Name: Abdullah Khan

Roll no: 122A1002

Branch: CE

Batch: C1

Experiment #4

Aim: a) Create a child process in Linux using the fork system call. From the child process obtainthe process ID of both child and parent by using getpid and getppid system call. Explore wait andwaitpid before termination of process. b) Explore the following system calls: open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid..

Theory:

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. After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()***system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of **fork ()**:

- 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.
- Therefore, after the system call to **fork()**, a simple test can tell which process is the child.

To get a process name, given its pid:

```
pid=42
ps -o comm= -p $pid
```
To get the names of the child processes of a given pid (Linux procps):
```
ps -o comm= --ppid $pid
```

Wait and Waitpid: The **waitpid**() system call suspends execution of the current process until a child specified by *pid*argument has changed state. By default, **waitpid**() waits only for terminated children, but this behaviour is modifiable via the *options* argument.

The **wait**() system call suspends execution of the current process until one of its children terminates. The call *wait(&status)* is equivalent to:

```
waitpid(-1, &status, 0);
```

Open(): A call to **open**() creates a new *open file description*, an entry in the system-wide table of open files. A file descriptor is a reference to one of these entries; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file.
read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

write() writes up to *count* bytes to the file referenced by the file descriptor *fd* from the buffer starting at *buf*.

close() closes a file descriptor, so that it no longer refers to any file and may be reused.
getpid() returns the process ID of the current process.
getuid() returns the real user ID of the current process.
geteuid() returns the effective user ID of the current process.
getgid() returns the real group ID of the current process.
getegid() returns the effective group ID of the current process.

Running C program to execute various system call in Ubuntu.

1. Step 1: Install the build-essential packages. ...
2. Step 2: Write a simple **C** program. ...
3. Step 3: **Compile** the **C** program with gcc Compiler. ...
4. Step 4: **Run** the program.

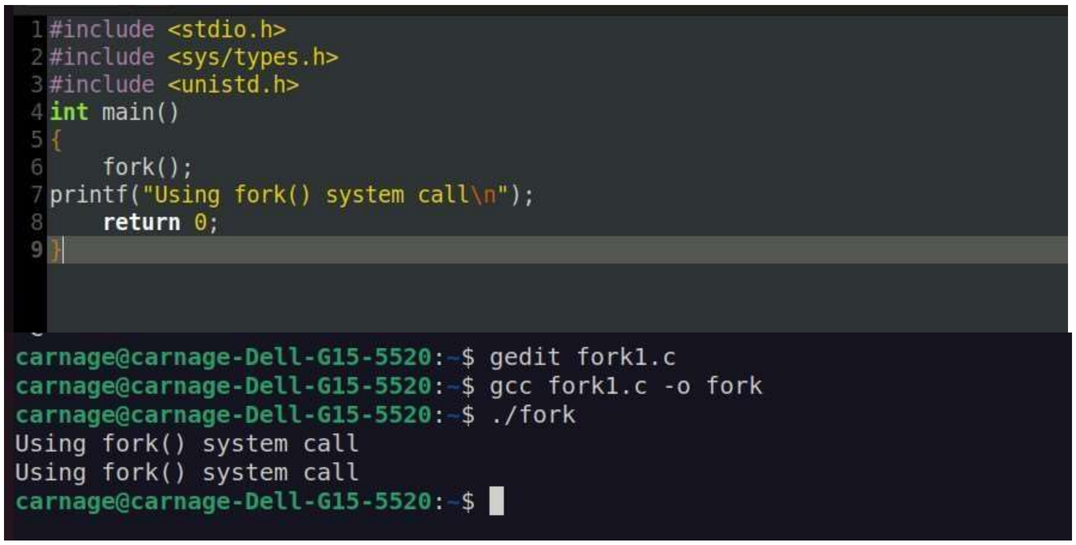
Running Fork:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   fork();
   printf("Using fork() system call\n");
   return 0;
}
```

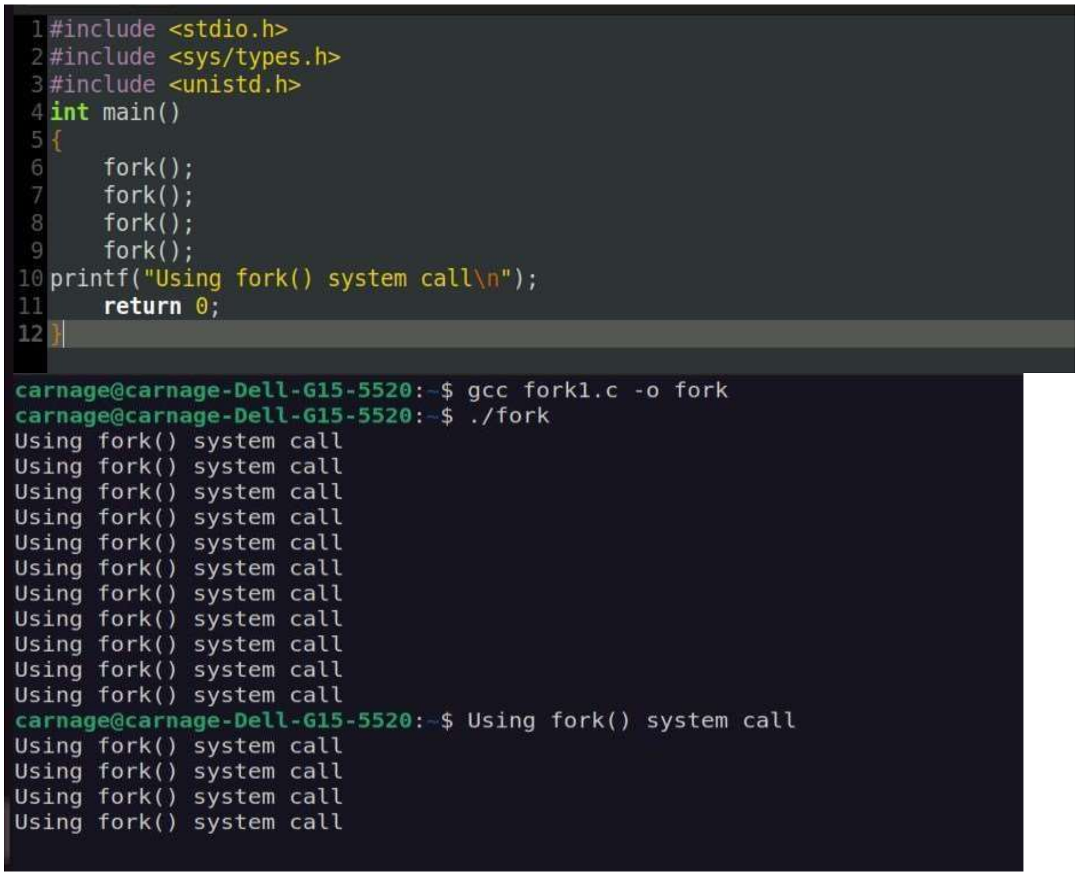
Output:

```
Using fork() system call
Using fork() system call
```



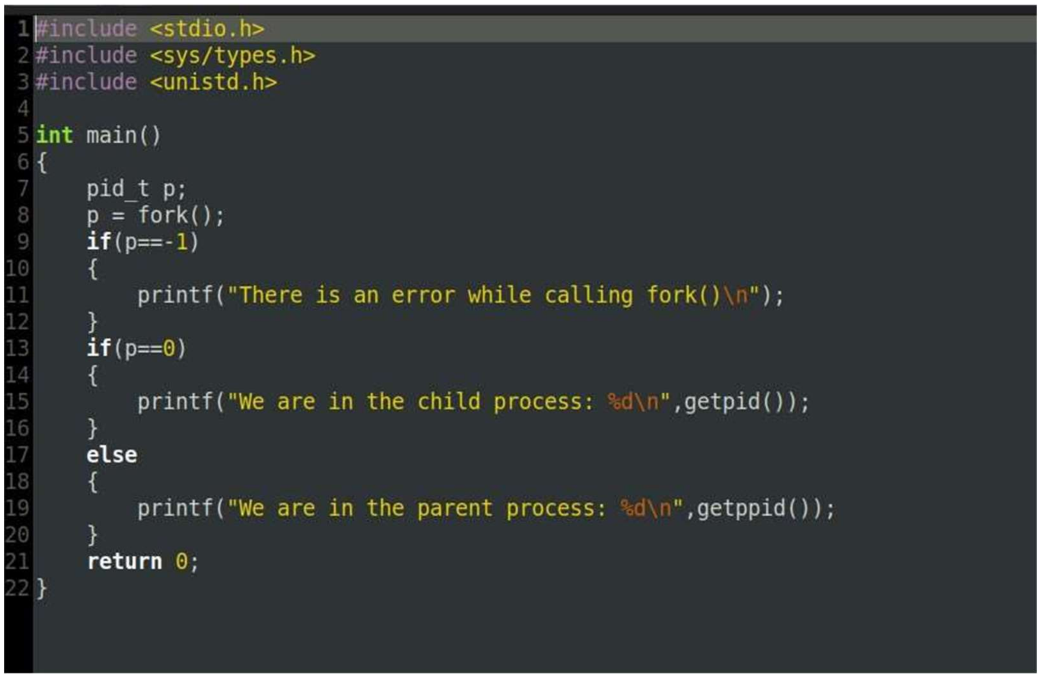
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
printf("Using fork() system call\n");
    return 0;
}
```

```
carnage@carnage-Dell-G15-5520:~$ gedit fork1.c
carnage@carnage-Dell-G15-5520:~$ gcc fork1.c -o fork
carnage@carnage-Dell-G15-5520:~$ ./fork
Using fork() system call
Using fork() system call
carnage@carnage-Dell-G15-5520:~$ 
```



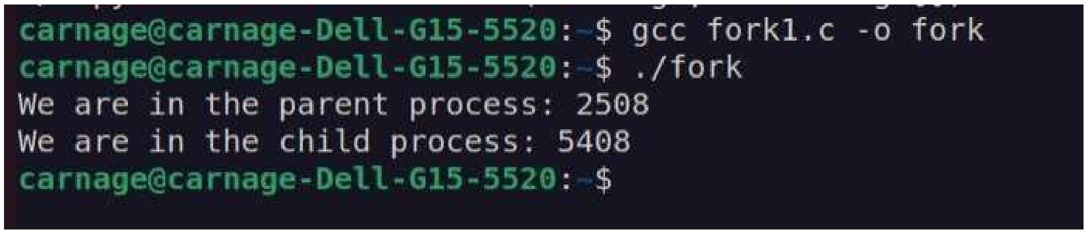
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    fork();
printf("Using fork() system call\n");
    return 0;
}
```

```
carnage@carnage-Dell-G15-5520:~$ gcc fork1.c -o fork
carnage@carnage-Dell-G15-5520:~$ ./fork
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
carnage@carnage-Dell-G15-5520:~$ Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
```



```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    pid_t p;
    p = fork();
    if(p==-1)
    {
        printf("There is an error while calling fork()\n");
    }
    if(p==0)
    {
        printf("We are in the child process: %d\n",getpid());
    }
    else
    {
        printf("We are in the parent process: %d\n",getppid());
    }
    return 0;
}
```



```
carnage@carnage-Dell-G15-5520:~$ gcc fork1.c -o fork
carnage@carnage-Dell-G15-5520:~$ ./fork
We are in the parent process: 2508
We are in the child process: 5408
carnage@carnage-Dell-G15-5520:~$
```

https://linuxhint.com/fork-system-call-linux/#:~:text=The%20syntax%20of%20fork(),to%20the%20child%20process%20itself.

Conclusion: Using Above command a shell scripts is created for the same.