

Process id of the child process in parent is 9314 Process

id of the parent of parent is 2825

Result:

Thus the program was executed and verified successfully

Ex.no:3c Program using system calls wait( ) & exit( )

Date:

Aim :

To write the program to implement the system calls wait( ) and exit( ).

Algorithm :

1 : Declare the variables pid and i as integers.

2 : Get the child id value using the system call fork().

3 : If child id value is less than zero then print “fork failed”.

4 : Else if child id value is equal to zero , it is the id value of the child and then start the child process to execute and perform Steps 6 & 7.

5 : Else perform Step 8.

6 : Use a for loop for almost five child processes to be called. Step

7 : After execution of the for loop then print “child process ends”.

8 : Execute the system call wait( ) to make the parent to wait for the child process to get over.

9 : Once the child processes are terminated , the parent terminates and hence print “Parent process ends”.

10 : After both the parent and the chid processes get terminated it execute the wait( ) system call to permanently get deleted from the OS.

System call used :

1. fork ( )

Used to create new process. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.

Syntax: fork ( )

2. wait ( )

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

Syntax: wait (NULL)

3. exit ( )

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

Syntax: exit(0)

Program :

```
#include<stdio.h>
#include<unistd.h>
main( )
{
        int i, pid;
        pid=fork( );
        if(pid== -1)
        {
                perror("fork failed");
                exit(0);
        }
        else if(pid==0)
        {
                printf("\n Child process starts");
                for(i=0; i<5; i++)
                {
                        printf("\n Child process %d is called", i);
                }
                printf("\n Child process ends");
```

```
        }
        else
        {
                wait(0);
                printf("\n Parent process ends");
        }
        exit(0);
}
```

Output:

Child process starts Child
process 0 is called Child
process 1 is called Child
process 2 is called Child
process 3 is called Child
process 4 is called Child
process ends Parent process
ends

Result:

Thus the program was executed and verified successfully

Ex.no: 3d Program using system call stat( ) and close()

Date:

Aim:

To write the program to implement the system call stat( ) and close().

Algorithm :

1 : Include the necessary header files to execute the system call stat( ).

2 : Create a stat structure thebuf. Similarly get the pointers for the two structures passwd and group.

3 : Declare an array named path[20] of character type to get the input file along with its extension and an integer i.

4 : Get the input file along with its extension.

5 : If not of stat of a particular file then do the Steps 6 to 18.

6 : Print the file‟s pathname as file‟s name along with its extension.

7 : To check the type of the file whether it is a regular file or a directory use S_ISREG( ).

8 : If the above function returns true value the file is an regular file else it is directory.

9 : Display the file mode in octal representation using thebuf.st_mode.

10 : Display the device id in integer representation using thebuf.st_dev.

11 : Display the user id in integer representation using thebuf.st_uid.

12 : Display the user‟s pathname.

13 : Display the group id in integer representation using thebuf.st_gid.

14 : Display the group‟s pathname.

15 : Display the size of the file in integer representation using thebuf.st_size.

16 : Display the last access in time and date format using ctime(&thebuf.st_atime).

17 : Display the last modification in time and date format using

ctime(&thebuf.st_atime).

18 : Display the last status in time and date format using

ctime(&thebuf.st_atime).

19 : If Step 5 fails then print “Cannot get the status details for the given file”.

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
main()
{
        int fd1,fd2,n;
        char source[30],ch[5];
        struct stat s,t,w;
        fd1=creat("text.txt",0644);
        printf("Enter the file to be copied\n");
        scanf("%s",source);
        fd2=open(source,0);
        if(fd2==-1)
        {
                perror("file doesnot exist");
                exit(0);
        }
        while((n=read(fd2,ch,1))>0)
                write(fd1,ch,n);
                close(fd2);
                stat(source,&s);
                printf("Source file size=%d\n",s.st_size);
                fstat(fd1,&t);
                printf("Destination file size =%d\n",t.st_size);
```

```
        close(fd1);
    }
```

Result:

Thus the program was executed successfully.