

Viva Preparation for Operating Systems Course LAB

Bahria University, Lahore Campus

Department of Computer Sciences

Operating System Lab Journals

Course Code: CSL-320

Spring 2024

Name: [AFFAN AHMAD]

Enroll No: [03-134221-003]

Viva Objective:

The objective of this viva is to assess the understanding of core concepts and practical applications covered in the Operating Systems Lab. This includes threads, process scheduling, memory management, inter-process communication, signal handling, and basic shell programming.

Lab Journal 01

Date: 21-09-2023

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To understand basic concepts of Operating System.

Lab Tasks:

1. What is an Operating System?

An operating system (OS) is a software program that serves as the intermediary between a computer's hardware and its users. It manages and coordinates various hardware components, such as the CPU, memory, storage devices, and peripherals, to enable the execution of software applications. Additionally, an operating system provides a user interface, either through a graphical interface (GUI) or a command-line interface (CLI), allowing users to interact with the computer system, run programs, manage files, and

perform other tasks efficiently. Examples of popular operating systems include Microsoft Windows, macOS, Linux, and Unix.

2. Which OS is being used in the Lab?

However, common operating systems used in laboratory settings include various versions of Windows, macOS, and Linux, depending on the specific requirements and preferences of the lab and its users. If you have access to the lab's computers, you can typically determine the operating system by checking the system settings or by observing the user interface.

3. Install VMWARE and UBUNTU on your laptops.

4. What is a Virtual Machine? Differentiate between Guest and Host OS.

A virtual machine (VM) is a software-based emulation of a physical computer that runs an operating system (OS) and applications. It allows multiple virtualized operating systems to run concurrently on a single physical machine, known as the host machine. Each virtual machine is isolated from the others and operates as if it were a standalone computer with its own CPU, memory, storage, and network interfaces.

Differentiating between the guest and host OS:

Host OS:

The host OS is the operating system installed directly on the physical hardware of the computer.

It manages the hardware resources of the host machine, such as CPU, memory, disk, and network interfaces.

Examples of host OS include Windows, macOS, or Linux distributions.

Guest OS:

The guest OS is the operating system installed and running within a virtual machine.

It operates independently of the host OS and other guest OS instances, unaware that it's running within a virtualized environment.

Each virtual machine can have its own guest OS, which can be different from the host OS or other guest OS instances.

Examples of guest OS include various versions of Windows, Linux distributions, macOS (on supported virtualization platforms), and other operating systems.

Lab Journal 02

Date: 2/29/2024

Max Marks: 20

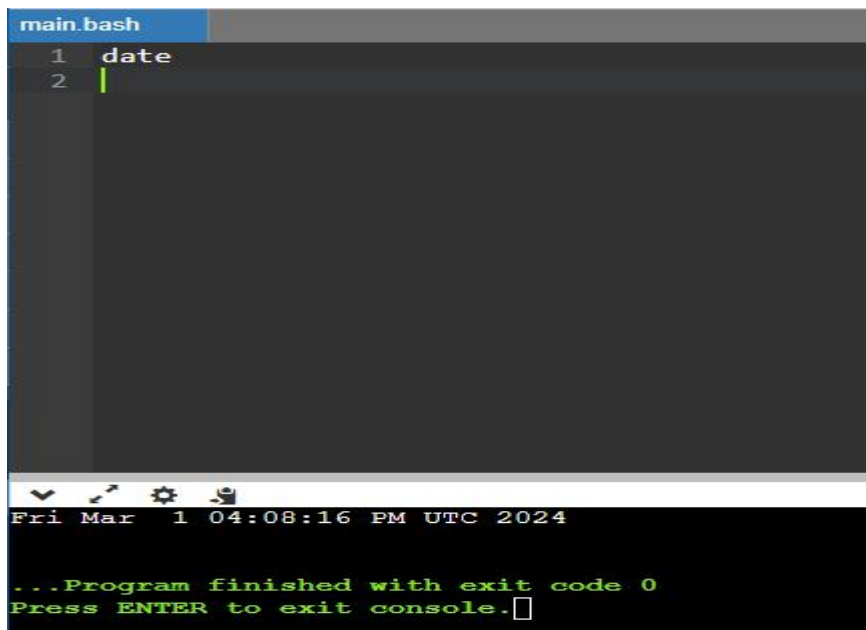
Faculty's Name: Abdullah

Objective(s):

- To study and execute the commands in Linux.

Lab Tasks:

1. Execute the Date Commands and write the output.



```
main.bash
1 date
2 |
...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows a terminal window titled 'main.bash'. The command 'date' is entered on line 1. On line 2, there is a vertical bar '|' indicating the command is being executed. Below the terminal window, a status bar shows the date and time: 'Fri Mar 1 04:08:16 PM UTC 2024'. At the bottom, a message states: '...Program finished with exit code 0' and 'Press ENTER to exit console.' followed by a small square icon.

```
main.bash
1 date +%H-%M-%S
2

16-25-59

...Program finished with exit code 0
Press ENTER to exit console.
```

```
[root@localhost ~]# cal 2 2024
    February 2024
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
```

2. Execute the mentioned LINUX Commands and generate output.

```
[root@localhost ~]# echo text
text
[root@localhost ~]# echo i am a student of bahria university
i am a student of bahria university
[root@localhost ~]#
```

```
anshul@anshul-VirtualBox:~$ banner 1234567890
# ##### # ##### # ##### # ##### # ##### #
## # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # #
# ##### # ##### # ##### # ##### # # #
# # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # #
##### # ##### # ##### # ##### # ##### #
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
[anshul@anshul-VirtualBox:~$]

[anshul@localhost ~]# who -b -H
NAME LINE TIME PID COMMENT
[anshul@localhost ~]#
```

```
[root@localhost ~]# whoami
root
[root@localhost ~]# echo "12+5" | bc
17
[root@localhost ~]#
```

3. Execute the below File Commands and write the output.

```
[root@localhost ~]# cat > affan1
hi i am affan

^C
[root@localhost ~]# cat < affan1
hi i am affan

[root@localhost ~]#
```

```
[root@localhost ~]# cat -s affan1
hi i am affan

[root@localhost ~]#
```

```
[root@localhost ~]# cat affan1 >> affan2
[root@localhost ~]# cat < affan1
hi i am affan

[root@localhost ~]# cat < affan2
i am a student of bahria university
hi i am affan
```

```

[root@localhost ~]# cat -n affan1
 1 hi i am affan
 2
 3
~
[root@localhost ~]# cp affan2 affan1
[root@localhost ~]# cat < affan1

i am a student of bahria university
hi i am affan


[root@localhost ~]# mv affan2 affan1
[root@localhost ~]# cat < affan1

i am a student of bahria university
hi i am affan


[root@localhost ~]# cat < affan2
sh: affan2: No such file or directory
[root@localhost ~]#

[root@localhost ~]# rm affan1
[root@localhost ~]# cat < affan1
sh: affan1: No such file or directory
[root@localhost ~]#

[root@localhost ~]# cat < affan1
hi

[root@localhost ~]# wc affan1
2 1 5 affan1
[root@localhost ~]#

```

4. Execute FILTERS AND PIPES commands and write the output.

```
[root@localhost ~]# head affan1
hi
```

```
[root@localhost ~]#
```

```
[root@localhost ~]# tail affan1
hi
```

```
[root@localhost ~]#
```

```
[root@localhost ~]# sort affan1
```

```
hi
```

```
[root@localhost ~]#
```

```
[root@localhost ~]# echo 1+1 | bc
```

```
2
```

```
[root@localhost ~]#
```

```
[root@localhost ~]# cat affan1 | tr [a-z] [A-Z]
```

```
HI
```

Lab Journal 03

Date: 03-07-2024

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- Understanding of Shell Programming.
- Understanding of variables, loops, operators.

Lab Tasks:

1. Write the output of programs for LINUX variables.

```
7  
8 echo "Variable Name : "  
9 Name="Operating System"  
10 echo $Name
```

Variable Name :
Operating System

...Program finished with exit code 0
Press ENTER to exit console.

```
7  
8 echo "Enter your name"  
9 read NAME  
10 echo "Enter your age"  
11 read AGE  
12 echo "Enter your enrollment"  
13 read ENROLLMENT  
14 echo "Hello " $NAME, "Your age is :" $AGE "Your enrollment is :" $ENROLLMENT
```

Enter your name
affan ahmad
Enter your age
20
Enter your enrollment
03-134221-003
Hello affan ahmad, Your age is : 20 Your enrollment is : 03-134221-003

...Program finished with exit code 0
Press ENTER to exit console.


```
7 echo "Readonly Variables"
8     Name="David"
9     readonly Name
10    Name="John"
```

input

Readonly Variables
main.bash: line 10: Name: readonly variable

...Program finished with exit code 1
Press ENTER to exit console.

```
6      #Variable Assignment & Accessing
7      echo "Unset Variables :"
8      read Name
9      unset Name
10     echo $Name
```

input

Unset Variables :
affan

...Program finished with exit code 0
Press ENTER to exit console.

```
7      echo "File Name = $0"
8      echo "First Parameter = $1"
9      echo "Second Parameter = $2"
10     echo "Quoted Values = $@"
11     echo "Quoted Values = $*"
12     echo "Total number of parameters = $#"
```

input

File Name = main.bash
First Parameter =
Second Parameter =
Quoted Values =
Quoted Values =
Total number of parameters = 0
0

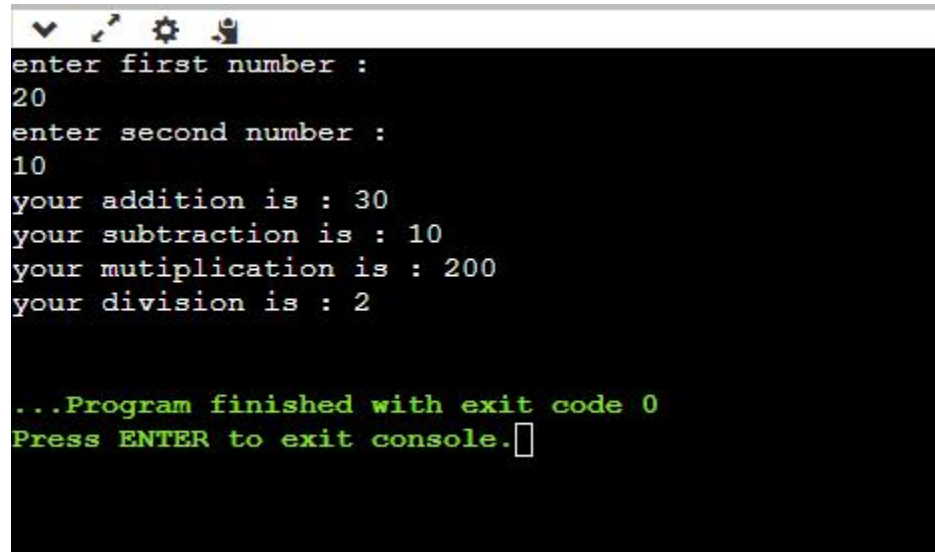
...Program finished with exit code 0
Press ENTER to exit console.

Write a program to calculate the addition, subtraction, multiplication, and division of numbers. **echo "enter first number :"**

read value1

```
echo "enter second number :"  
  
read value2  
  
var=$((expr $value1 + $value2))  
  
echo "your addition is : "$var  
  
var=$((expr $value1 - $value2))  
  
echo "your subtraction is : "$var  
  
var=$((expr $value1 \* $value2))  
  
echo "your mutiplication is : "$var  
  
var=$((expr $value1 / $value2))  
  
echo "your division is : "$var
```

Output:



```
enter first number :  
20  
enter second number :  
10  
your addition is : 30  
your subtraction is : 10  
your mutiplication is : 200  
your division is : 2  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

3.1 Write a program that compares two numbers if a is greater than b it displays “a is greater than b”, otherwise it displays that ‘a is not equal to b’.

Program:

```
echo -n "Enter a number: "
```

```
read VAR1

echo -n "Enter b number: "

read VAR2

if [[ $VAR1 -gt $VAR2 ]]
then

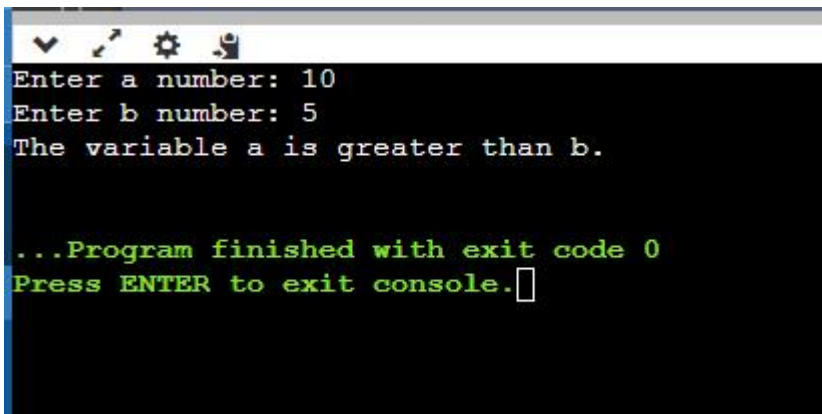
    echo "The variable a is greater than b."

else

    echo "The variable b equal or less than a."

fi
```

Output:



```
Enter a number: 10
Enter b number: 5
The variable a is greater than b.

...Program finished with exit code 0
Press ENTER to exit console.
```

3.2 Write a program that compares two numbers to check whether the numbers are equal, a is greater than b or a is less than b.

Program:

```
echo -n "Enter a number: "

read VAR1

echo -n "Enter b number: "

read VAR2
```

```
if [[ $VAR1 -eq $VAR2 ]]
then

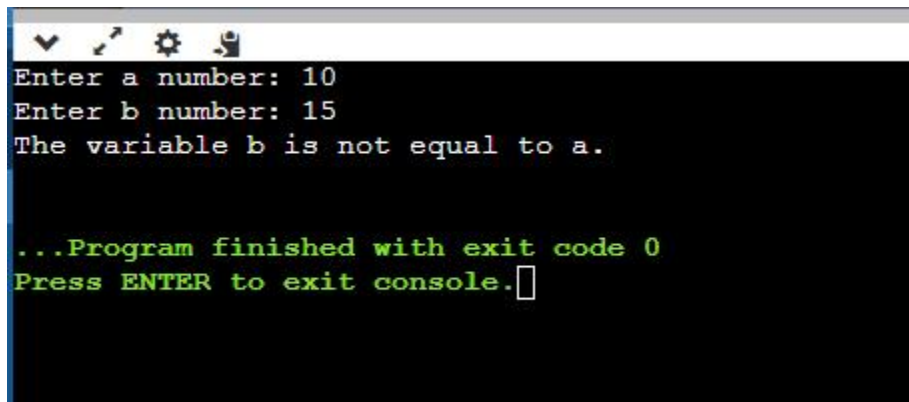
    echo "The variable a is equal to b."

else

    echo "The variable b is not equal to a."

fi
```

Output:



```
Enter a number: 10
Enter b number: 15
The variable b is not equal to a.

...Program finished with exit code 0
Press ENTER to exit console.
```

4 Write a program using “case” that inputs a fruit from the user and displays “Apple pie” on the input of apple, “I like banana” on the input of banana and “New Zealand famous for kiwi” on the input of kiwi.

Program:

```
echo -n "enter fruit name : "

read FRUIT

case $FRUIT in

    "apple" | "APPLE")

        echo -n "APPLE PIE "

        ;;
```

```
"BANANA" | "banana")
```

```
echo -n "I LIKE BANANA "
```

```
;;
```

```
"KIWI" | "kiwi")
```

```
echo -n "NEW ZEALAND FAMOUS FOR KIWI"
```

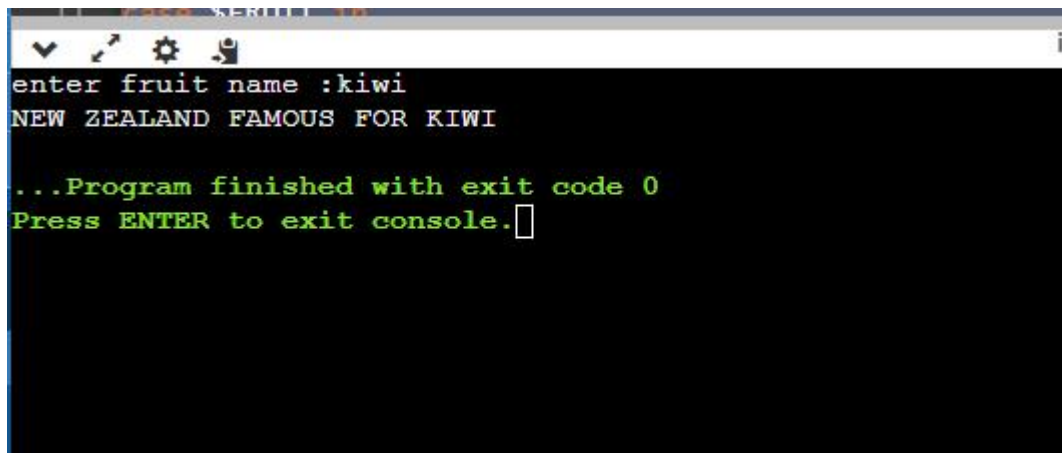
```
;;
```

```
*)
```

```
echo -n "Invalid FRUIT";;
```

```
esac
```

Output:



```
enter fruit name :kiwi
NEW ZEALAND FAMOUS FOR KIWI

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Journal 04

Date: 12-10-2023

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To study loops, arrays, strings, file testing, positional parameters.

Lab Tasks:

1. Write a program to display the numbers from 10 to 20 in reverse order using for loop.

Code

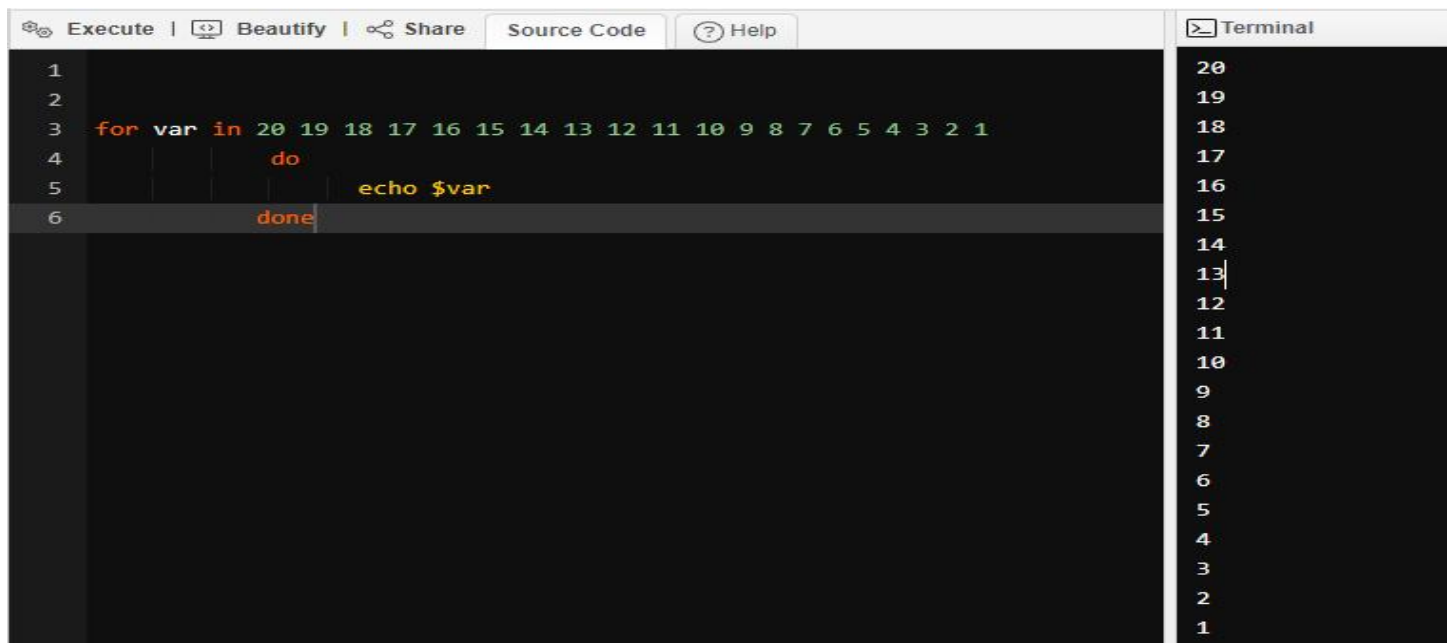
```
for var in 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

```
do
```

```
    echo $var
```

```
done
```

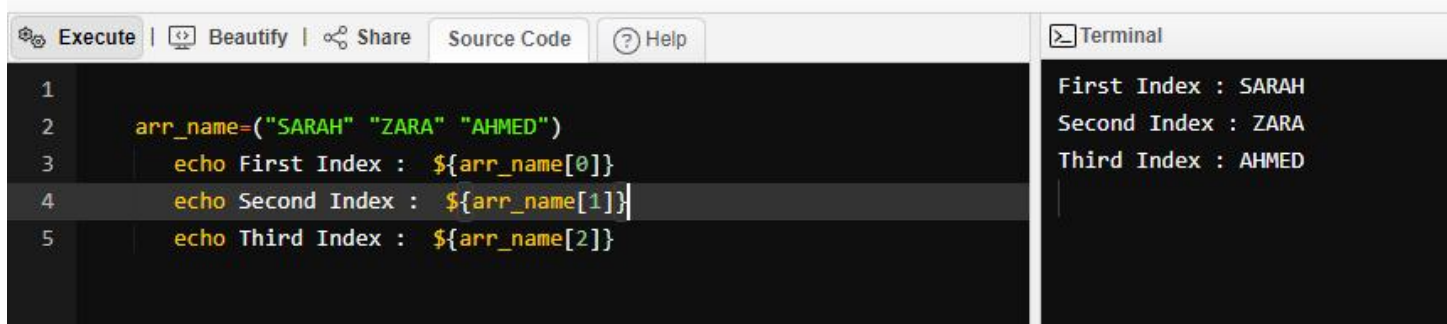
Output



The screenshot shows a code editor with a dark theme. The editor has tabs for 'Execute', 'Beautify', 'Share', 'Source Code', and 'Help'. The code is as follows:

```
1  
2  
3 for var in 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1  
4 do  
5     echo $var  
6 done
```

The 'Terminal' tab on the right shows the output of the script, which is a list of numbers from 20 down to 1, each on a new line.



The screenshot shows a code editor with a dark theme. The editor has tabs for 'Execute', 'Beautify', 'Share', 'Source Code', and 'Help'. The code is as follows:

```
1  
2 arr_name=("SARAH" "ZARA" "AHMED")  
3 echo First Index : ${arr_name[0]}  
4 echo Second Index : ${arr_name[1]}  
5 echo Third Index : ${arr_name[2]}
```

The 'Terminal' tab on the right shows the output of the script, which is:

```
First Index : SARAH  
Second Index : ZARA  
Third Index : AHMED
```

2. Write the output of the programs of Array.

```
Execute | Beautify | Share | Source Code | Help
```

```
1  set apples bananas oranges peach
2      echo $1
3      echo $3
4      echo $*
5      echo $#
6      echo $@
```

```
Terminal
apples
oranges
apples bananas oranges peach
4
apples bananas oranges peach
```

```
Execute | Beautify | Share | Source Code | Help
```

```
1  Names=""
2  Names="${Names} TheFirstNameHere"
3  Names="${Names} TheSecondNameHere"
4  Names="${Names} TheLastNameHere"
5  echo "Displaying one at a time ..."
6  for Names in ${Names};
7  do
8      echo ${Names};
9  done
10 echo "Only one element now"
11 echo ${Names}
```

```
Terminal
"Displaying one at a time ..."
main.sh: line 6: syntax error near unexpected token `$\n'
main.sh: line 6: `      for Names in ${Names};
.'
```

3. Write the output of the File Testing Program.

```
enter the first string
1
': not a valid identifier`
enter the second string
2
': not a valid identifier`
the concatenated string is" 12

...Program finished with exit code 0
Press ENTER to exit console.
```


4. Write a shell program to compare the two strings, whether the strings are equal or not.

```
GNU nano 5.3 demo.sh
string1="affan"
string2="affan"
if test "$string1" = "$string2"
then
    echo "string are equal"
else
    echo " string are not equal "
fi
```

Lab Journal 05

Date: 29-10-2023

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To write a program to create a process in LINUX.
- To understand exec process.
- To create child with sleep and wait command.
- To understand getpid() and getppid().

Lab Tasks:

1. Write the program for process creation using fork command.

Program

```
#include <iostream>
```

```
#include <unistd.h>
```

```
using namespace std;
```

```
int main()
```

```

{ pid_t fork(void);

pid_t c_pid = fork();

if (c_pid == -1) {

    perror("fork");

    exit(EXIT_FAILURE); }

else if (c_pid > 0) {

    cout << "printed from parent process " << getpid() << endl; }

else {

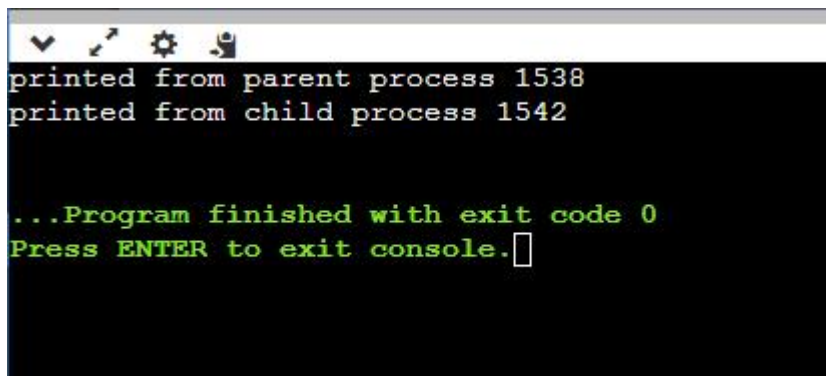
    cout << "printed from child process " << getpid() << endl; }

return 0; }

```

Program Execution

OUTPUT

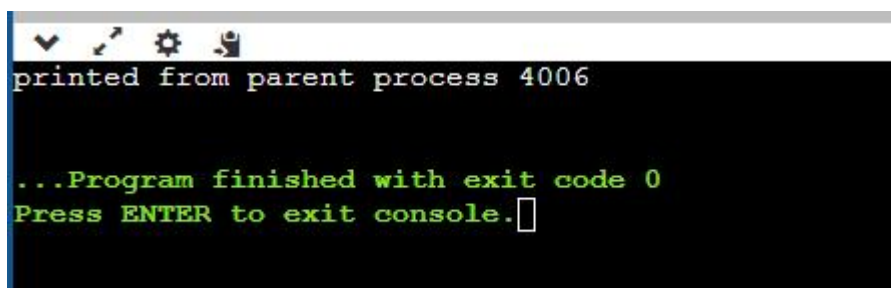


```

printed from parent process 1538
printed from child process 1542

...Program finished with exit code 0
Press ENTER to exit console.

```



```

printed from parent process 4006

...Program finished with exit code 0
Press ENTER to exit console.

```

Program

```
#include<iostream>
```

```

using namespace std;

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main(){

    pid_t cpid;

    if (fork()== 0)

        exit(0);

    else

        cpid = wait(NULL);

    cout <<"Parent pid = "<< getpid()<< endl;

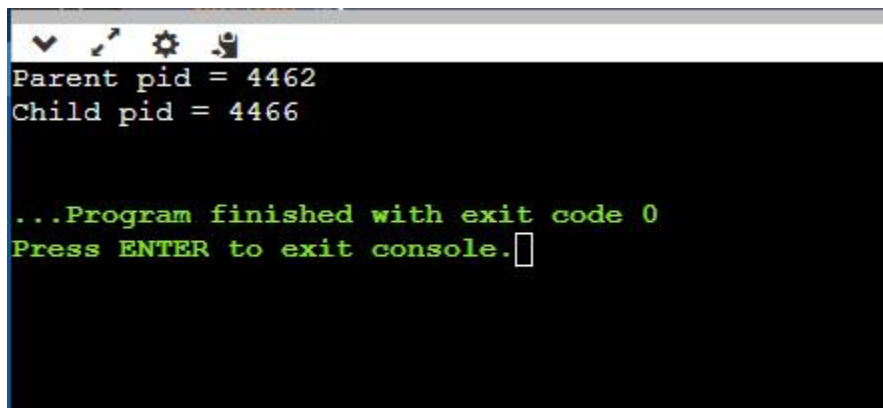
    cout << "Child pid = "<< cpid<< endl;

    return 0;}

```

Program Execution

OUTPUT



```

Parent pid = 4462
Child pid = 4466

...Program finished with exit code 0
Press ENTER to exit console.

```

2. Write a program illustrating the sleep command during process creation.

Program

```

echo "Program for executing LINUX command using Shell Programming"
echo "Welcome"

```

```
ps
exec ls
```

OUTPUT

```
[root@localhost ~]# bash ls
program for executing linux commands for shell programming
welcome
  PID TTY          TIME CMD
   47 hvc0      00:00:04 sh
  224 hvc0      00:00:00 bash
  225 hvc0      00:00:01 ps
bench.py fork hello.c ls sleep
[root@localhost ~]#
```

3. Write a program illustrating the wait command during process creation.

Program

```
#include <iostream>

using namespace std;

#include <unistd.h>

#include <sys/types.h>

int main() {

    pid_t child_pid;

    child_pid = fork();

    if (child_pid < 0) {

        cout << "Fork failed." << endl;

        return 1;

    } else if (child_pid == 0) {

        cout << "Child process: PID = " << getpid() << ", PPID = " << getppid() << endl;

        sleep(5)
```

```

        cout << "Child process after sleep: PID = " << getpid() << " PPID = " << getppid() << endl;
    } else {

        sleep(10);

        cout << "Parent process after sleep: PID = " << getpid() << std::endl;}

    return 0;}
Program Execution

```

OUTPUT

```

input
Child process: PID = 4318, PPID = 4314
Child process after sleep: PID = 4318 PPID = 4314
Parent process after sleep: PID = 4314

...Program finished with exit code 0
Press ENTER to exit console.

```

4. Write a program in C to create two processes Parent and Child through fork.

Program

```

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

    pid_t child_pid;

    child_pid = fork();

    if (child_pid < 0) {

        fprintf(stderr, "Fork failed.\n");

        return 1;
    }
}

```

```
} else if (child_pid == 0) {  
  
    printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());  
  
    int n;  
  
    printf("Enter a number: ");  
  
    scanf("%d", &n);  
  
    printf("Square of %d is: %d\n", n, n * n);  
  
} else {  
  
    printf("Parent process: PID = %d\n", getpid());  
  
    int num;  
  
    printf("Enter a number: ");  
  
    scanf("%d", &num);  
  
    printf("Table of %d:\n", num);  
  
    for (int i = 1; i <= 10; ++i) {  
  
        printf("%d x %d = %d\n", num, i, num * i);  
  
    }  
  
}return 0;}  
Program Execution
```

OUTPUT

```
input
Parent process: PID = 2493
Enter a number: Child process: PID = 2497, PPID = 2493
Enter a number: 10
Table of 10:
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Journal 06

Date: 03-28-2024

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To understand how the processes will cooperate by communicating with each other using the approach called as shared memory approach.

Lab Tasks:

1. Write the steps for sharing a common memory segment.

```
Parent Writing.....
Waiting for Child
Reading.....ZYXWVUTSRQPONMLKJIHGFEDCBA
I am Done

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Write the functions required for creating, attaching, detaching, and removing the memory segment.

```
Process 1: I have put the message '1'
Process 1: Contents of shared memory after writing '1': 1
Process 2: Contents of shared memory after writing 'hello': 2 hello
Process 3: Contents of shared memory after writing 'memory': 3 hello memory
```

3. Write a program in C to perform communication between parent and child through shared memory.
4. Write a program that creates a shared memory segment and waits until three other separate processes write something into that shared memory segment after which it prints what is written in shared memory.

Lab Journal 07

Date: 05/02/2024

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To write a C program to implement the CPU scheduling algorithm for First Come First Serve.
- To write a C program to implement the CPU scheduling algorithm for Shortest Job First.

Lab Tasks:

1. Calculate the Average Time using FCFS Algorithm.

Process	Duration	Order	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

Gantchart:

P1	P2	P3
-----------	-----------	-----------

P1 waiting time : 0

P2 waiting Time : 24

P3 waiting Time : 27

Average waiting Time is :

$$(0+24+27)/3=17$$

2. Write the program for First Come First Serve scheduling algorithm.

Program

```
#include<iostream>
```

```
using namespace std;
```

```
void findWaitingTime(int processes[], int n,
```

```
int bt[], int wt[])
```

```
{ wt[0] = 0;
```

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

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

```
void findTurnAroundTime( int processes[], int n,
```

```

        int bt[], int wt[], int tat[]){

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

    tat[i] = bt[i] + wt[i];}

void findavgTime( int processes[], int n, int bt[]){

    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, wt);

    findTurnAroundTime(processes, n, bt, wt, tat);

    cout << "Processes " << " Burst time "

        << " Waiting time " << " Turn around time\n";

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

        total_wt = total_wt + wt[i];

        total_tat = total_tat + tat[i];

        cout << " " << i+1 << "\t\t" << bt[i] << "\t "

            << wt[i] << "\t\t" << tat[i] << endl;}

    cout << "Average waiting time = "

        << (float)total_wt / (float)n;

    cout << "\nAverage turn around time = "

        << (float)total_tat / (float)n;}

int main(){

    int processes[] = { 1, 2, 3};

    int n = sizeof processes / sizeof processes[0];

    int burst_time[] = {10, 5, 8};

    findavgTime(processes, n, burst_time);

    return 0;}

```

OUTPUT

```
input
Processes  Burst time  Waiting time  Turn around time
  1         10         0         10
  2          5        10         15
  3          8        15         23
Average waiting time = 8.33333
Average turn around time = 16

...Program finished with exit code 0
Press ENTER to exit console.
```

3. Calculate the Average Time using SJF Algorithm.

Sorted Process Is :

Process	Duration	Order	Arrival Time
P4	3	4	0
P1	6	1	0
P3	7	3	0
P2	8	2	0

Process Order: P4 -> P1 -> P3 -> P2

Process	Duration	Order	Arrival Time	Completion Time	Waiting Time
P4	3	4	0	3	0
P1	6	1	0	9	3
P3	7	3	0	16	9
P2	8	2	0	24	16

Average Waiting Time = $(0 + 3 + 9 + 16) / 4 = 28 / 4 = 7$

4. Write the program for Shortest Job First scheduling algorithm.

Program

```
#include <bits/stdc++.h>

using namespace std;

struct Process {
    int pid;
```

```

int bt;

int art;

};

void findWaitingTime(Process proc[], int n,
                    int wt[]){

int rt[n];

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

    rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT_MAX;

int shortest = 0, finish_time;

bool check = false;

while (complete != n) {

    for (int j = 0; j < n; j++) {

        if ((proc[j].art <= t) &&

            (rt[j] < minm) && rt[j] > 0) {

            minm = rt[j];

            shortest = j;

            check = true;}}

    if (check == false) {

        t++;

        continue;}

    rt[shortest]--;

    minm = rt[shortest];

    if (minm == 0)

        minm = INT_MAX;

    if (rt[shortest] == 0) {

        complete++;

        check = false;

        finish_time = t + 1;

```

```

        wt[shortest] = finish_time -
            proc[shortest].bt -
            proc[shortest].art;
        if (wt[shortest] < 0)
            wt[shortest] = 0;}
    t++;}}

void findTurnAroundTime(Process proc[], int n,
    int wt[], int tat[]){
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];}

void findavgTime(Process proc[], int n){
    int wt[n], tat[n], total_wt = 0,
        total_tat = 0;
    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);
    cout << " P\t\t"
        << "BT\t\t"
        << "WT\t\t"
        << "TAT\t\t\n";
    for (int i = 0; i < n; i++) {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
            << proc[i].bt << "\t\t " << wt[i]
            << "\t\t " << tat[i] << endl;
    }
    cout << "\nAverage waiting time = "
        << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;}

```

```

int main(){
    Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },
                       { 3, 8, 1 }, { 4, 3, 0 }, { 5, 4, 4 } };

    int n = sizeof(proc) / sizeof(proc[0]);

    findavgTime(proc, n);

    return 0;
}

```

OUTPUT

```

input
P      BT      WT      TAT
1       6       7       13
2       2       0        2
3       8      14       22
4       3       0        3
5       4       2        6

Average waiting time = 4.6
Average turn around time = 9.2

...Program finished with exit code 0
Press ENTER to exit console.

```

Lab Journal 08

Date: 11 Nov 2023

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To write a C program to implement CPU scheduling algorithm for Priority Scheduling and Shortest Remaining Time First.

Lab Tasks:

- Write a C program to implement Priority Scheduling algorithm.

Code

```
#include <iostream>
```

```

#include <vector>

#include <algorithm>

using namespace std;

struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;};

bool compare_priority(const Process &a, const Process &b) {
    return a.priority < b.priority;}

void calculate_waiting_time(vector<Process> &processes) {
    int total_waiting_time = 0;

    processes[0].waiting_time = 0; // First process has 0 waiting time
    for (size_t i = 1; i < processes.size(); i++) {
        processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
        total_waiting_time += processes[i].waiting_time;
    }
}

void calculate_turnaround_time(vector<Process> &processes) {
    int total_turnaround_time = 0;

    for (size_t i = 0; i < processes.size(); i++) {
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        total_turnaround_time += processes[i].turnaround_time;
    }
}

void display_results(const vector<Process> &processes) {
    cout << "Process ID\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n";

    for (size_t i = 0; i < processes.size(); i++) {
        cout << processes[i].process_id << "\t\t" << processes[i].arrival_time << "\t\t"

```

```

        << processes[i].burst_time << "\t\t" << processes[i].priority << "\t\t"
        << processes[i].waiting_time << "\t\t" << processes[i].turnaround_time << endl;
    }
}

void priority_scheduling(vector<Process> &processes) {
    sort(processes.begin(), processes.end(), compare_priority);
    calculate_waiting_time(processes);
    calculate_turnaround_time(processes);
    display_results(processes);}

int main() {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;
    vector<Process> processes(n);
    for (int i = 0; i < n; i++) {
        cout << "Enter details for process " << i + 1 << ":\n";
        cout << "Process ID: ";
        cin >> processes[i].process_id;
        cout << "Arrival Time: ";
        cin >> processes[i].arrival_time;
        cout << "Burst Time: ";
        cin >> processes[i].burst_time;
        cout << "Priority: ";
        cin >> processes[i].priority;}
    priority_scheduling(processes);
    return 0;}

```

OUTPUT


```
input
Enter the number of processes: 2
Enter details for process 1:
Process ID: 1
Arrival Time: 2
Burst Time: 5
Priority: 3
Enter details for process 2:
Process ID: 2
Arrival Time: 4
Burst Time: 9
Priority: 2
Process ID      Arrival Time      Burst Time      Priority      Waiting Time      Turnaround Time
2                4                9                2                0                9
1                2                5                3                9                14

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Write the output of a C program for Shortest Remaining Time First.

```
Enter the Total Number of Processes: 3
Enter Details of 3 Processes
Enter Arrival Time: 3
Enter Burst Time: 4
Enter Arrival Time: 2
Enter Burst Time: 5
Enter Arrival Time: 6
Enter Burst Time: 7

Average Waiting Time: 3
Average Turnaround Time: 8.33333

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Journal 09

Date: 5/16/24

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

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

Lab Tasks:

1. Calculate the Average Time using Round Robin. Draw the GANTT Chart.

Process No	Arrival Time (AT)	Burst Time (BT)	Complete Time (CT)	Turnaround Time (CT-AT)	Waiting Time (TAT-BT)	Response Time
P1	0	10	22	22	12	0
P2	1	4	12	11	7	2

P3	2	5	18	16	11	4
P4	3	3	17	14	11	6

Ready Queue:

Gantt Chart:

P1	P2	P3	P4	P1	P2	P3	P4	P1	P3	P1	
----	----	----	----	----	----	----	----	----	----	----	--

0 2 4 6 8 10 12 14 15 17 18 22

2. Write the output for Round Robin Scheduling Algorithm.

```

input
Enter the no. of processes: 4
Enter the quantum:
2
Enter the process numbers:
1
2
3
4
Enter the Arrival time of processes:
0
1
2
3
Enter the Burst time of processes:
10
4
5
3
Process number Arrival time Burst time Start time Final time Wait Time TurnAround Time
1 0 10 0 6 14 18 20 22 12 22
2 1 4 2 10 12 7 11
3 2 5 4 12 17 11 16
4 3 3 8 16 17 11 14
The average wait time is: 10.25
The average TurnAround time is: 15.75

...Program finished with exit code 0
Press ENTER to exit console.

```

Lab Journal 10

Date: 5/23/2024

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To study about Signal Handling. Use top, ps, and Kill commands.

Lab Tasks:

1. Write the output for top and ps. Differentiate between the both terms.

```
top - 18:26:40 up 20 days, 3:48, 1 user, load average: 1.06, 1.18, 1.56
Tasks: 337 total, 2 running, 335 sleeping, 0 stopped, 0 zombie
%Cpu(s): 9.3 us, 3.0 sy, 0.0 ni, 85.7 id, 2.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7699.5 total, 137.4 free, 6411.4 used, 1150.7 buff/cache
MiB Swap: 5897.7 total, 225.7 free, 5672.0 used. 443.6 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3325	adminis+	20	0	5903612	167292	24648	R	16.7	2.1	340:22.43	gnome-shell
3193	adminis+	20	0	839340	28200	5180	S	8.7	0.4	259:31.95	Xorg
3028484	adminis+	20	0	836836	21760	10040	S	4.0	0.3	1:57.12	gnome-terminal-
2932480	adminis+	20	0	33.5g	476384	140408	S	3.3	6.0	379:23.73	chrome
3180832	adminis+	20	0	1132.0g	494504	90716	S	3.3	6.3	15:30.80	chrome
4188	adminis+	20	0	1123.0g	42940	27640	S	3.0	0.5	18:50.60	cliq
4380	adminis+	20	0	1134.9g	124584	38336	S	2.7	1.6	465:22.38	cliq
2932524	adminis+	20	0	33.0g	181400	81228	S	2.3	2.3	443:20.27	chrome
4315	adminis+	20	0	32.5g	31324	13992	S	1.7	0.4	363:25.04	cliq
197	root	-51	0	0	0	0	S	1.0	0.0	13:25.20	irq/51-DELL09ED

2. Write the output for the use of aux with ps for the firefox program.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	892	572	?	S1	Nov28	0:00	/init
root	227	0.0	0.0	900	80	?	Ss	Nov28	0:00	/init
root	228	0.0	0.0	900	88	?	S	Nov28	0:00	/init
zaphod	229	0.0	0.1	749596	31000	pts/0	Ssl+	Nov28	0:15	docker
root	240	0.0	0.0	0	0	?	Z	Nov28	0:00	[init] <defunct>
root	247	0.0	0.0	900	88	?	S	Nov28	0:00	/init
root	248	0.0	0.1	1758276	31408	pts/1	Ssl+	Nov28	0:10	/mnt/wsl/docker-deskt
root	283	0.0	0.0	892	80	?	Ss	Dec01	0:00	/init
root	284	0.0	0.0	892	80	?	R	Dec01	0:00	/init
zaphod	285	0.0	0.0	11964	5764	pts/2	Ss	Dec01	0:00	-zsh
zaphod	343	0.0	0.0	23764	9836	pts/2	T	17:44	0:00	vi foo
root	349	0.0	0.0	892	80	?	Ss	17:45	0:00	/init
root	350	0.0	0.0	892	80	?	S	17:45	0:00	/init
zaphod	351	0.0	0.0	11964	5764	pts/3	Ss+	17:45	0:00	-zsh
zaphod	601	0.0	0.0	10612	3236	pts/2	R+	18:24	0:00	ps aux

3. Write the output of program for Kill Signal for firefox. 4.1 Write a C program with an infinite loop and a custom signal handler to handle the interrupt signal (Ctrl+C).

In C++:

```
#include <iostream>

using namespace std;

#include <csignal>

#include <unistd.h>
```

```

void handle_sigint(int sig){
    cout << "Caught signal " << sig << std::endl;}

int main() {
    signal(SIGINT, handle_sigint);
    while (true) {
        cout << "hello world" << std::endl;
        sleep(1);}
    return 0;}

```

Output

```

hello world
hello world
hello world
hello world
hello world

```

```

^CCaught signal 2

```

4.2 Write a program in C with an infinite loop and a custom signal handler to handle at least kill -15 (SIGTERM) and kill -9 (SIGKILL). Send both these signals (kill -9 and -15) using your running process's PID.

Solution:

```

#include <iostream>

#include <csignal>

#include <cstdlib>

#include <unistd.h>

void handle_sigterm(int sig);

int main() {

    std::signal(SIGTERM, handle_sigterm);

    while (true) {

```

```
std::cout << "Running... (PID: " << getpid() << ")" << std::endl;

sleep(1);}

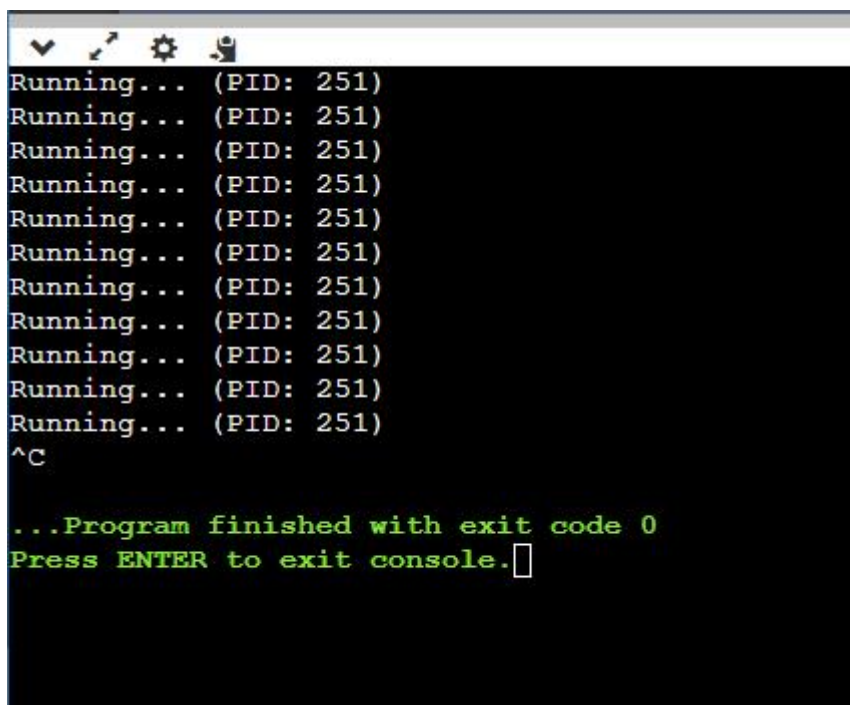
return 0;}

void handle_sigterm(int sig) {

std::cout << "Caught signal " << sig << " (SIGTERM), exiting gracefully..." << std::endl;

std::exit(0);}
```

Output:

A screenshot of a terminal window with a black background and white text. The window has a title bar with standard Linux window controls (minimize, maximize, close) and a toolbar with icons for search, settings, and a file manager. The output of the program is displayed as follows:

```
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
Running... (PID: 251)
^C

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Journal 11

Date: 05/30/24

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- Understanding of threads, creation of threads, passing arguments to threads and to joining threads.

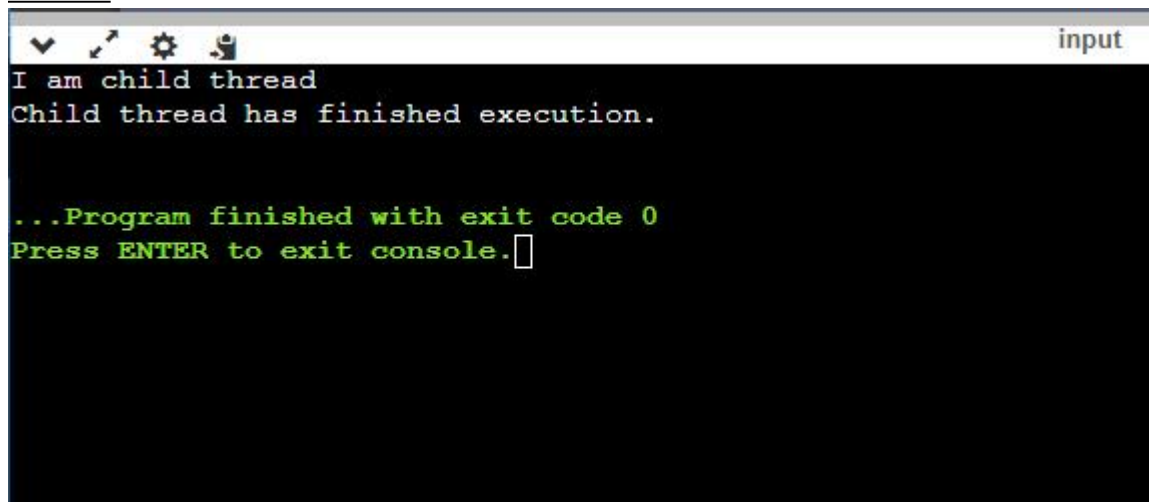
Lab Tasks:

1. Create a simple program for thread creation and termination. The created thread should display "I am child thread".

Program

```
#include <iostream>  
using namespace std;  
#include <thread>  
void childThreadFunction() {  
    cout << "I am child thread" << endl;}  
int main() {  
    thread childThread(childThreadFunction);  
    childThread.join();  
    cout << "Child thread has finished execution." << endl;  
    return 0;}
```

OUTPUT



```
input  
I am child thread  
Child thread has finished execution.  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

2. Write a program that creates a number of threads and each thread should print "Hello World!" along with its number passed as argument in a thread.

Program

```
#include <iostream>  
using namespace std;  
#include <thread>  
#include <vector>  
void printHelloWorld(int threadNumber) {
```

```

    cout << "Hello World! from thread " << threadNumber << endl;}

int main() {

    const int numThreads = 10;

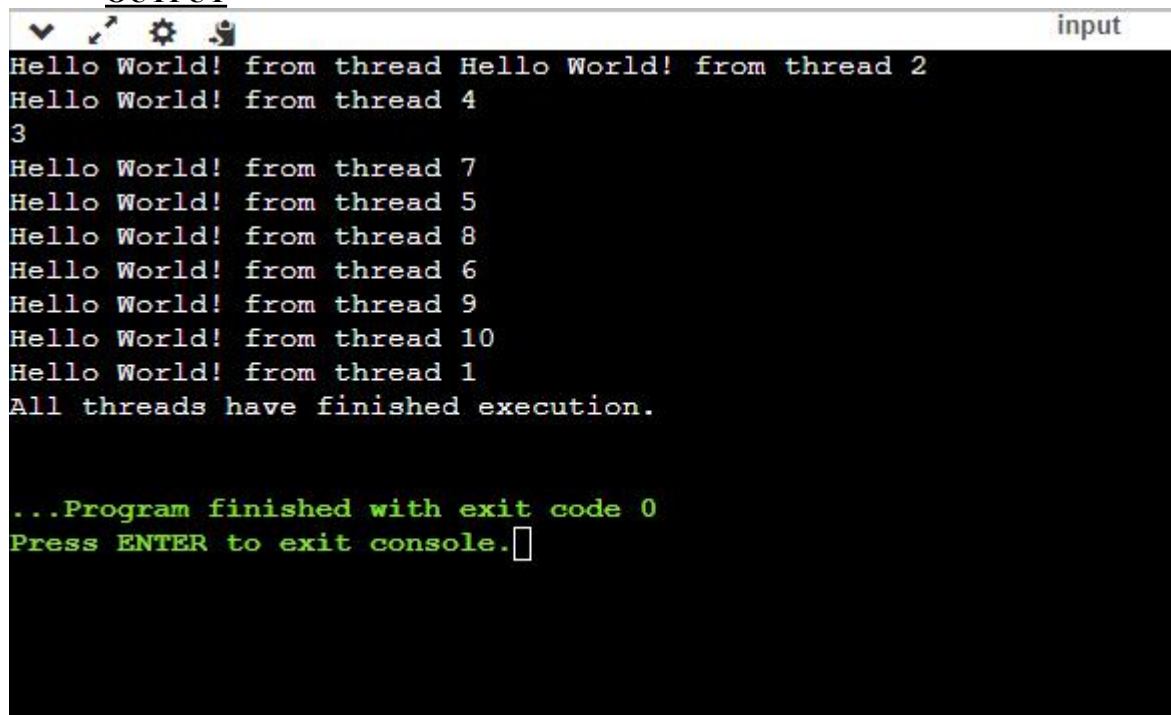
    vector<thread> threads;
    for (int i = 0; i < numThreads; ++i) {
        threads.push_back(thread(printHelloWorld, i + 1));}
    for (auto& thread : threads) {
        thread.join();}

    cout << "All threads have finished execution." << endl;

    return 0;}

```

OUTPUT



```

input
Hello World! from thread Hello World! from thread 2
Hello World! from thread 4
3
Hello World! from thread 7
Hello World! from thread 5
Hello World! from thread 8
Hello World! from thread 6
Hello World! from thread 9
Hello World! from thread 10
Hello World! from thread 1
All threads have finished execution.

...Program finished with exit code 0
Press ENTER to exit console.

```

3. Write the output of the program for the difference between processes and threads. Check the values of Global and Local variables in threads and processes.


```
input
Create two threads to see what content they share

Thread :
  Global Variable :10
  Local Variable:200
Thread after incrementing global variable ::
  Global Variable :11
  Local Variable:200
Thread :
  Global Variable :11
  Local Variable:200
Thread after incrementing global variable ::
  Global Variable :12
  Local Variable:200
After thread global var = 12

Before Fork :
Global Variable:12
Local Variable:20
In child:
Global Variable=12
Local Variable=20
Child set: Global variable=100
Local Variable=133Local Variable:20
In parent:
Global Variable:12
Local Variable=20

...Program finished with exit code 0
Press ENTER to exit console.
```

4.1 Write a program to create two threads. One should take input from user and stores the factorial of that input. Other should take two variable base and power, calculate power.

```
#include <iostream>
using namespace std;
#include <thread>
#include <cmath>
void calculateFactorial() {
    int number;
    cout << "Enter a number to calculate its factorial: ";
```

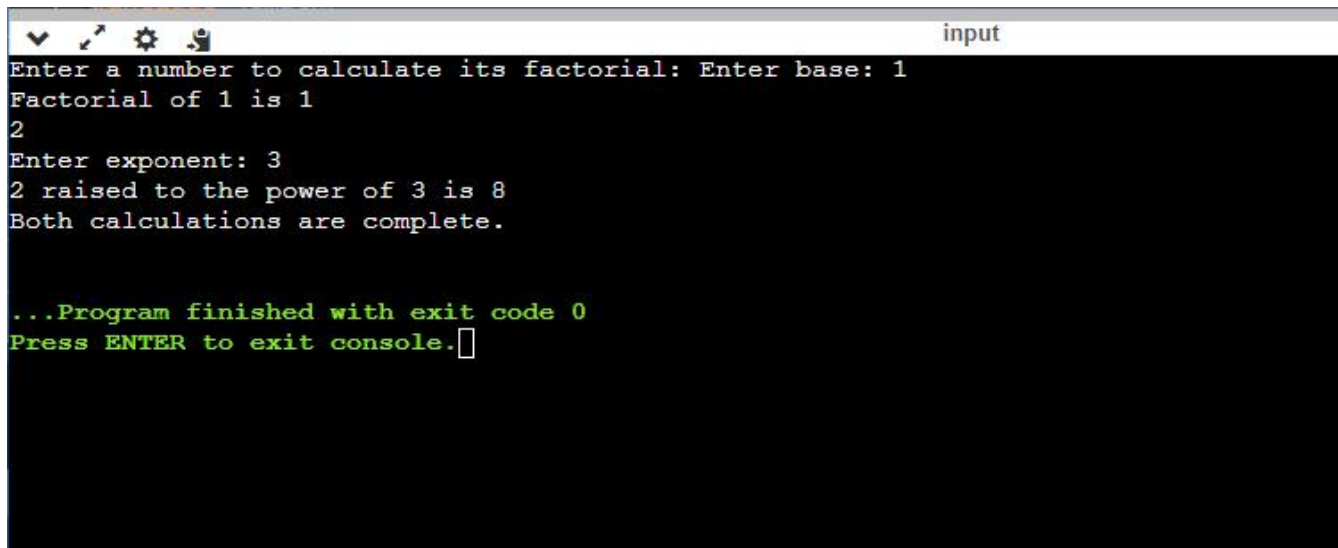
```

    cin >> number;
    long long factorial = 1;
    for (int i = 1; i <= number; ++i) {
        factorial *= i;}
    cout << "Factorial of " << number << " is " << factorial << endl;}

void calculatePower() {
    double base;
    int exponent;
    cout << "Enter base: ";
    cin >> base;
    cout << "Enter exponent: ";
    cin >> exponent;
    double result = pow(base, exponent);
    cout << base << " raised to the power of " << exponent << " is " << result <<
endl;}

int main() {
    thread factorialThread(calculateFactorial);
    thread powerThread(calculatePower);
    factorialThread.join();
    powerThread.join();
    cout << "Both calculations are complete." << endl;
    return 0;}

```

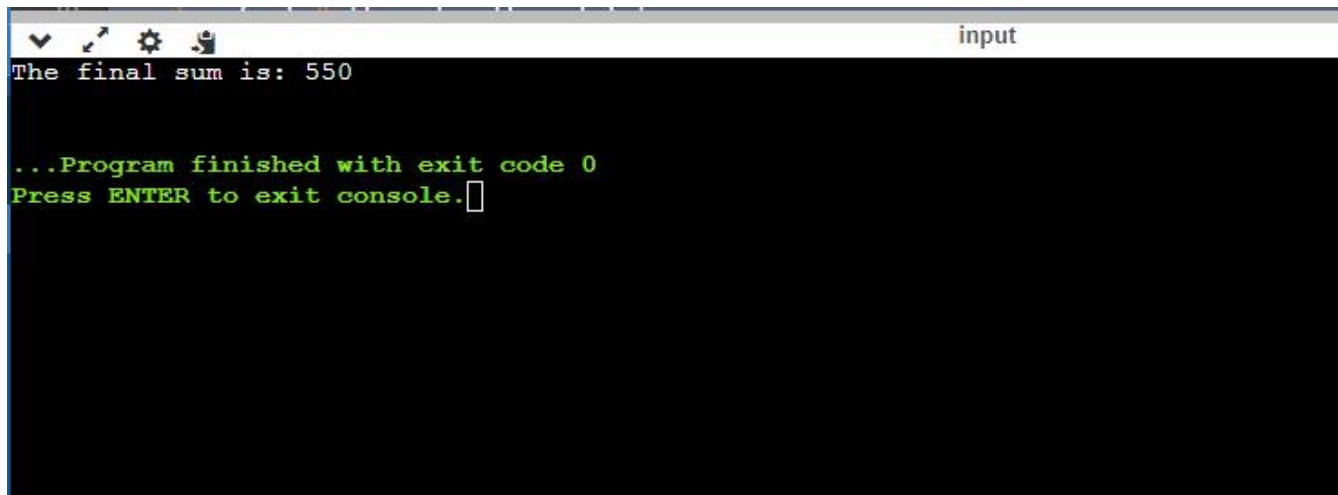


```
input
Enter a number to calculate its factorial: Enter base: 1
Factorial of 1 is 1
2
Enter exponent: 3
2 raised to the power of 3 is 8
Both calculations are complete.

...Program finished with exit code 0
Press ENTER to exit console.□
```

4.2 Create ten threads. A global variable is declared sum. As each thread is created a function is called to calculate the sum of number from 1-10.

```
#include <iostream>
using namespace std;
#include <thread>
#include <vector>
#include <mutex>
int sum = 0;
mutex sumMutex;
void calculateAndAddSum() {
    int localSum = 0;
    for (int i = 1; i <= 10; ++i) {
        localSum += i;}
    lock_guard<mutex> guard(sumMutex);
    sum += localSum;}
int main() {
    const int numThreads = 10;
    vector<thread> threads;
    for (int i = 0; i < numThreads; ++i) {
        threads.push_back(thread(calculateAndAddSum));}
    for (auto& thread : threads) {
        thread.join();}
    cout << "The final sum is: " << sum << endl;
    return 0;}
```



```
input
The final sum is: 550

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Journal 12

Date: 6/06/24

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To understand the use of semaphores and how they are used in process synchronization.

Lab Tasks:

1. Write a program to create two processes that share a common variable using a semaphore to synchronize access.
2. Write a program to implement the Producer-Consumer problem using semaphores.
3. Write a program to implement the Reader-Writer problem using semaphores.

Lab Journal 13

Date: 6/13/24

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To understand the implementation of Deadlock avoidance using Banker's Algorithm.

Lab Tasks:

1. Write a program to simulate Banker's Algorithm for Deadlock avoidance.
2. Write a program to detect Deadlock in a given set of processes.

Lab Journal 14

Date: 6/20/24

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To understand paging and segmentation in memory management.

Lab Tasks:

1. Write a program to simulate Paging in memory management.
 2. Write a program to simulate Segmentation in memory management.
-

Lab Journal 15

Date: 6/27/24

Max Marks: 20

Faculty's Name: Abdullah

Objective(s):

- To understand file systems and how they are managed in an operating system.

Lab Tasks:

1. Write a program to simulate the creation and deletion of files.
2. Write a program to simulate the management of file permissions.