# Lab Exercise - 3.1

**Aim : 3-A : Write a program using various process management system calls**

**a) Process Creation**<br>
**b) Executing a command**<br>
**c) Sleep command**<br>
**d) Signal handling using kill**<br>
**e) Wait command**

**Theory : Students are required to study and write description of following points**

**System Call**:

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

**Types of System Call**:

**Process Control**

These system calls deal with processes such as process creation, process termination etc.

**File Management**

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

**Device Management**

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

**Information Maintenance**

These system calls handle information and its transfer between the operating system and the user program.

**Communication**

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

## System Call Interface:

The system call interface is the programming interface for application programmers.
The programmer must live with the interface that T&R have defined. The interface provides the process, interprocess communication, file, tty, and user abstractions.
Most programming languages provides a system call interface
- It serves as the link to system calls made available by the operating system
- It intercepts function calls in the API and invokes the necessary system call within the operating system
- Most of the details of the operating system interfaces are hidden from the programmer by the API

**Steps to perform:**

1. **Read the following algorithms and programs carefully, understand it and type only programs in any word processor like wordpad, notepad etc.**
2. **Save as it with .c extension**
3. **Compile and run it using commands given below. Verify Actual output with sample output**
4. **Take a snapshot of the actual Output and paste in the box provided for output.**

### a) Process Creation:

**ALGORITHM:**

```
STEP 1: Start the program.
STEP 2: Declare pid as integer.
STEP 3: Create the process using Fork command.
STEP 4: Check pid is less than 0 then print error else if pid is equal to 0 then print “child
        process created” else print ”parent process created”.
STEP 5: Stop the program.
```

### PROGRAM:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
pid_t id;
id=fork();
if(id<0)
{
printf("cannot create the file");
_exit(-1);
}
if(id==0)
{
printf("\nchild process created");
_exit(0);
}
else
{
printf("\nparent process created");
}

return 0;
}
```

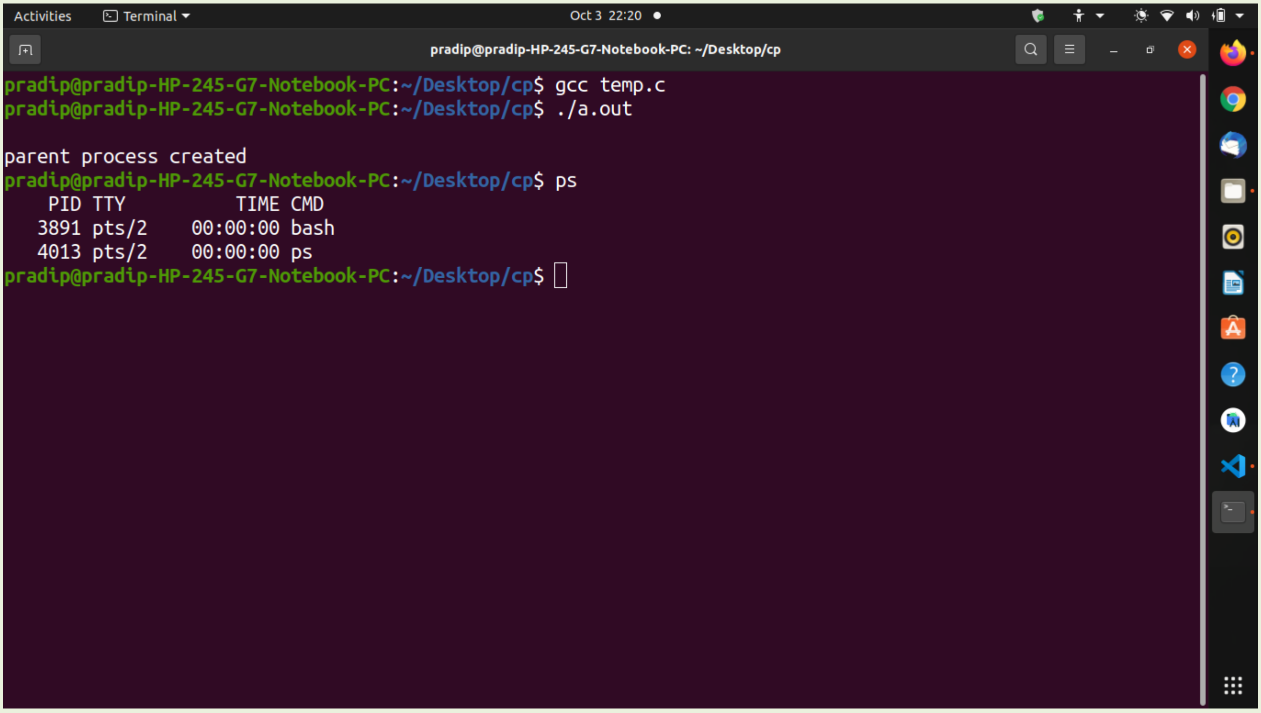
**Compile program with following command (Change name of file with your filename)**
***$gcc pc.c -o pc***

**Run the program with following command (Change name of file with your filename):**
***$./pc***

**SAMPLE OUTPUT:**

```
parent process created
$ child process  created
$ps
PID CLS PRI TTY TIME COMD
5913 TS 70 pts022 0:00 ksh
6229 TS 59 pts022 0:00 ps
```

**ACTUAL OUTPUT (Paste snapshot of actual output here in the box given below):**



```
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ gcc temp.c
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ ./a.out

parent process created
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ ps
    PID TTY          TIME CMD
   3891 pts/2    00:00:00 bash
   4013 pts/2    00:00:00 ps
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ 
```

**b) Executing a script:**

## PROGRAM (Save it as exec.sh extension):

```
echo Program for executing UNIX command using shell programming
echo Welcome
ps
```

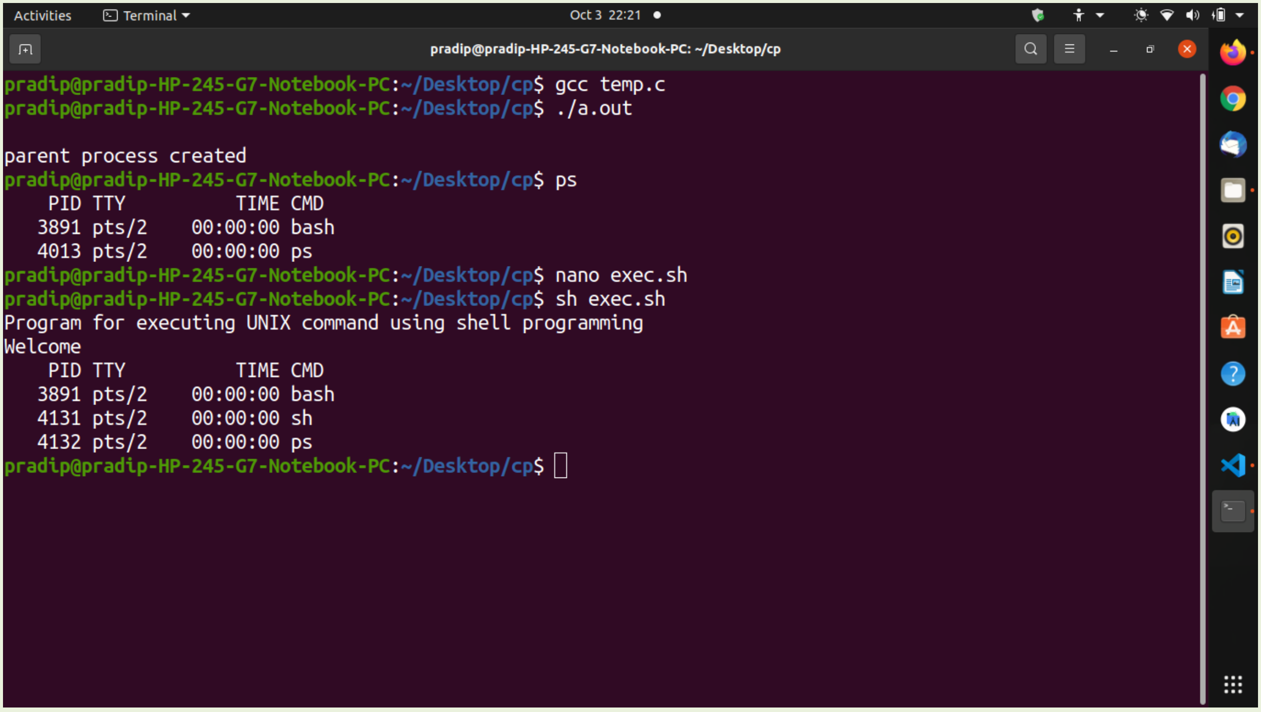
**Run script with following command (Change name of file with your filename)**

***$ sh exec.sh***

## SAMPLE OUTPUT:

```
program for executing UNIX command using shell programming
Welcome
PID CLS PRI TTY
TIME COMD
958 TS 70 pts001 0:00 ksh
971 TS 70 pts001 0:00 sh
972 TS 59 pts001 0:00 ps
```

## ACTUAL OUTPUT (Paste snapshot of actual output here in the box given below):



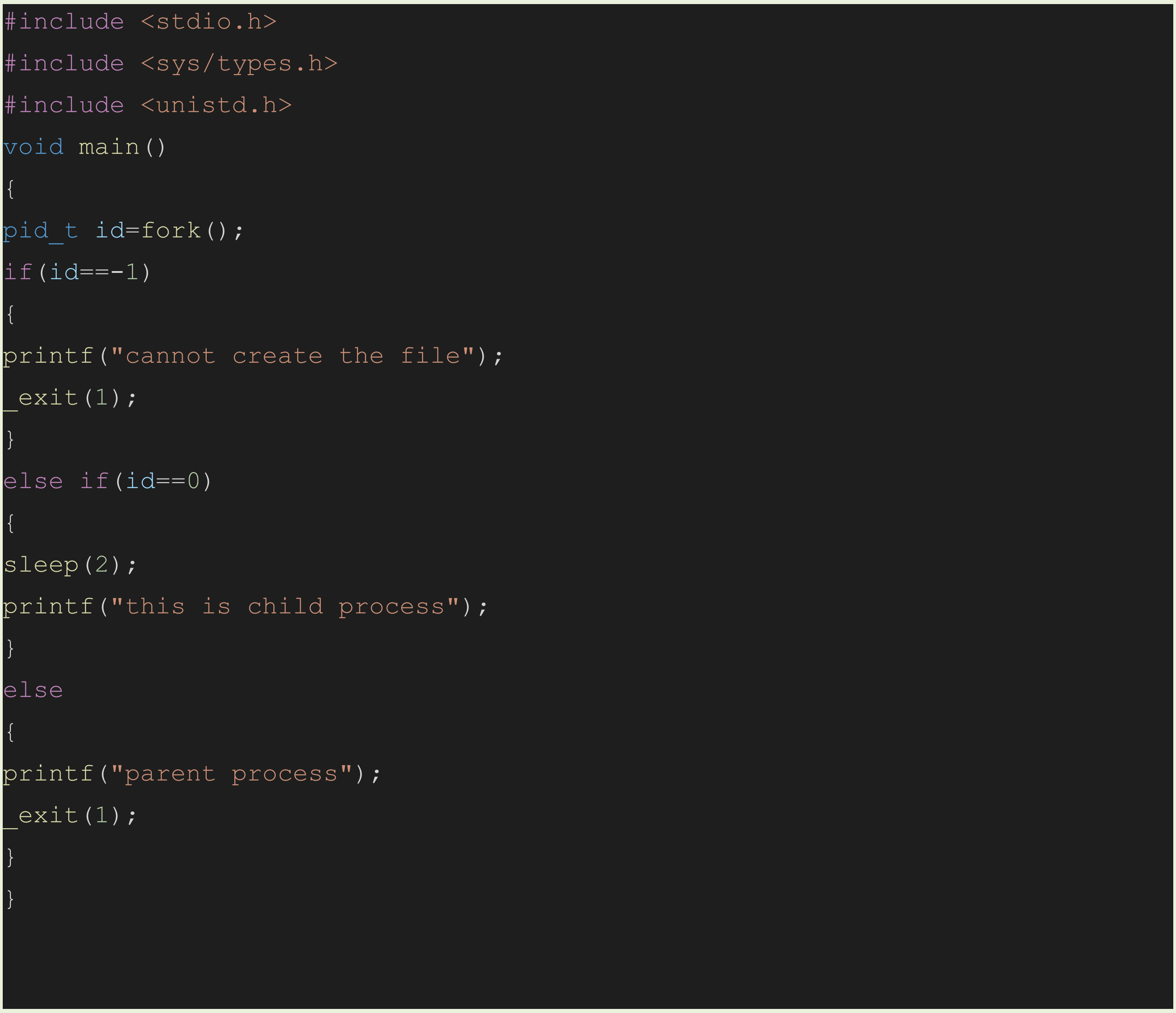
```
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ gcc temp.c
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ ./a.out

parent process created
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ ps
    PID TTY          TIME CMD
   3891 pts/2    00:00:00 bash
   4013 pts/2    00:00:00 ps
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ nano exec.sh
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ sh exec.sh
Program for executing UNIX command using shell programming
Welcome
    PID TTY          TIME CMD
   3891 pts/2    00:00:00 bash
   4131 pts/2    00:00:00 sh
   4132 pts/2    00:00:00 ps
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ 
```

## c) Sleep command:

## ALGORITHM:

```
    STEP 1: Start the program.
    STEP 2: Create process using fork and assign into a variable.
    STEP 3: If the value of variable is < zero print not create and > 0 process create and
    else print child create.
    STEP 4: Create child with sleep of 2.
    STEP 5: Stop the program.
```

## PROGRAM:



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
pid_t id=fork();
if(id==-1)
{
printf("cannot create the file");
_exit(1);
}
else if(id==0)
{
sleep(2);
printf("this is child process");
}
else
{
printf("parent process");
_exit(1);
}
}
```

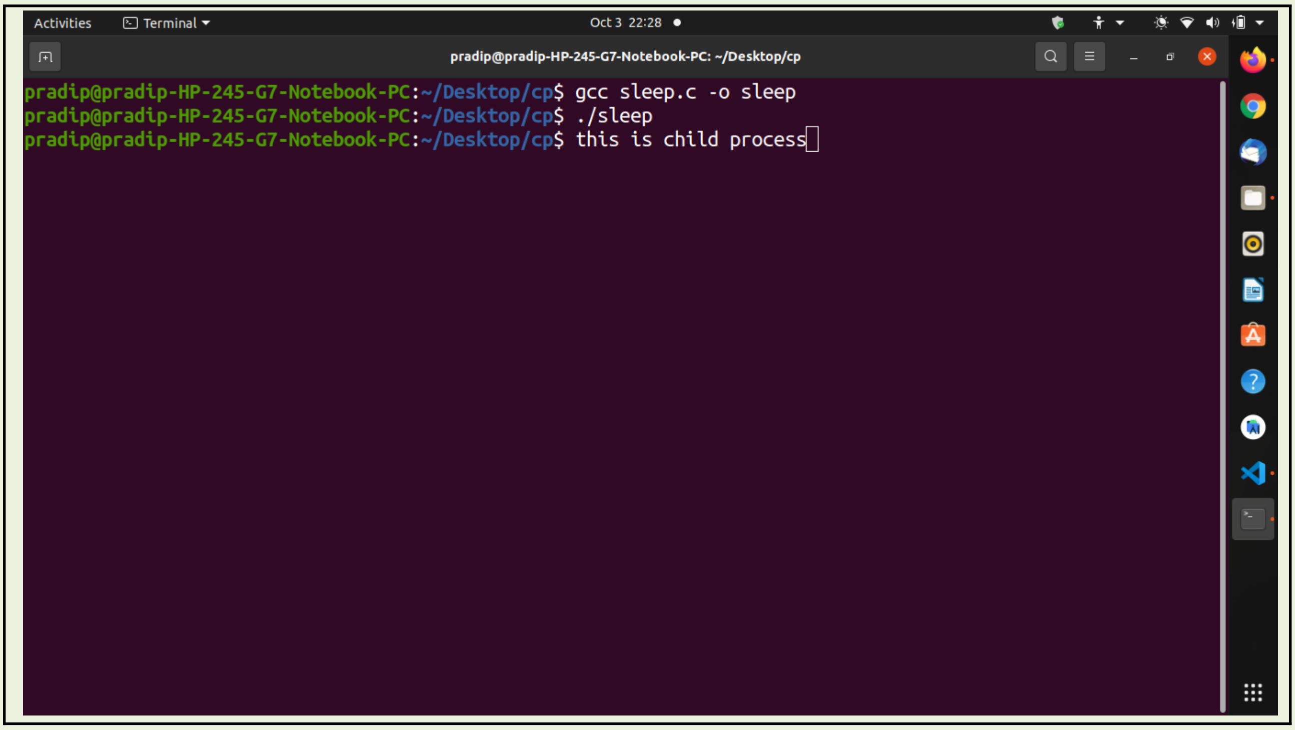
**Compile program with following command (Change name of file with your filename)**
***$ gcc sleep.c -o sleep***

**Run the program with following command (Change name of file with your filename):**
***$ . /sleep***

## SAMPLE OUTPUT:

```
parent process
$ this is child process
```

**ACTUAL OUTPUT (Paste snapshot of actual output here in the box given below):**



```
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ gcc sleep.c -o sleep
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ ./sleep
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ this is child process
```

## d) Signal handling using kill:

### ALGORITHM:

STEP 1:start the program<br>
STEP 2:Read the value of pid.<br>
STEP 3:Kill the command surely using kill-9 pid.<br>
STEP 4:Stop the program.

**PROGRAM**:

```
    echo “program for performing KILL operations”
    ps
    echo enter the pid
    read pid
    kill $pid
    echo finished
```

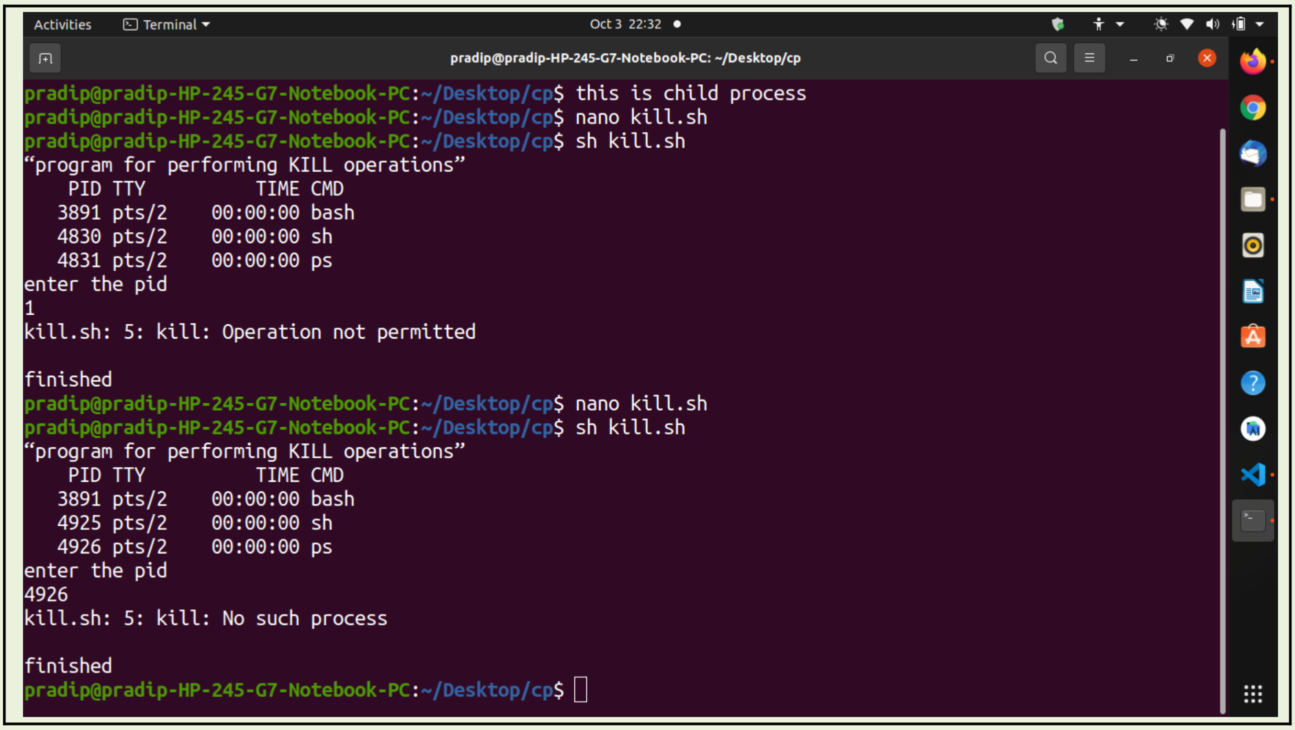
**Run script with following command (Change name of file with your filename)**

***$sh kill.sh***

### SAMPLE OUTPUT:

```
program for performing KILL operations
PID CLS PRI TTY
TIME COMD
858 TS 70 pts001 0:00 ksh
858 TS 70 pts001 0:00 sh
858 TS 59 pts001 0:00 ps
enter the pid:  858
finished
```

## ACTUAL OUTPUT (Paste snapshot of actual output here in the box given below):



```
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ this is child process
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ nano kill.sh
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ sh kill.sh
“program for performing KILL operations”
    PID TTY          TIME CMD
   3891 pts/2    00:00:00 bash
   4830 pts/2    00:00:00 sh
   4831 pts/2    00:00:00 ps
enter the pid
1
kill.sh: 5: kill: Operation not permitted

finished
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ nano kill.sh
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ sh kill.sh
“program for performing KILL operations”
    PID TTY          TIME CMD
   3891 pts/2    00:00:00 bash
   4925 pts/2    00:00:00 sh
   4926 pts/2    00:00:00 ps
enter the pid
4926
kill.sh: 5: kill: No such process

finished
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ 
```

### e) Wait command:

### ALGORITHM:

```
STEP 1:Start the execution
STEP 2:Create process using fork and assign it to a variable
STEP 3:Check for the condition pid is equal to 0
STEP 4:If it is true print the value of i and terminate the child process
STEP 5:If it is not a parent process has to wait until the child terminate
STEP 6:Stop the execution
```

### PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int i=10;
int main(){
   pid_t pid = fork();
   if(pid==0){
       printf("\nInitial value of i : %d",i);
       i+=10;
       printf("\nValue of i changed to : %d",i);
       printf("\nChild process terminated");
   }
```

```
    else{
        wait(0);
        printf("\nvalue of i in parent process : %d",i);
    }
    return 0;
}
```

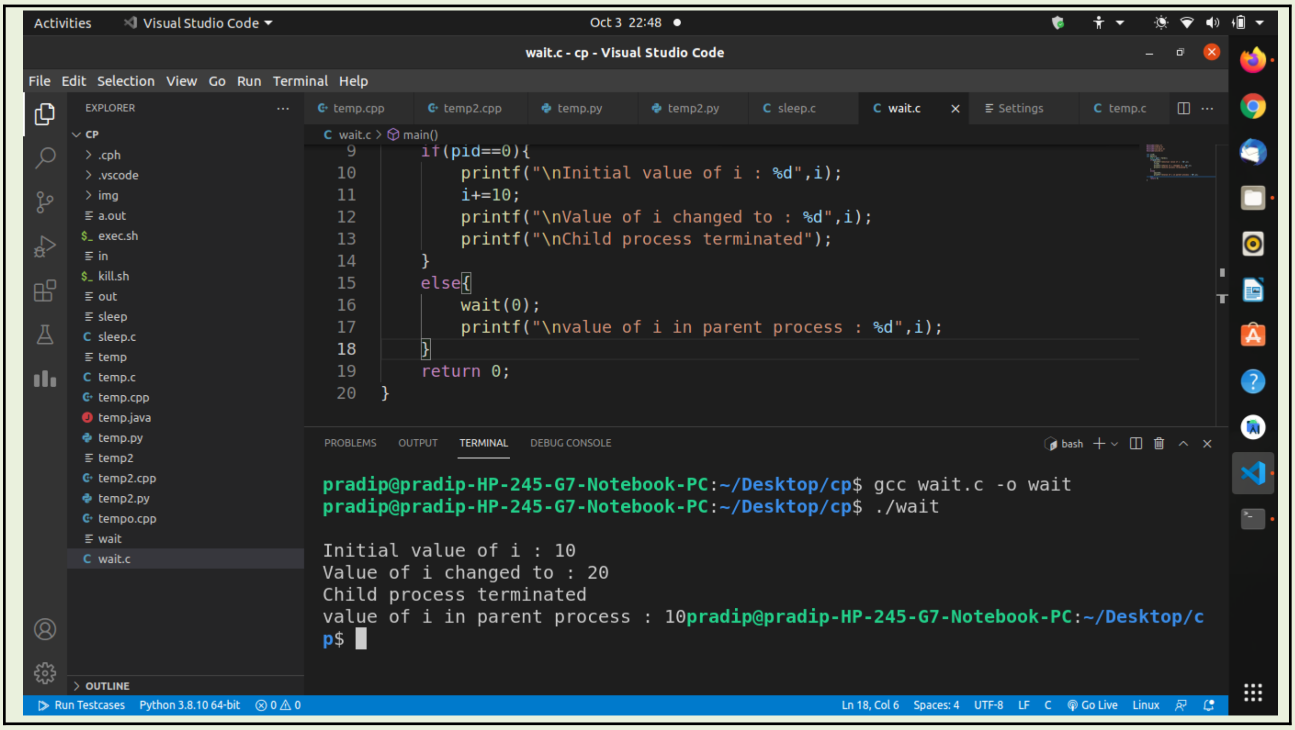
**Compile program with following command (Change name of file with your filename)**
***$gcc wait.c -o wait***

**Run the program with following command (Change name of file with your filename):**
***$ . / wait***

**SAMPLE OUTPUT:**

initial value of i 10
value of i 20
child teriminated
value of i in parent process 10$

**ACTUAL OUTPUT (Paste snapshot of actual output here in the box given below):**



```
    if(pid==0){
        printf("\nInitial value of i : %d",i);
        i+=10;
        printf("\nValue of i changed to : %d",i);
        printf("\nChild process terminated");
    }
    else{
        wait(0);
        printf("\nvalue of i in parent process : %d",i);
    }
    return 0;
}
```

```
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ gcc wait.c -o wait
pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/cp$ ./wait

Initial value of i : 10
Value of i changed to : 20
Child process terminated
value of i in parent process : 10pradip@pradip-HP-245-G7-Notebook-PC:~/Desktop/c
p$
```

**Result (Write result of  your experiment in box given below):**

In this practical, I leant about different system calls.