Experiment No.4
Create a child process in Linux using the fork system call.
Date of Performance:
Date of Submission:

Aim: Create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system call.

Objective: The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call.

Theory:

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. This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system.

System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a **new** process, which becomes the child process of the caller.

- If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the **process ID** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.

If the call to **fork()** is executed successfully, Unix will make two identical copies of address spaces, one for the parent and the other for the child.

getpid, getppid - get process identification

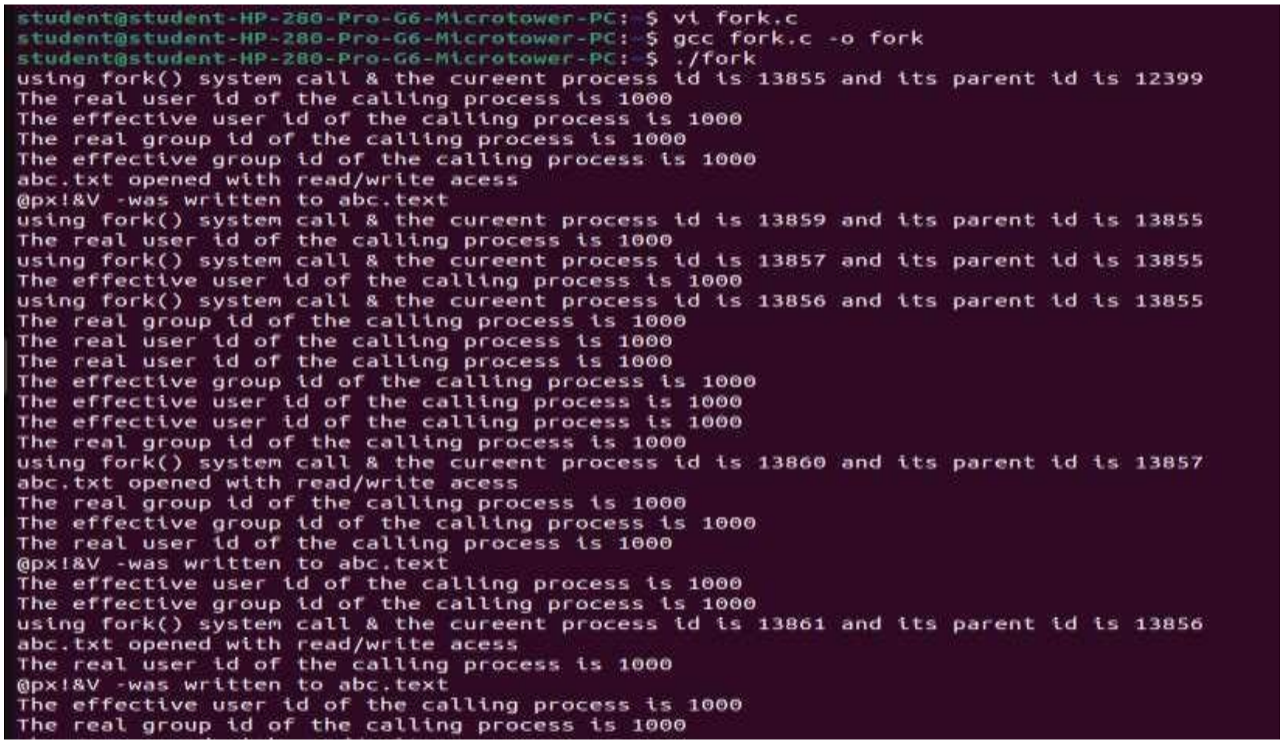
- **getpid()** returns the process ID (PID) of the calling process. This is often used by routines that generate unique temporary filenames.
- **getppid()** returns the process ID of the parent of the calling process. This will be either the ID of the process that created this process using fork().

Result:

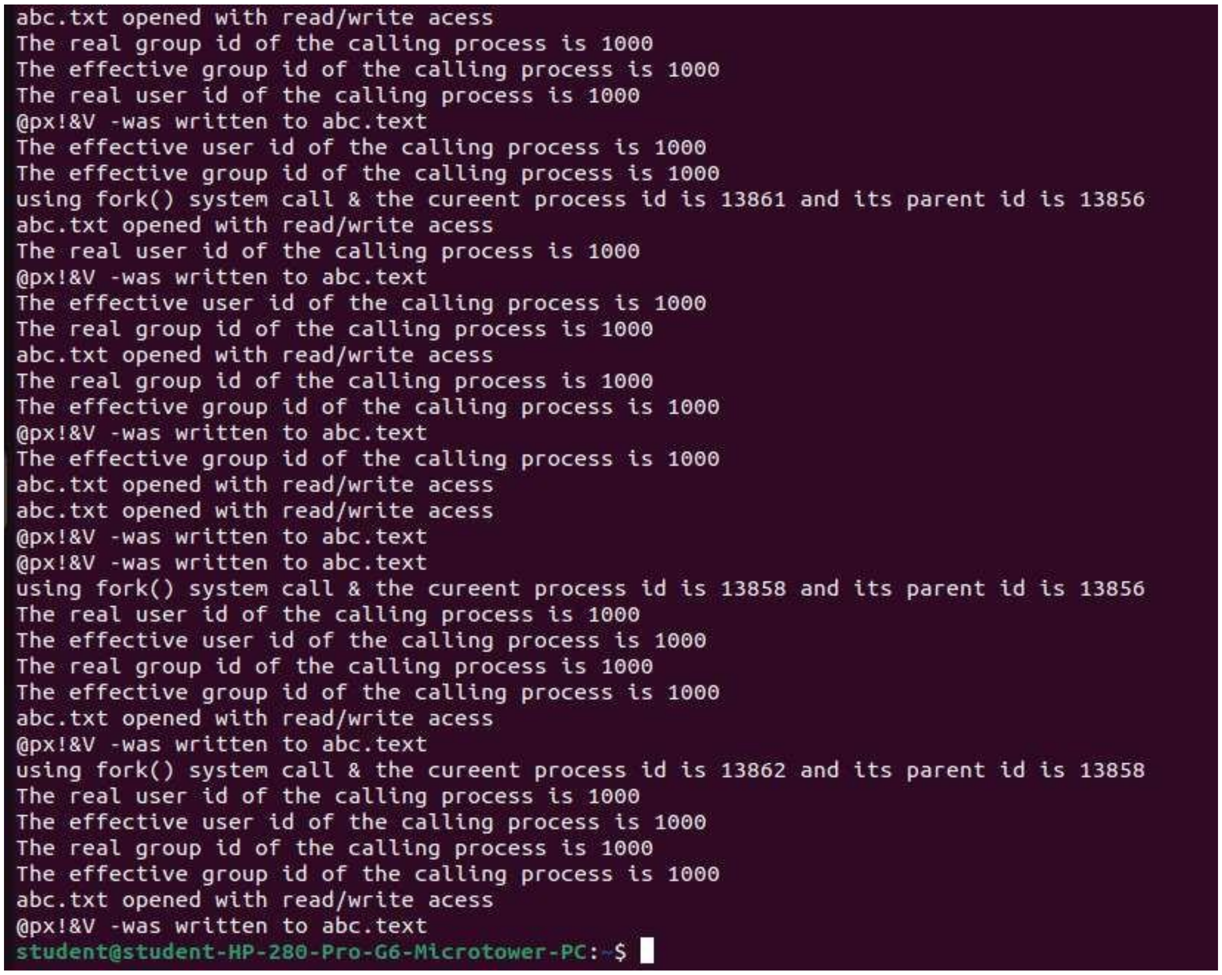
```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

    pid_t p, p1, p2;

    int num1 = 10, num2 = 20, num3 = 15;

    int largest, smallest;

    // Find largest number

    if (num1 >= num2 && num1 >= num3)

        largest = num1;

    else if (num2 >= num1 && num2 >= num3)

        largest = num2;

    else

        largest = num3;

    // Find smallest number

    if (num1 <= num2 && num1 <= num3)

        smallest = num1;

    else if (num2 <= num1 && num2 <= num3)

        smallest = num2;

    else

        smallest = num3;

    printf("The largest number is %d\n", largest);
    printf("The smallest number is %d\n", smallest);
    fork();
    fork();
    fork();
    p = getpid();
    p1 = getppid();
    printf("Current process id is %d and its parent id is %d \n", p, p1);
    p2 = getuid();
    printf("The real user id of the calling process is %d \n", p2);
    p2 = geteuid();
    printf("The effective user id of the calling process is %d \n", p2);
    p2 = getgid();
    printf("The real group id of the calling process is %d \n", p2);
    p2 = getegid();
    printf("The effective group id of the calling process is %d \n", p2);
    int fd;
    char buffer[80];
    static char message[ ] = "Comparison done";
    fd = open("abc.txt", O_RDWR);
    if (fd != -1) {
        printf("abc.txt opened with read/write access \n");
        write(fd, message, sizeof(message));
        lseek(fd, 0, 0);
        read(fd, buffer, sizeof(message));
```

```
        printf("%s - was written to abc.txt\n", buffer);

        close(fd);

    } else {

        printf("Failed to open abc.txt\n");

    }

    return 0;

}
```

OUTPUT



```
student@student-HP-280-Pro-G6-Microtower-PC:~$ vi fork.c
student@student-HP-280-Pro-G6-Microtower-PC:~$ gcc fork.c -o fork
student@student-HP-280-Pro-G6-Microtower-PC:~$ ./fork
using fork() system call & the cureent process id is 13855 and its parent id is 12399
The real user id of the calling process is 1000
The effective user id of the calling process is 1000
The real group id of the calling process is 1000
The effective group id of the calling process is 1000
abc.txt opened with read/write acess
@px!&V -was written to abc.text
using fork() system call & the cureent process id is 13859 and its parent id is 13855
The real user id of the calling process is 1000
using fork() system call & the cureent process id is 13857 and its parent id is 13855
The effective user id of the calling process is 1000
using fork() system call & the cureent process id is 13856 and its parent id is 13855
The real group id of the calling process is 1000
The real user id of the calling process is 1000
The real user id of the calling process is 1000
The effective group id of the calling process is 1000
The effective user id of the calling process is 1000
The effective user id of the calling process is 1000
The real group id of the calling process is 1000
using fork() system call & the cureent process id is 13860 and its parent id is 13857
abc.txt opened with read/write acess
The real group id of the calling process is 1000
The effective group id of the calling process is 1000
The real user id of the calling process is 1000
@px!&V -was written to abc.text
The effective user id of the calling process is 1000
The effective group id of the calling process is 1000
using fork() system call & the cureent process id is 13861 and its parent id is 13856
abc.txt opened with read/write acess
The real user id of the calling process is 1000
@px!&V -was written to abc.text
The effective user id of the calling process is 1000
The real group id of the calling process is 1000
```



```
abc.txt opened with read/write acess
The real group id of the calling process is 1000
The effective group id of the calling process is 1000
The real user id of the calling process is 1000
@px!&V -was written to abc.text
The effective user id of the calling process is 1000
The effective group id of the calling process is 1000
using fork() system call & the cureent process id is 13861 and its parent id is 13856
abc.txt opened with read/write acess
The real user id of the calling process is 1000
@px!&V -was written to abc.text
The effective user id of the calling process is 1000
The real group id of the calling process is 1000
abc.txt opened with read/write acess
The real group id of the calling process is 1000
The effective group id of the calling process is 1000
@px!&V -was written to abc.text
The effective group id of the calling process is 1000
abc.txt opened with read/write acess
abc.txt opened with read/write acess
@px!&V -was written to abc.text
@px!&V -was written to abc.text
using fork() system call & the cureent process id is 13858 and its parent id is 13856
The real user id of the calling process is 1000
The effective user id of the calling process is 1000
The real group id of the calling process is 1000
The effective group id of the calling process is 1000
abc.txt opened with read/write acess
@px!&V -was written to abc.text
using fork() system call & the cureent process id is 13862 and its parent id is 13858
The real user id of the calling process is 1000
The effective user id of the calling process is 1000
The real group id of the calling process is 1000
The effective group id of the calling process is 1000
abc.txt opened with read/write acess
@px!&V -was written to abc.text
student@student-HP-280-Pro-G6-Microtower-PC:~$
```

Conclusion: In Unix-like operating systems, the `fork()` system function duplicates the current process to start a new one. It is utilized for concurrency, isolation, parallel processing, parallel execution, and process formation. Programs may execute tasks concurrently thanks to `fork()}, effectively splitting workloads. It makes it possible to build concurrent and multitasking processes, each with its own memory and resources. Furthermore, `fork()} is necessary for networking applications that leverage forking servers and the creation of background processes.