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 new process is created because an existing process makes an exact copy of itself. This child process has the same environment as its parent, only the process ID number is different. This procedure is called *forking*.

After the forking process, the address space of the child process is overwritten with the new process data. This is done through an *exec* call to the system.

The *fork-and-exec* mechanism thus switches an old command with a new, while the environment in which the new program is executed remains the same, including configuration of input and output devices, environment variables and priority. This mechanism is used to create all UNIX processes, so it also applies to the Linux operating system. Even the first process, **init**, with process ID 1, is forked during the boot procedure in the so-called *bootstrapping* procedure.

There are a couple of cases in which **init** becomes the parent of a process, while the process was not started by **init**, as we already saw in the **pstree** example. Many programs, for instance, ***daemonize*** their child processes, so they can keep on running when the parent stops or is being stopped. A window manager is a typical example; it starts an **xterm** process that generates a shell that accepts commands. The window manager then denies any further responsibility and passes the child process to **init**. Using this mechanism, it is possible to change window managers without interrupting running applicatios.

Flowchart:

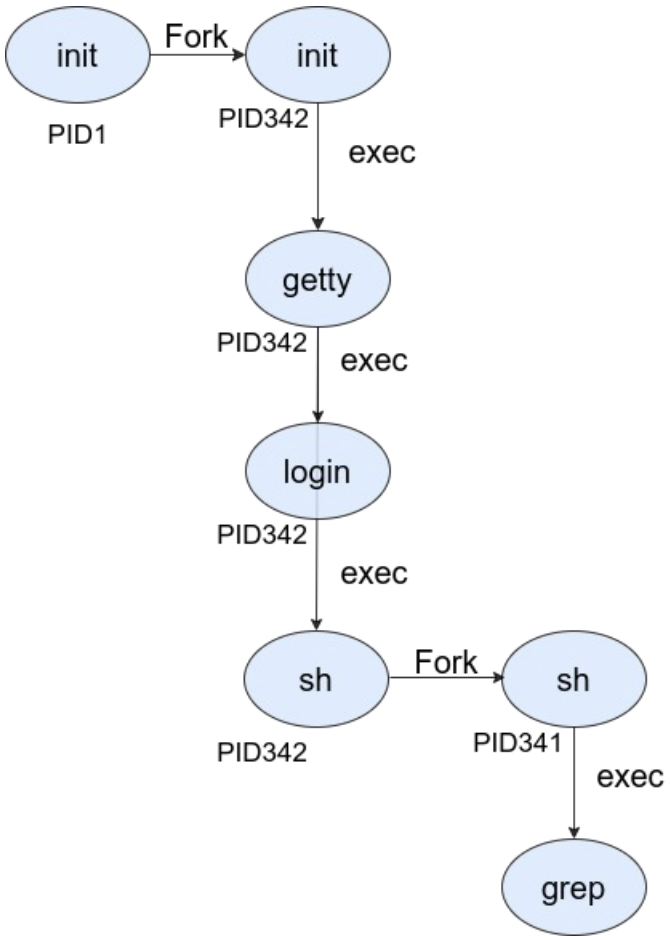

Fig: 1.1 Flowchart of fork

Data Dictionary:

Sr Number	Variable/Function	Datatype	Use
1	Counter	int	Used to increment number of child and parent processes.
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:

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1a$ gedit 1a.c

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1a$ gcc 1a.c

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1a$ ./a.out

Beginning

Parent process = 1

Parent process = 2

Parent process = 3

Parent process = 4

Parent process = 5

Parent Ended

Child process = 1

Child process = 2

Child process = 3

Child process = 4

Child process = 5

Child Ended

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] www.tutorialspoint.com/unix_system_calls/

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