K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: B2	**Roll No.: 16010121194**
Experiment No. 03	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Shell Programming and system calls

AIM: To study the shell script and write the program using shell.

Expected Outcome of Experiment:

CO 1. To introduce basic concepts and functions of operating systems.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **William Stallings "Operating Systems" Person, Seventh Edition Edition.**
3. **Sumitabha Das " UNIX Concepts & Applications", McGraw Hill Second Edition.**

Pre Lab/ Prior Concepts:

The shell provides you with an interface to the UNIX system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell Scripts
The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign, #, describing the steps.

Steps to create a Shell Script:
create a file using any text editor say vi, gedit, nano etc
1.$ vi filename

2.Insert the script/ commands in file and save the file to execute the file we need to give execute permission to the file
3.$ chmod 775 filename
4.Now execute the above file using any of following methods:
$ sh filename
OR
$ ./filename

NOTE: Before adding anything to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example –
#!/bin/sh.

Description of the application to be implemented:

1. Write a shell Script that accepts two file names as command line arguments and compare two file contents and check whether contents are same or not. If they are same, then delete second file.

2. Write a shell script that accepts integer and find the factorial of the number.

3. Write a shell script for adding users.

4. Write a shell script for counting no of logged in users.

5. Write a shell script for counting no of processes running on system

Program for System Call:

1. Write a Program for creating process using System call (E.g fork())
Create a child process. Display the details about that process using getpid and getppid functions. In a child process, Open the file using file system calls and read the contents and display.

Implementation details: (printout of code / screen shot)

1.

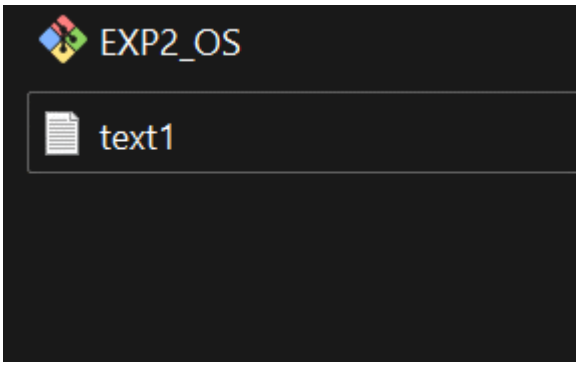
Code:

```
touch text1.txt
touch text2.txt
echo ""
echo "Contents of first text are:"
echo ""
cat text1.txt
echo ""
echo ""
echo "Contents of second text are:"
echo ""
cat text2.txt
echo ""
if cmp -s "text1.txt" "text2.txt"
then
echo ""
echo "Content of both files are same, thus we delete text2."
echo ""
rm text2.txt
else
echo ""
echo "Content of both files are different."
echo ""
fi
```

Output:

```
uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/Col
$ ./EXP2_OS.sh

Contents of first text are:

Text 1 content

Contents of second text are:

Text 2 Content

Content of both files are different.
```

```
uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/College
$ ./EXP2_OS.sh

Contents of first text are:

New Content

Contents of second text are:

New Content

Content of both files are same, thus we delete text2.
```



2. Code:

```
echo "Enter a number: "
read num

fact=1

for((i=2;i<=num;i++)){
    fact=$((fact*i))
}

echo "The factorial of " $num "is" $fact
```

Output:

```
uditi@LAPTOP-07TURRLK MINGW64 ~
$ cd 'C:\Users\uditi\OneDrive\Desktop\College stuff\Homework\OS'

uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/College stuff/Ho
$ ./EXP2_OS.sh
Enter a number:
5
The factorial of  5 is 120

uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/College stuff/Ho
$
```

3.

```
uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/
$ git config --global user.name
Peng Sinha

uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/
$ git config --global user.name "Uditi Sinha"

uditi@LAPTOP-07TURRLK MINGW64 ~/OneDrive/Desktop/
$ git config --global user.name
Uditi Sinha
```

4.

Code:

```
echo who | wc -l
```

Output:

```
uditi@LAPTOP-07TURRLK MINGW64 ~/One
$ ./EXP2_OS.sh
1
```

5. Code:

```
ps aux | wc -l | awk '{ print $1 - 1}'
```

Output:

```
uditi@LAPTOP-07TURRLK MINGW64 ~
$ ./EXP2_OS.sh
5
```

System Call Code:

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(){

    if(fork() == 0){
        printf("Child process ID %d\n",getpid());

        int file_descriptor = open("file1.txt", O_RDONLY);
        if (file_descriptor == -1) {
            perror("Error opening file");
            return 1;
        }

        off_t file_size = lseek(file_descriptor, 0, SEEK_END);
        lseek(file_descriptor, 0, SEEK_SET);

        char *buffer = (char *)malloc(file_size + 1);
        if (buffer == NULL) {
            perror("Memory allocation error");
            close(file_descriptor);
            return 1;
        }

        ssize_t bytes_read = read(file_descriptor, buffer, file_size);
```

```
        if (bytes_read != file_size) {
            perror("Read error");
            free(buffer);
            close(file_descriptor);
            return 1;
        }
        buffer[file_size] = '\0';

        close(file_descriptor);

        printf("%s", buffer);

        free(buffer);



    }

    else{
        printf("Parent processID %d\n",getpid());
    }

}
```

Output:

```
/tmp/XtDkG1aUuN.o
Parent processID 69345
Child process ID 69346
```

Conclusion :

In this lab, I learned about different commands in BASH and how to write code in shell. I also understood about different users, processes and fork.

Post Lab Descriptive Questions

1. What are the different types of commonly used shells on a typical linux system?

Ans)

On a typical Linux system, there are several commonly used shells, each with its own features and capabilities. Here are some of the most popular ones:

1. Bash (Bourne-Again Shell): Bash is the default shell on many Linux distributions. It's known for its extensive features, scripting capabilities, and compatibility with the Bourne shell. It offers command-line editing, history, tab completion, and support for job control, making it a versatile choice for both interactive use and scripting.

2. Zsh (Z Shell): Zsh is an interactive shell that offers advanced customization options and features, including powerful tab completion, spelling correction, customizable prompts, and extensive plugin support. It's particularly popular among power users and developers.

3. Fish (Friendly Interactive Shell): Fish is designed to be user-friendly with features like syntax highlighting, autosuggestions, and intuitive tab completion. It aims to provide an improved interactive experience out of the box, especially for beginners.

4. Dash: Dash is a minimalistic shell designed for speed and efficiency. It's often used as the default `/bin/sh` shell on many systems due to its lightweight nature. It doesn't have all the interactive features of other shells but is well-suited for scripting.

5. Ksh (KornShell): Ksh is a shell that combines features from the Bourne shell and the C shell. It offers advanced scripting capabilities, job control, and other features found in shells like Bash. There are different versions of Ksh, including ksh88 and ksh93.

6. Csh (C Shell): Csh is known for its C-like syntax and features like history and interactive editing. It's used by some users, but its syntax

and limitations have led many to prefer other shells like Bash or Zsh.

7. Tcsh (Enhanced C Shell): Tcsh is an enhanced version of the C shell with additional features like command-line editing, history, and better scripting capabilities. It's an improved alternative to the original C shell.

These are just a few examples of the commonly used shells on Linux systems. The choice of shell often depends on personal preference, scripting needs, and the features you require for your tasks.

2. How do you find out what's your shell?

Ans)
We can find out which shell we are currently using by using the `echo` command along with the special variable `$SHELL`. Open a terminal and type the following command:

echo $SHELL

After running this command, the terminal will display the path to the currently active shell executable. For example, if the output is `/bin/bash`, it means we are using the Bash shell. If the output is `/bin/zsh`, we are using the Zsh shell, and so on.

3. List the advantages and disadvantages of shell scripting.

Ans)

Shell scripting, like any programming approach, comes with its own set of advantages and disadvantages.

Advantages of Shell Scripting:

1. Ease of Use: Shell scripting provides a simple and intuitive way to automate tasks and interact with the operating system. It uses commands that are similar to those used on the command line.

2. Rapid Prototyping: Writing and testing shell scripts is usually faster compared to compiled languages. This makes shell scripting great for quickly prototyping solutions.

3. Access to Command-Line Utilities: Shell scripts can leverage the vast array of command-line utilities available in Unix-like systems, allowing you to perform complex tasks without writing a lot of code.

4. Task Automation: Shell scripts are particularly well-suited for automating repetitive tasks, managing files, and handling system administration tasks like backups and updates.

5. Interactive Debugging: Debugging shell scripts can often be done interactively by adding print statements or using tools like `echo` or `printf` to trace the execution.

6. Integration with System Utilities: Shell scripts can easily interact with other system utilities and services, making them suitable for system administrators and DevOps tasks.

Disadvantages of Shell Scripting:

1. Limited Performance: Shell scripts are interpreted, which can result in slower execution compared to compiled languages. They might not be suitable for computationally intensive tasks.

2. Limited Data Structures: Shell scripts have limited support for complex data structures like arrays or dictionaries, which can make managing and processing data more challenging.

3. Limited Portability: Shell scripts might not be fully portable across different shell implementations or operating systems. They often rely on specific commands that could differ.

4. Security Concerns: Writing secure shell scripts can be challenging due to the need to properly sanitize inputs and manage permissions. Poorly written scripts can pose security risks.

5. Complex Logic: While shell scripting is great for simple tasks and automation, as scripts become more complex, they can become difficult to read, maintain, and debug.

6. Less Extensive Error Handling: Shell scripts often lack advanced error handling features found in higher-level programming languages, making it harder to catch and manage errors.

7. Limited Testing: Automated testing of shell scripts can be more challenging compared to traditional programming languages, making it harder to ensure code quality.

Date: _____ **Signature of faculty in-charge**