Processing Environment

Subject:- Unix OperatingSystem

Class :- TYIT

Name PRN

Shraddha Kore 21620001

Assignment No - 1b

Title-Write the application or program to create Childs assign the task to them by variation exec system calls.

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-

fork vs exec:

The use of fork and exec exemplifies the spirit of UNIX in that it provides a very simple way to start new processes. The fork call basically makes a duplicate of the current process, identical in almost every way (not everything is copied over, for example, resource limits in some implementations but the idea is to create as close a copy as possible).

The new process (child) gets a different process ID (PID) and has the PID of the old process (parent) as its parent PID (PPID). Because the two processes are now running exactly the same code, they can tell which is which by the return code of fork - the child gets 0, the parent gets the PID of the child. This is all, of course, assuming the fork call works - if not, no child is created and the parent gets an error code.

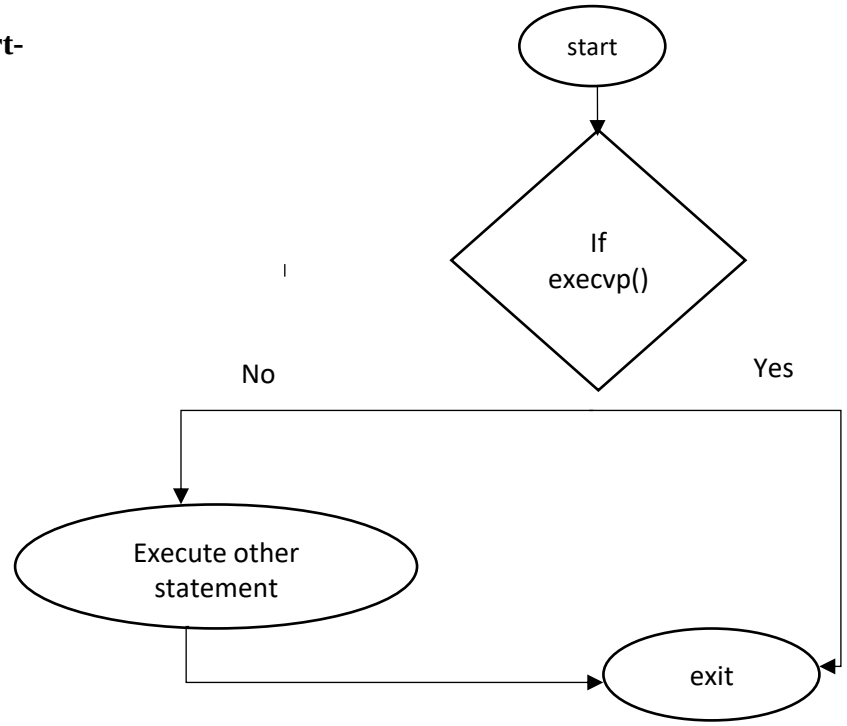
The exec call is a way to basically replace the entire current process with a new program. It loads the program into the current process space and runs it from the entry point. So, fork and exec are often used in sequence to get a new program running as a child of a current process. Shells typically do this whenever you try to run a program like find - the shell forks, then the child loads the find program into memory, setting up all command line arguments, standard I/O and so forth.

Processing Environment:

Process creation Unless the system is being bootstrapped a process can only come into existance as the child of another process. This done by the fork system call. The first process created is "hand tooled" by the boot process. This is the swapper process. The swapper process creates the init process, which is the ancestor of all further processes.

In particular, init forks off a process getty, which monitors terminal lines and allows users to log in. Upon login, the command shell is run as the first process. The command shell for a given user is specified in the /etc/passwd file. From thereon, any process may fork to produce new processes, considered to be children of the forking process. The process table and uarea Information about processes is described in two data structures, the kernel process table and a "uarea" associated with each process. The process table holds information required by the kernel The uarea holds information required by the process itself.

Flowchart-



Program-

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
char *args[]={"./EXEC",NULL};
execvp(args[0],args);
printf("Ending\n");
return 0;
}

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
printf("I am EXEC.c called by execvp()\n");
return 0;
}
```

Output-

```
shraddha@ shraddha:~/Desktop/UOS$ gcc 1b.c
shraddha@ shraddha:~/Desktop/UOS$ ./a.out
Ending
shraddha@ shraddha:~/Desktop/UOS$ gcc EXEC.c -o EXEC
shraddha@ shraddha:~/Desktop/UOS$ ./EXEC
I am EXEC.c called by execvp()
```

Conclusion:

Different versions of exec like execvp() can be used to assign task like running another program while fork just creates child which shares resources with parent and runs the same way as the parent.

References:

www.tutorialspoint.com/unix_system_calls/