1. Processing Environment

1.3 Write the program to use fork/vfork system call. Justify the difference by using suitable application of fork/vfork 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():

The **fork()** is a system call use to create a **new process.** The new process created by the fork() call is the child process, of the process that invoked the fork() system call. The code of child process is identical to the code of its parent process. After the creation of child process, both process i.e. parent and child process start their execution from the next statement after fork() and both the processes get executed **simultaneously.**

vfork():

The modified version of fork() is vfork(). The **vfork()** system call is also used to create a new process. Similar to the fork(), here also the new process created is the child process, of the process that invoked vfork(). The child process code is also identical to the parent process code. Here,the child process **suspends the execution** of parent process till it completes its execution as both the process share the same address space to use.

Comparison Chart:

Basis for Comparison	**fork()**	**vfork()**
Basic	Child process and parent process has separate address spaces.	Child process and parent process shares the same address space.
Execution	Parent and child process execute simultaneously.	Parent process remains suspended till child process completes its execution.
Modification	If the child process alters any page in the address space, it is invisible to the parent process as the address space are separate.	If child process alters any page in the address space, it is visible to the parent process as they share the same

Basis for Comparison	fork()	vfork()
		address space.
Copy-on-write	fork() uses copy-on-write as an alternative where the parent and child shares same pages until any one of them modifies the shared page.	vfork() does not use copy-on-write.

Table 1.3 fork and vfork

Flowchart:

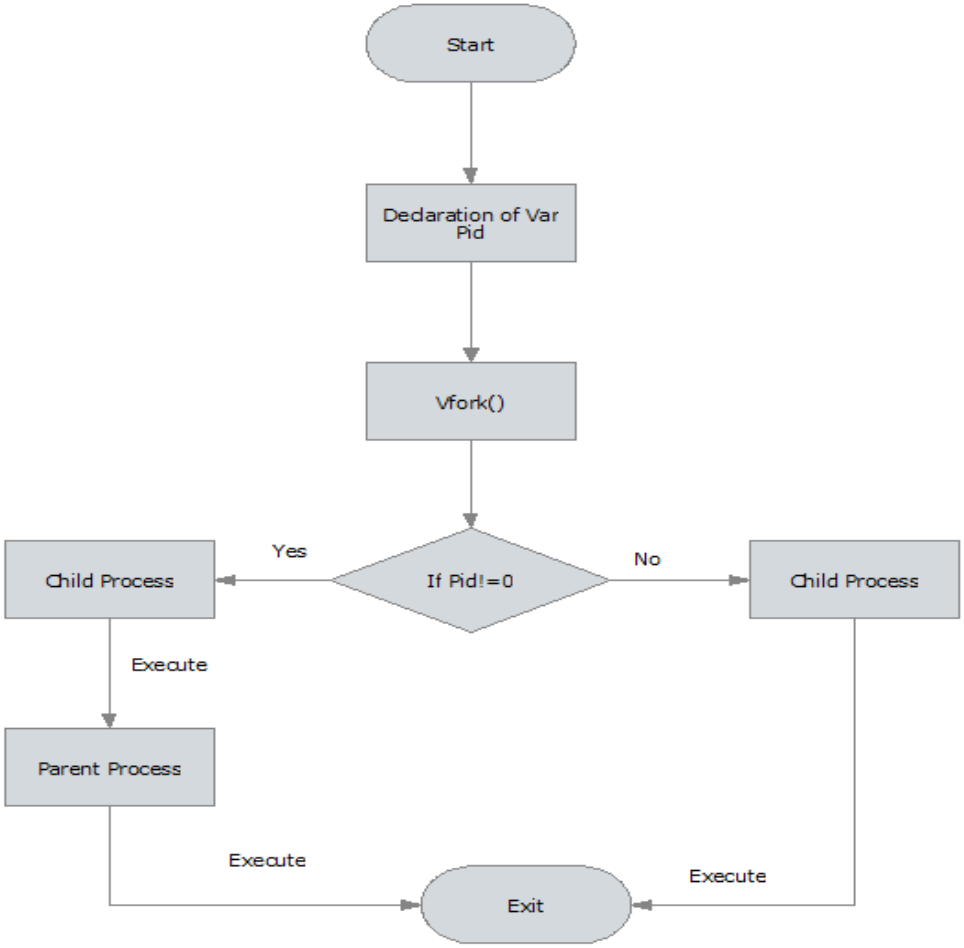

Table 1.3 vfork

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

Table 1.3 Data Dictonary

Program 1 (fork()):

```
#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/1/1c$ gcc fork1.c
sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1c$ ./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
```

Program 2 (vfork()):

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

void main()
{
        pid_t p = vfork();
        if(p < 0)
        {
              printf("vfork() failed\n");
        }
        else if(p == 0)
        {
               printf("In Child Process Started with pid = %d\n",getpid());
               for(int i=1;i<=5;i++)
               {
                      printf("In Child : %d\n",i);
               }
               printf("Child Finished\n");
               exit(0);
        }
        else
        {
               printf("Parent Process Starded with pid = %d\n",getpid());
              for(int i=1;i<=5;i++)
              {
                    printf("In Parent : %d\n",i);
              }
              printf("Parent Finished\n");
        }
}
```

Output:

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1c$ gcc vfork1.c

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

In Child Process Started with pid = 365692

In Child : 1

In Child : 2

In Child : 3

In Child : 4

In Child : 5

Child Finished

Parent Process Starded with pid = 365691

In Parent : 1

In Parent : 2

In Parent : 3

In Parent : 4

In Parent : 5

Parent Finished

Conclusion:

1. fork() and vfork() system calls have some differences which allows different type of execution of child processes.
2. learn wait and waitpid system calls.

References:

[1] www.tutorialspoint.com/unix_system_calls/