Lab : Understanding and Using fork()

Process Creation

Goal: In this lab you will write programs using the fork() system call.

The TAs will discuss the review problems and the programming exercise. To finish, you will have to write three programs and demonstrate them to a TA.

1 Understanding fork() using example programs

A call to fork() might fail with a return value of -1. For each problem below, assume that fork() succeeds at every call.

1. Enumerate all possible outputs of the following program.

```
int main() {
    int x = 3;
    if (fork() != 0)
       printf("x=%d\n", ++x);
       printf("x=%d\n", --x);
       exit(0);
       }
```

2. How many "hello" output lines does this program print?

```
void doit() {
     if (fork() == 0) {
         fork();
         printf("hello\n");
         exit(0);
         }
         return;
         }

int main() {
    doit();
    printf("hello\n");
```

```
    exit(0);
    }
```

3. How many "hello" output lines does this program print?

```
void doit() {
     if (fork() == 0) {
         fork();
         printf("hello\n");
         return;
         }
     return;
     }

int main() {
    doit();
    printf("hello\n");
    exit(0);
    }
```

4. Consider the following program.

```
int main() {
    int i, nchildren = 0;
    pid_t pid;

    for (i = 1; i <= 2; i++) {
        pid = fork();
        if (pid > 0)
            nchildren++;
            }

    printf("I had %d children\n", nchildren);
    return nchildren;
    }
```

   (a) How many new processes (in addition to the initial process) are created by this code? Draw a process graph that illustrates the processes at run-time. Hint: It helps to write down the values of n children for each process.

   (b) I executed this program once and generated the following output.

```
I had 1 children
I had 0 children
I had 2 children
I had 1 children
```

Adding the numbers suggests that 4 child processes were created. Explain why this differs from your answer for part (a).

5. Consider the following code snippet.

```
for (i = 0; fork(); i++) {
     if (i == 4) break;
     printf("PID: %d, i= %d\n", getpid(), i);
}
```

Assume that initially the process ID executing this code is 2394, and the PID always increases sequentially by 1. For example, the first time this process calls fork() the PID of the child is 2395. You can also assume no other processes in the system are calling fork() after this program starts to execute.

What are the possible outputs of this program?

6. Now consider the following code snippet.

```
for (i = 0; i < 4; i++) {
     if (fork()) break;
     printf("PID: %d, i = %d\n", getpid(), i);
}
```

Under the same assumption as above, what are the possible outputs of this program?

2 Programming Assignment

Below is a program to generate the ' family tree using exactly SIX fork() calls. The parent child links are not stored by the program but are part of the process hierarchy. The variable pid stores the process id of the parent process. Each of the six processes created by the program enters an infinite loop, and is later terminated by a SIGKILL signal. When a key is pressed, the program invokes pstree using the system function. Because a call to system() will fork a process, the output of pstree will show an extra process that runs pstree itself.

Save the source code in a file fork6.c, compile and run the program.

```
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
```

```
main()
{
    pid_t pid;
    char cmd[20];

    pid = getpid();

    if (fork() == 0)
    {
        if (fork() == 0)
           while(1);
        while(1);
    }

    if (fork() == 0)
    {
        if (fork() == 0)
           while(1);
        if (fork() == 0)
           while(1);
        while(1);
    }
    if (fork() == 0)
        while(1);

    getc(stdin);
    sprintf(cmd, "pstree -p %d\n", pid);
    system(cmd);
    kill(pid, SIGKILL);
}
```

- The above process hierarchy can also be generated with exactly five and four fork() calls.

Your task is to code the above possible scenario. Write a separate program for each scenario. Call your programs fork5.c and fork4.c respectively.