Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

Academic Year : 2023-24

NAME:-JENIL KOTADIA

ROLL NO:-19

Experiment no 4

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.

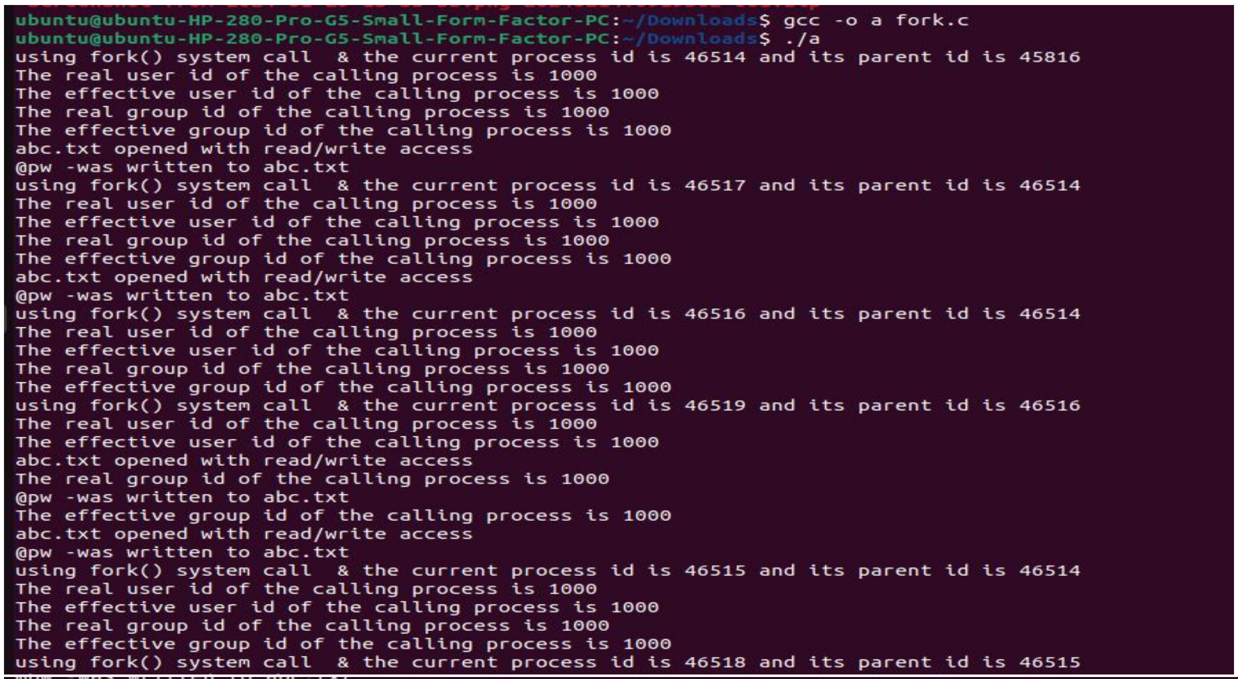
If the call to fork() is executed successfully, Unix will make two identical copies of address spaces, one for the parent and the other for the child.

getpid, getppid - get process identification

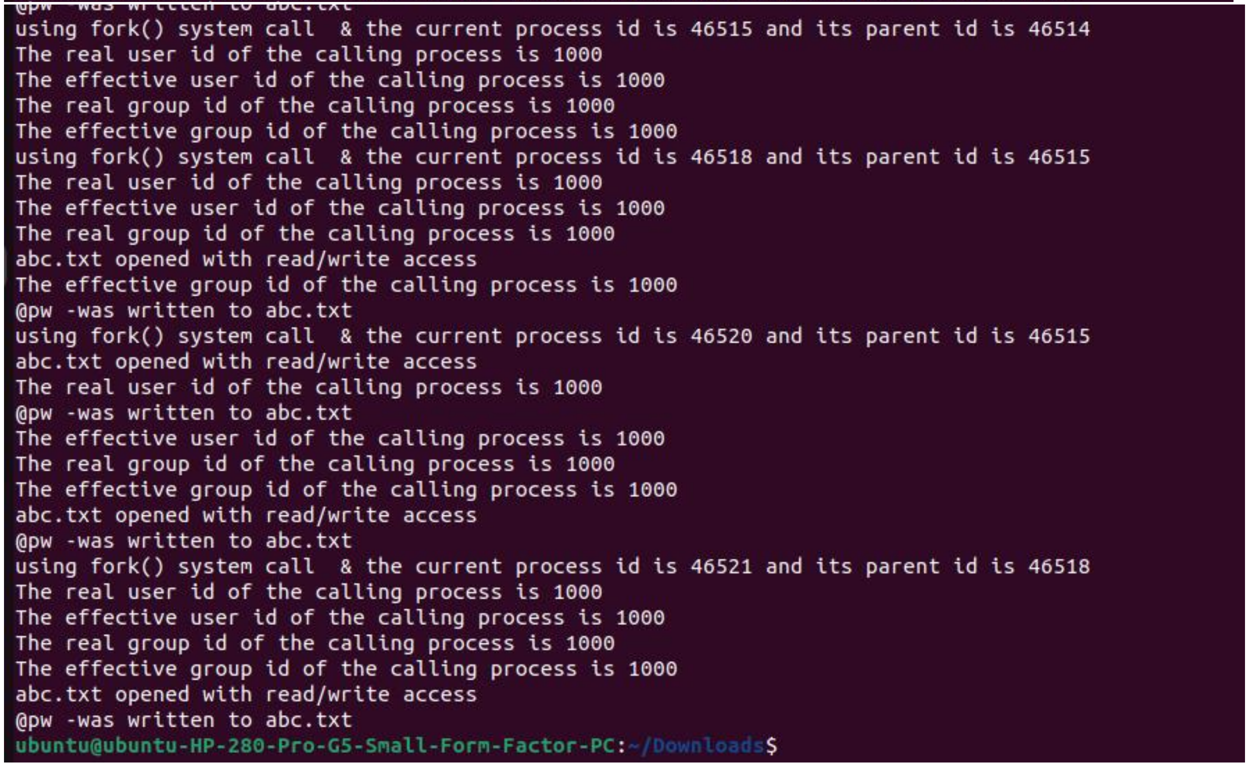
getpid() returns the process ID (PID) of the calling process. This is often used by routines that generate unique temporary filenames.

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().

Code:-



```
ubuntu@ubuntu-HP-280-Pro-G5-Small-Form-Factor-PC:~/Downloads$ gcc -o a fork.c
ubuntu@ubuntu-HP-280-Pro-G5-Small-Form-Factor-PC:~/Downloads$ ./a
using fork() system call  & the current process id is 46514 and its parent id is 45816
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 access
@pw -was written to abc.txt
using fork() system call  & the current process id is 46517 and its parent id is 46514
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 access
@pw -was written to abc.txt
using fork() system call  & the current process id is 46516 and its parent id is 46514
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
using fork() system call  & the current process id is 46519 and its parent id is 46516
The real user id of the calling process is 1000
The effective user id of the calling process is 1000
abc.txt opened with read/write access
The real group id of the calling process is 1000
@pw -was written to abc.txt
The effective group id of the calling process is 1000
abc.txt opened with read/write access
@pw -was written to abc.txt
using fork() system call  & the current process id is 46515 and its parent id is 46514
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
using fork() system call  & the current process id is 46518 and its parent id is 46515
```



```
using fork() system call  & the current process id is 46515 and its parent id is 46514
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
using fork() system call  & the current process id is 46518 and its parent id is 46515
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
abc.txt opened with read/write access
The effective group id of the calling process is 1000
@pw -was written to abc.txt
using fork() system call  & the current process id is 46520 and its parent id is 46515
abc.txt opened with read/write access
The real user id of the calling process is 1000
@pw -was written to abc.txt
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 access
@pw -was written to abc.txt
using fork() system call  & the current process id is 46521 and its parent id is 46518
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 access
@pw -was written to abc.txt
ubuntu@ubuntu-HP-280-Pro-G5-Small-Form-Factor-PC:~/Downloads$
```

Conclusion: The **fork()** system call in Unix-like operating systems serves the purpose of creating a new process, known as the child process, which is an exact copy of the current process, known as the parent process. This system call allows for process creation and the execution of separate tasks concurrently within a program or system. In short, the primary purpose of the **fork()** system call is to facilitate process creation and parallel execution, enabling multitasking and concurrent execution of tasks within a Unix-based environment.