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

Aim: Create a child process in Linux using the fork system call.

Objective:

Create a child process using fork system call.

From the child process obtain the process ID of both child and parent by using getpid and getppid system calls.

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

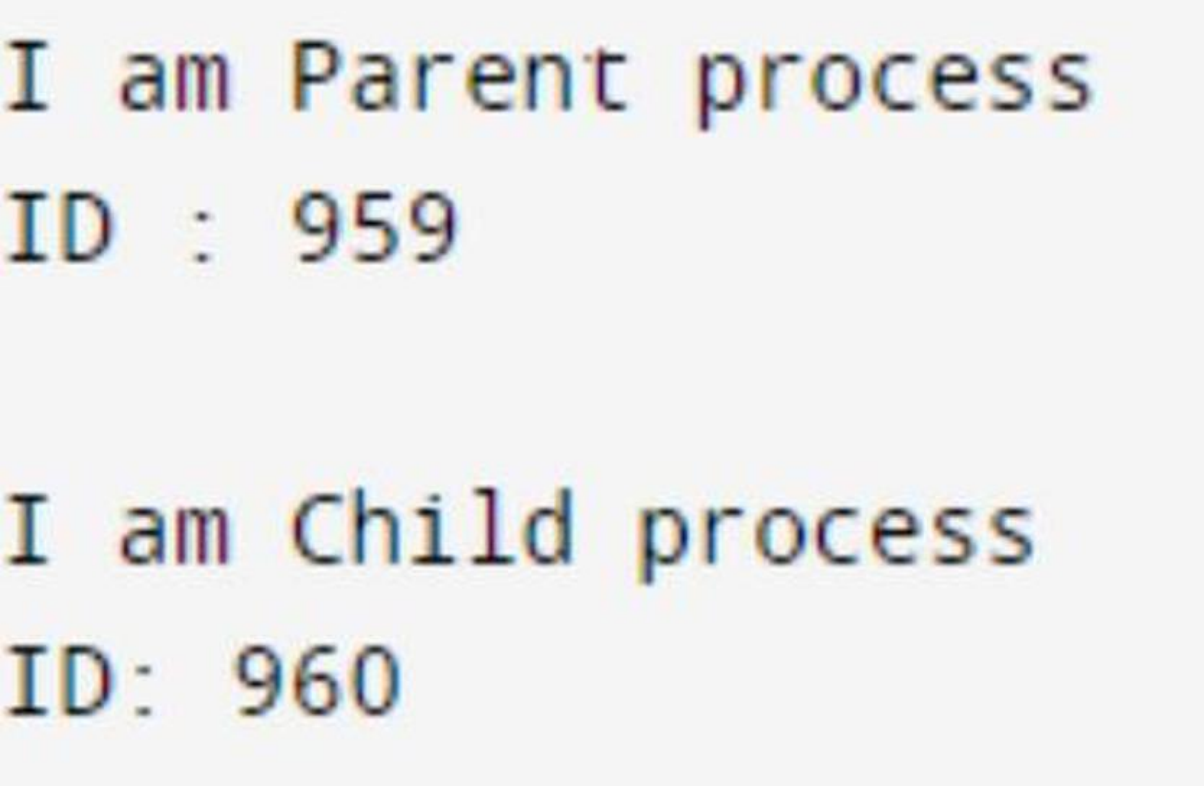
- **getpid()** returns the process ID (PID) of the calling process. This is often used by routines that generate unique temporary filenames.
- **getppid()** returns the process ID of the parent of the calling process. This will be either the ID of the process that created this process using fork().

Program:

```
#include<stdio.h>
 #include<sys/types.h>
 #include <unistd.h>
int main() {
// fork() Create a child process
int pid = fork();
if (pid > 0) {
        printf("I am Parent process\n");
        printf("ID : %d\n\n", getpid());
} else if (pid == 0) {
printf("I am Child process\n");
// getpid() will return process id of child process
printf("ID: %d\n", getpid());

 } else {
 printf("Failed to create child process");
}
 return 0;
```

Output:



Conclusion:

What do you mean by system call?

A system call serves as a vital communication link between user-level applications and the kernel of an operating system. It facilitates the transfer of control from user space, where applications reside, to kernel space, where the operating system's core functionalities operate. This transition enables user programs to access privileged operations and resources, such as file I/O, network communication, process management, and hardware control, which are typically restricted to the operating system's domain. System calls follow a predefined interface and protocol, allowing applications to request services from the operating system in a standardized manner. Upon receiving a system call request, the kernel executes the requested operation on behalf of the application, ensuring proper security, resource management, and coordination with other processes. Thus, system calls play a fundamental role in enabling user programs to interact with and harness the full capabilities of the underlying operating system and hardware infrastructure.