Name: - Atharva Manoj Kulkarni

PRN: - 2019BTEIT00076

**Chapter 1 Processing Environment**

**1.1 - Write the application or program to open applications of Linux by creating new processes using fork system call. Comment on how various application’s/command’s process get created in linux.**

**Objectives:**

1. To learn about Processing Environment.
2. To know the difference between fork/vfork and various execs variations.
3. Use of system call to write effective programs.

**Theory:**

How various application’s/command’s process get created in linux?

A process begins its life when it is created. A process goes through different states before it gets terminated. The first state that any process goes through is the creation of itself. Process creation happens through the use of fork() system call, which creates a new process(child process) by duplicating an existing one(parent process). The process that calls fork() is the parent, whereas the new process is the child. In most cases, we may want to execute a different program in child process than the parent. The exec() family of function calls creates a new address space and loads a new program into it. Finally, a process exits or terminates using the exit() system call. This function frees all the resources held by the process(except for pcb). A parent process can enquire about the status of a terminated child using wait() system call. When the parent process uses wait() system call, the parent process is blocked till the child on which it is waiting terminates.

**Fork() System Call**

When a parent process uses fork(), it creates a duplicate copy of itself and this duplicates becomes the child of the process. The fork() is implemented using clone() system call in linux which returns twice from kernel.

• A non-zero value(Process ID of child) is returned to the parent.

• A value of zero is returned to the child.

• In case the child is not created successfully due to any issues like low memory, -1 is returned to the fork().

**Flowchart:**

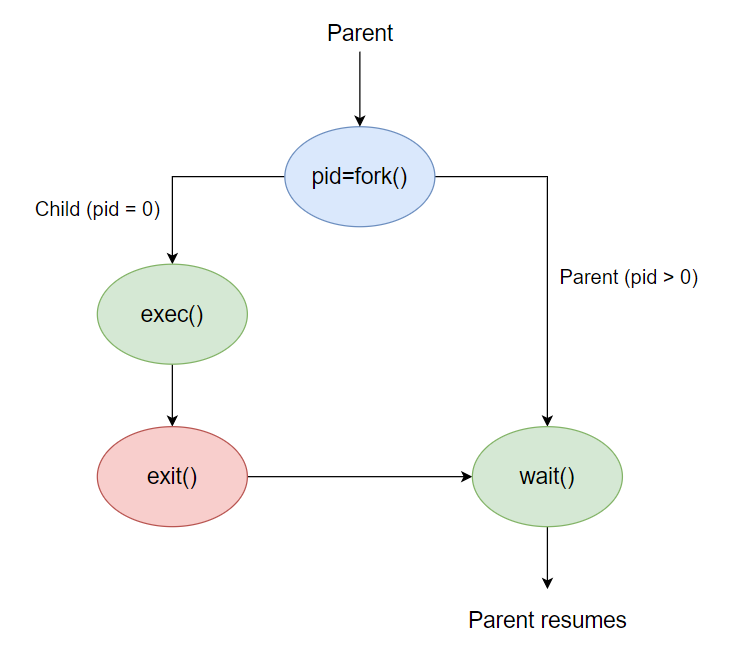


Fig: 1.1 Flowchart of fork

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr | Variable/Function | Datatype | Use |
| Number |  |  |  |
|  |  |  |  |
| 1 | Counter | int | Used to increment number of child and parent |
|  |  |  | Processes in the for loop. |
|  |  |  |  |
| 2 | pid | int | Process ID |
|  |  |  |  |

Fig:1.1 Data Dictionary

**Program:**

#include<stdio.h>

#include<unistd.h>

int main(){

printf("Beginning\n");

int counter = 0;

int pid = fork();

if(pid==0)

{

for(int i=0;i<5;i++)

{

printf("Child process = %d\n",++counter);

}

printf("Child Ended\n");

}

else if(pid>0)

{

for(int i=0;i<5;i++)

{

printf("Parent process = %d\n",++counter);

}

printf("Parent Ended\n");

}

else

{

printf("fork() failed\n");

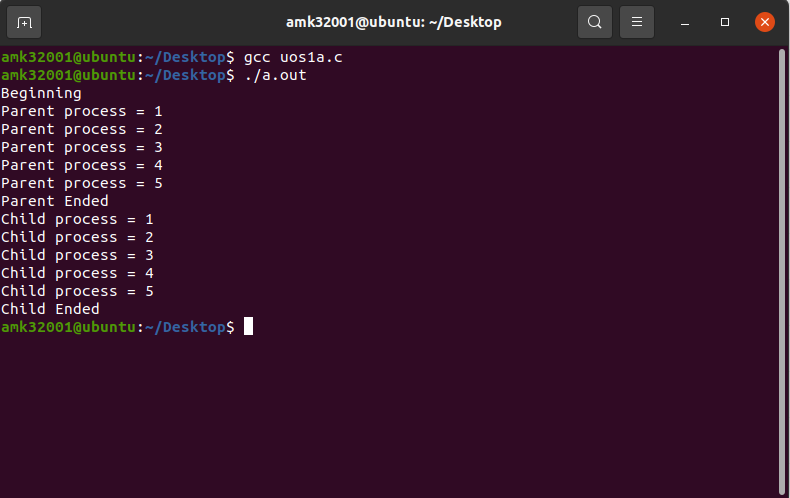
return 1;

}

return 0;

}

**Output:**



**Conclusion:**

* Fork system call can be used to create processes from a running process.
* These processes can be made to execute different application programs using various exec statements.

**References:**

[1] [Link 1](http://www.tutorialspoint.com/unix_system_calls/)

[2] [Link 2](https://users.cs.cf.ac.uk/Dave.Marshall/C/node22.html)