CSC565: Operating Systems

Spring 2017

Assignment 1

Due Date: February 23, 2017 -- Due Time: 23:59

Problem 1: fork() (written) [25 points]

Consider the following C code:

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

int main( )
{int i;
   fork();
   for (i=1; i < 3; i++)
    fork();
   fork();
   return 0;
}
```

A. Including the initial parent process, how many processes are created by this program?
B. To visualize your answer in A, draw the tree of processes created by this program.

Problem 2: fork (C Programming) [35 points]

I suggest that you write the program in two stages. Hand in only the second stage version (**program and output**).

Stage 1: Your program should use fork() to create a new child process.

The parent (the original process) should output a line:

> In parent: Child PID = 123, my PID = 456, my parent PID = 789.

but with the actual process IDs, not the numbers given above.

The child (the new process) should output a line:

> In child: My PID = 123, my parent PID = 456.

again, with the actual process IDs.

In the parent process, the value returned by fork() is the child PID. For either parent or child, you may use the library functions

getpid()
getppid()

for the other process IDs.

Stage 2: Modify the child process so that, after printing the indicated line, it uses execv() to replace the running program with

ps

That should list all of your processes at that time.

Additionally, modify the ~~child~~ process, so that after it has printed its line of output, it then waits for the child process to complete, using wait(). The parent should then print:

Child process has finished.

The exact order in which the lines of output appear is not important, and may vary from run to run (as will the PID numbers).

Problem 3: Pipes (C Programming) [40 points]

Design a C program using ordinary pipes in which one process sends a string message to a second process, and the second process reverses the case of each character in the message and sends it back to the first process. For example, if the first process sends the message "Hello", then the second process sends back the message "hELLO". This will require two pipes. You are required to use UNIX ordinary pipes for this problem.