Ayesha Zamurd

49733

BSCS-5

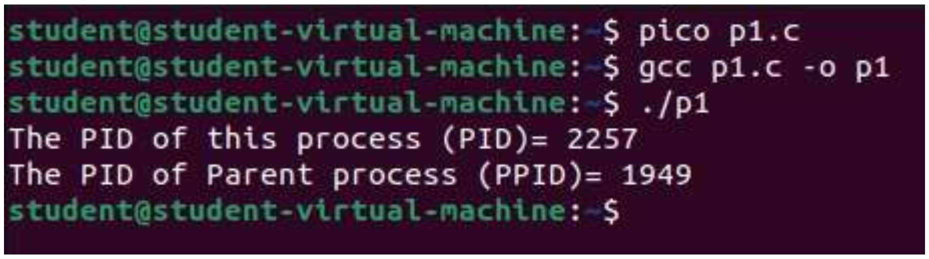
Lab #08

LAB MANUAL CODES:

Lab Code: 1

```
#include<stdio.h>
#include<unistd.h>
int main(void)
{
 printf("The PID of this process (PID)= %d\n", getpid());
 printf("The PID of Parent process (PPID)= %d\n", getppid());
 return 0;
}
```

Output:

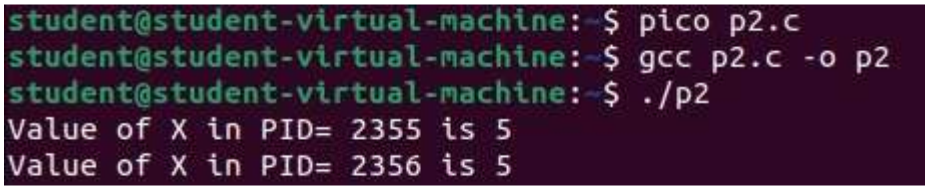


```
student@student-virtual-machine:~$ pico p1.c
student@student-virtual-machine:~$ gcc p1.c -o p1
student@student-virtual-machine:~$ ./p1
The PID of this process (PID)= 2257
The PID of Parent process (PPID)= 1949
student@student-virtual-machine:~$
```

Lab Code: 2

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    fork();
    int x=5;
    pid_t pid = getpid();
    printf("Value of X in PID= %d is %d\n",pid,x);
    return 0;
}
```

Output :

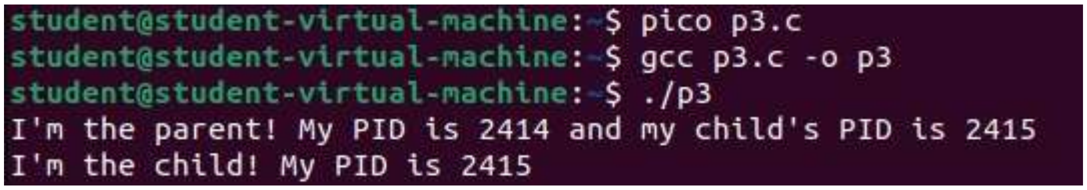


```
student@student-virtual-machine:~$ pico p2.c
student@student-virtual-machine:~$ gcc p2.c -o p2
student@student-virtual-machine:~$ ./p2
Value of X in PID= 2355 is 5
Value of X in PID= 2356 is 5
```

Lab Code: 3

```
#include <stdio.h>

#include <unistd.h>

int main() {

    int pid = fork();

    if (pid == 0) {

        printf("I'm the child! My PID is %d\n", getpid());

    } else {

        printf("I'm the parent! My PID is %d and my child's PID is %d\n", getpid(), pid);

    }

    return 0;

}
```

Output:

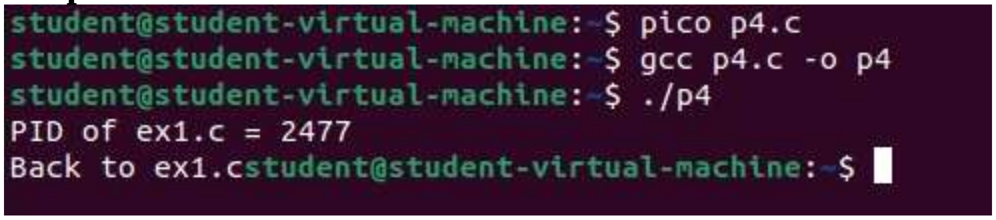


```
student@student-virtual-machine:~$ pico p3.c
student@student-virtual-machine:~$ gcc p3.c -o p3
student@student-virtual-machine:~$ ./p3
I'm the parent! My PID is 2414 and my child's PID is 2415
I'm the child! My PID is 2415
```

Lab Code: 4

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("PID of ex1.c = %d\n", getpid());

    char *args[] = {"Hello", NULL};
    execv("./ex2", args);

    printf("Back to ex1.c");
    return 0;
}
```

Output:

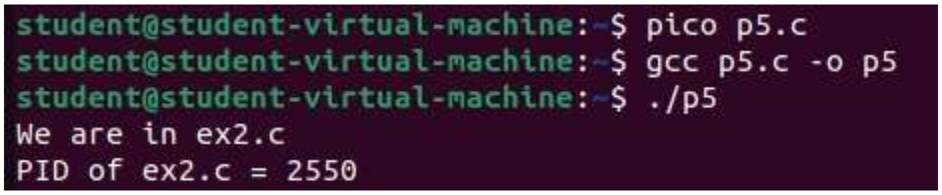


```
student@student-virtual-machine:~$ pico p4.c
student@student-virtual-machine:~$ gcc p4.c -o p4
student@student-virtual-machine:~$ ./p4
PID of ex1.c = 2477
Back to ex1.cstudent@student-virtual-machine:~$ 
```

Lab Code: 5

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

int main(int argc, char *argv[])
{
    printf("We are in ex2.c\n");
    printf("PID of ex2.c = %d\n", getpid());
    return 0;
}
```

Output:

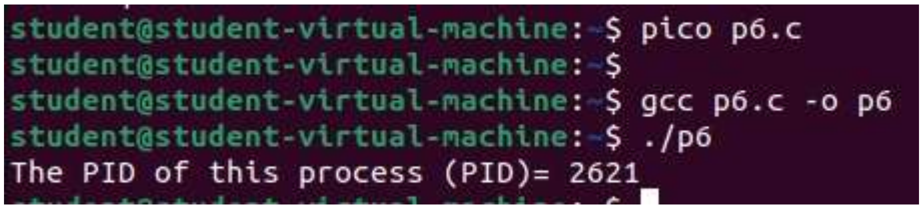


```
student@student-virtual-machine:~$ pico p5.c
student@student-virtual-machine:~$ gcc p5.c -o p5
student@student-virtual-machine:~$ ./p5
We are in ex2.c
PID of ex2.c = 2550
```

Lab Code: 6

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

int main(void)
{
 printf("The PID of this process (PID)= %d\n", getpid());
exit(0);
 printf("The PID of Parent process (PPID)= %d\n", getppid());
 return 0;
```

Output:

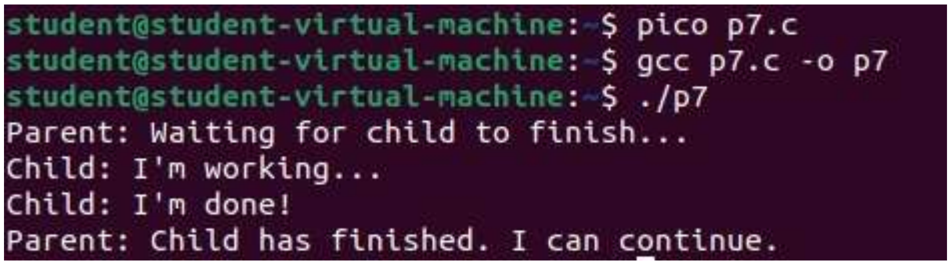


```
student@student-virtual-machine:~$ pico p6.c
student@student-virtual-machine:~$
student@student-virtual-machine:~$ gcc p6.c -o p6
student@student-virtual-machine:~$ ./p6
The PID of this process (PID)= 2621
```

Lab Code: 7

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child: I'm working...\n");
        sleep(2); // simulate work
        printf("Child: I'm done!\n");
        exit(0);
    } else {
        // Parent process
        printf("Parent: Waiting for child to finish...\n");
        wait(NULL); // wait for the child
        printf("Parent: Child has finished. I can continue.\n");
    }

    return 0;
}
```

Output:



```
student@student-virtual-machine:~$ pico p7.c
student@student-virtual-machine:~$ gcc p7.c -o p7
student@student-virtual-machine:~$ ./p7
Parent: Waiting for child to finish...
Child: I'm working...
Child: I'm done!
Parent: Child has finished. I can continue.
```

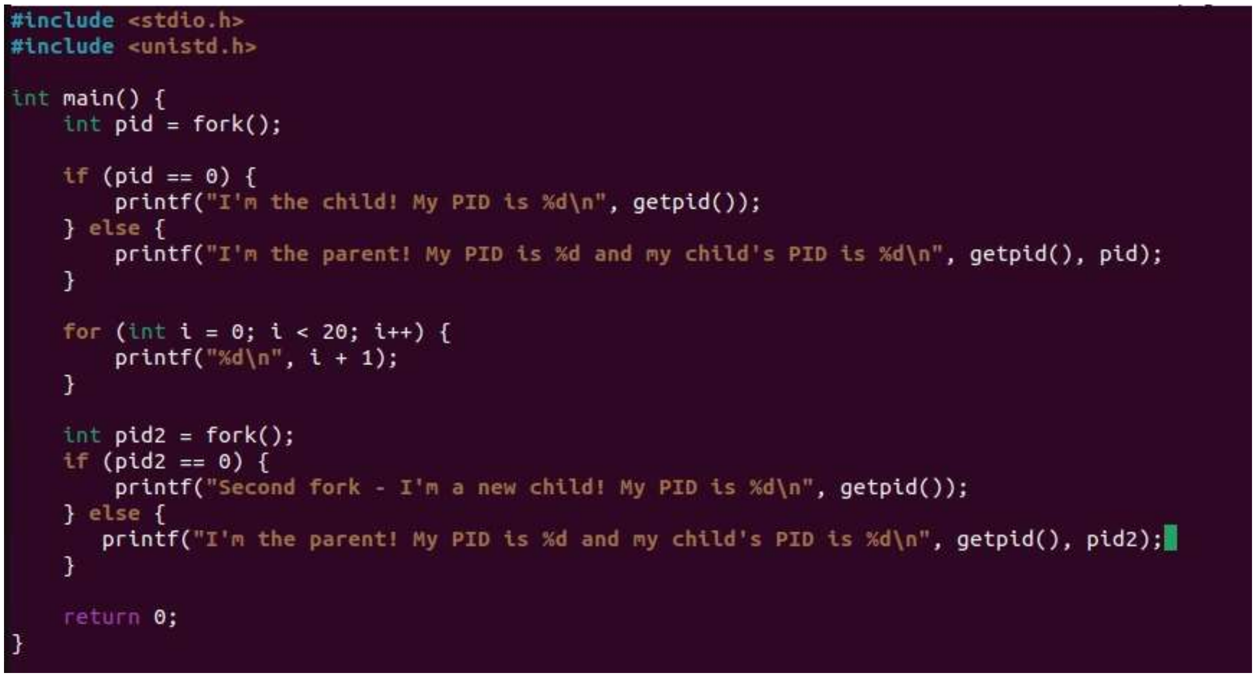
<u>TASKS</u>

Task: 01

Write a C++ program that uses two fork() calls . Each process should:

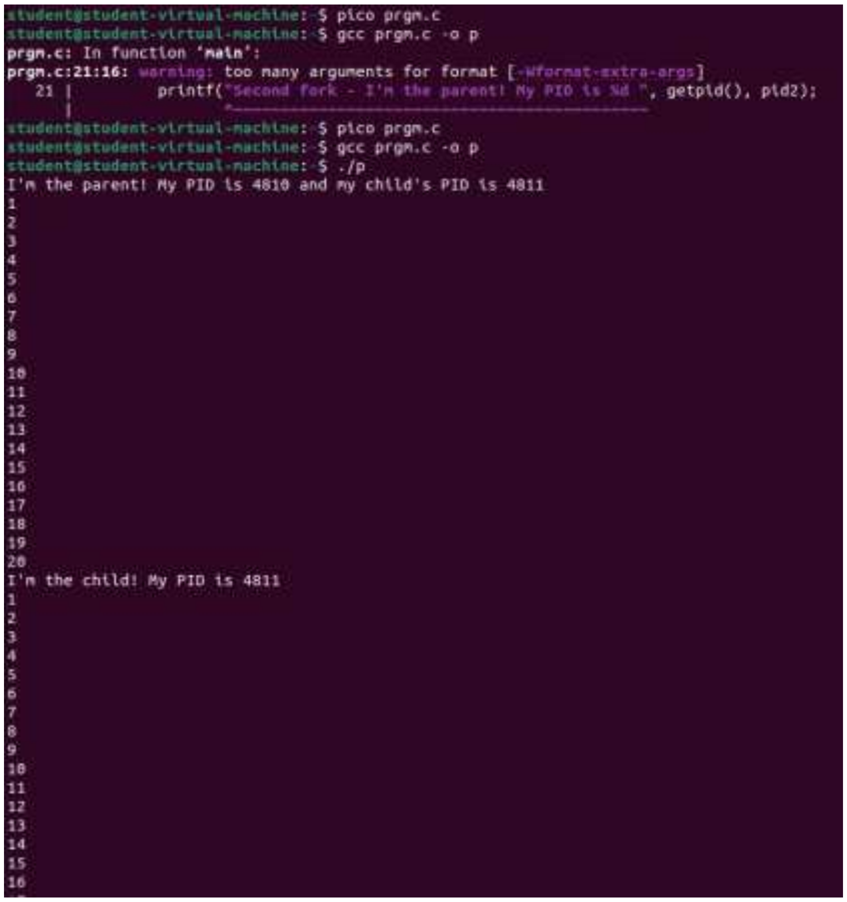
1. Print its process ID (PID) and a loop value from 1 to 20.

Code:

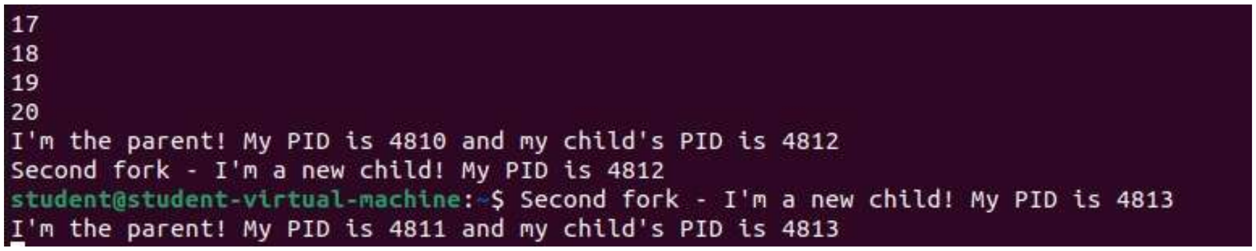


```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid = fork();

    if (pid == 0) {
        printf("I'm the child! My PID is %d\n", getpid());
    } else {
        printf("I'm the parent! My PID is %d and my child's PID is %d\n", getpid(), pid);
    }

    for (int i = 0; i < 20; i++) {
        printf("%d\n", i + 1);
    }

    int pid2 = fork();
    if (pid2 == 0) {
        printf("Second fork - I'm a new child! My PID is %d\n", getpid());
    } else {
       printf("I'm the parent! My PID is %d and my child's PID is %d\n", getpid(), pid2);
    }

    return 0;
}
```

Output:



```
student@student-virtual-machine:~$ pico prgm.c
student@student-virtual-machine:~$ gcc prgm.c -o p
prgm.c: In function 'main':
prgm.c:21:16: warning: too many arguments for format [-Wformat-extra-args]
   21 |         printf("Second fork - I'm the parent! My PID is %d ", getpid(), pid2);
      |                ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
student@student-virtual-machine:~$ pico prgm.c
student@student-virtual-machine:~$ gcc prgm.c -o p
student@student-virtual-machine:~$ ./p
I'm the parent! My PID is 4810 and my child's PID is 4811
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
I'm the child! My PID is 4811
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```



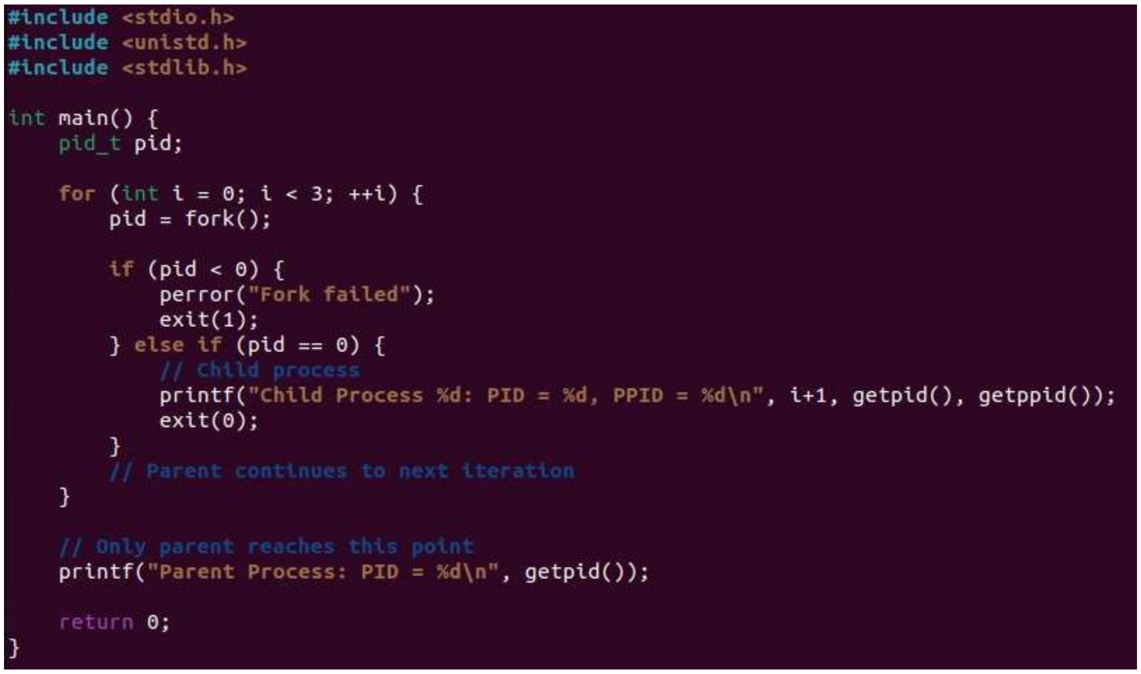
```
17
18
19
20
I'm the parent! My PID is 4810 and my child's PID is 4812
Second fork - I'm a new child! My PID is 4812
student@student-virtual-machine:~$ Second fork - I'm a new child! My PID is 4813
I'm the parent! My PID is 4811 and my child's PID is 4813
```

Task: 02

Write a C++ program that creates three child processes using the fork() system call. Each child process should:

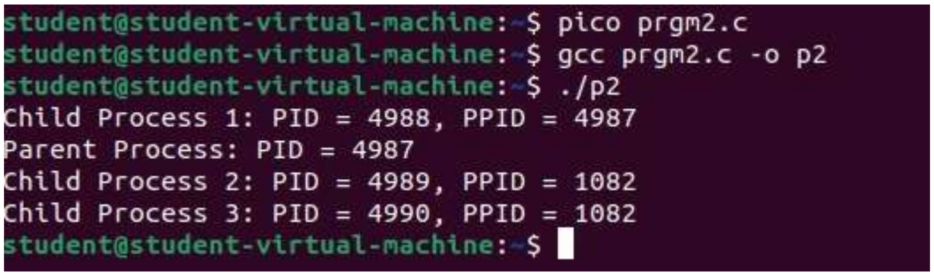
1. Print its own process ID (PID) and its parent process ID (PPID).
2. Terminate using exit().
3. After creating the child processes, the parent process should print its own PID.

Code:



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

int main() {
    pid_t pid;

    for (int i = 0; i < 3; ++i) {
        pid = fork();

        if (pid < 0) {
            perror("Fork failed");
            exit(1);
        } else if (pid == 0) {
            // Child process
            printf("Child Process %d: PID = %d, PPID = %d\n", i+1, getpid(), getppid());
            exit(0);
        }
        // Parent continues to next iteration
    }

    // Only parent reaches this point
    printf("Parent Process: PID = %d\n", getpid());

    return 0;
}
```

Output:



```
student@student-virtual-machine:~$ pico prgm2.c
student@student-virtual-machine:~$ gcc prgm2.c -o p2
student@student-virtual-machine:~$ ./p2
Child Process 1: PID = 4988, PPID = 4987
Parent Process: PID = 4987
Child Process 2: PID = 4989, PPID = 1082
Child Process 3: PID = 4990, PPID = 1082
student@student-virtual-machine:~$
```

Task: 03

Explain the working of system calls with its types and examples according to your understanding.

System calls are like bridges between a user program and the operating system (OS) kernel. Programs can't directly talk to hardware (like printer, memory, CPU). They **ask the OS to do it** for them using system calls. Examples:

- Read/write from files
- Start or stop processes
- Send messages over a network
- Access devices like scanners or printers

<u>Types of System Calls (with Examples):</u>

1. Process Control

Used to create, terminate, or manage processes.

Examples:

fork() – creates a new child process

exec() – runs a new program inside a process

wait() – waits for a child process to finish

exit() – ends a process

2. File Management

Used to work with files (create, open, read, write, etc.)

Examples:

open(), read(), write(), close()

3. Device Management

Access or control hardware devices.

Examples:

ioctl(), read(), write() (used with device drivers)

4. Information Maintenance

Get or set system data like time, process ID, user ID, etc.

Examples:

getpid(), getppid(), getuid()

5. Communication

For sending and receiving information between processes (Inter-process communication).

Examples:

pipe(), shmget(), msgget()