|  |
| --- |
| Experiment No.4 |
| Create a child process in Linux using the fork system call. |
| Date of Performance: 23/02/2024 |
| Date of Submission: 1/03/2024 |

**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](https://en.wikipedia.org/wiki/Computer_program) requests a service from the [kernel](https://en.wikipedia.org/wiki/Kernel_(computing)) of the [operating system](https://en.wikipedia.org/wiki/Operating_system) it is executed on. This may include hardware-related services (for example, accessing a [hard disk drive](https://en.wikipedia.org/wiki/Hard_disk_drive)), creation and execution of new [processes](https://en.wikipedia.org/wiki/Process_(computing)), and communication with integral [kernel services](https://en.wikipedia.org/wiki/Kernel_service) such as [process scheduling](https://en.wikipedia.org/wiki/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>

#include<fcntl.h>

int main(){

pid\_t p,p1,p2;

fork();

fork();

fork();

p=getpid();

p1=getppid();

printf("Using fork() system call, the current process id is %d and its parent process 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 of calling process is %d\n",p2);

p2=getegid();

printf("The effective group id of the calling process is %d\n",p2);

//Implementation of open(),close(), read() and write()

int fd;

char buffer[80];

static char message[] = "Hello";

fd=open("test.txt",O\_RDWR);

if(fd!=1){

printf("test.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 test.txt\n",buffer);

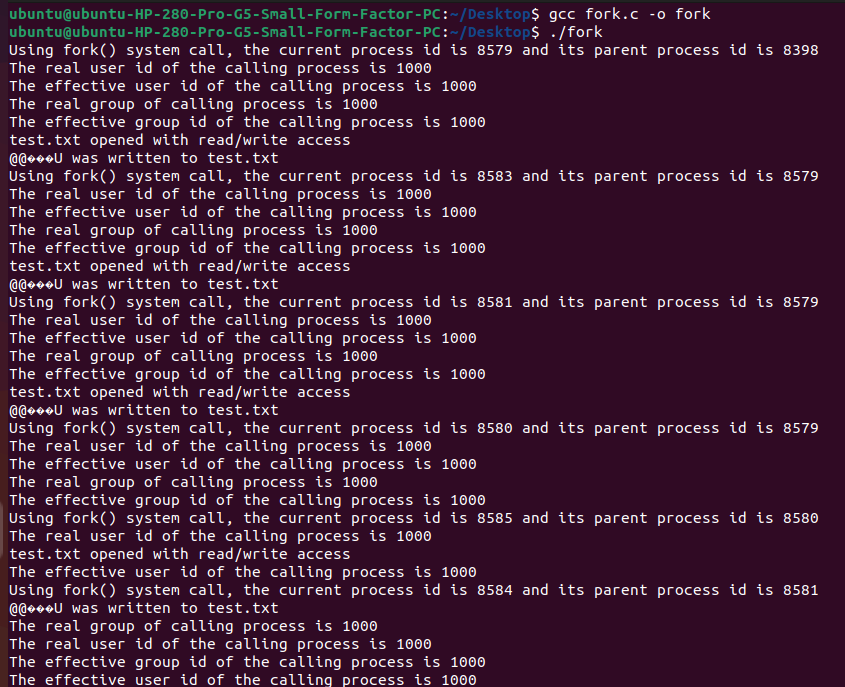
close(fd);

}

return 0;

}

**Result:**

****

**Conclusion:**

In conclusion , the purpose of the fork() system call is to create a new process (child process) that is a copy of the current process (parent process). This allows for parallel execution of multiple tasks within a program, enabling multitasking and concurrent processing. Fork() is fundamental for creating independent processes that can execute different tasks simultaneously, enhancing the efficiency and versatility of multi-process programming paradigms.