This programming assignment is intended (part 1) to help the user understand that, from the kernel's view point, the shell is simply viewed as an application program that uses system calls to spawn and to terminate other user programs, and (part 2) to familiarize the user with Linux programming using several system calls such as fork(), execlp(), wait(), pipe(), dup2(), and close(). Additionally, to become familiar with the already implemented ThreadOS operating system simulator, especially in part 2 of this assignment.

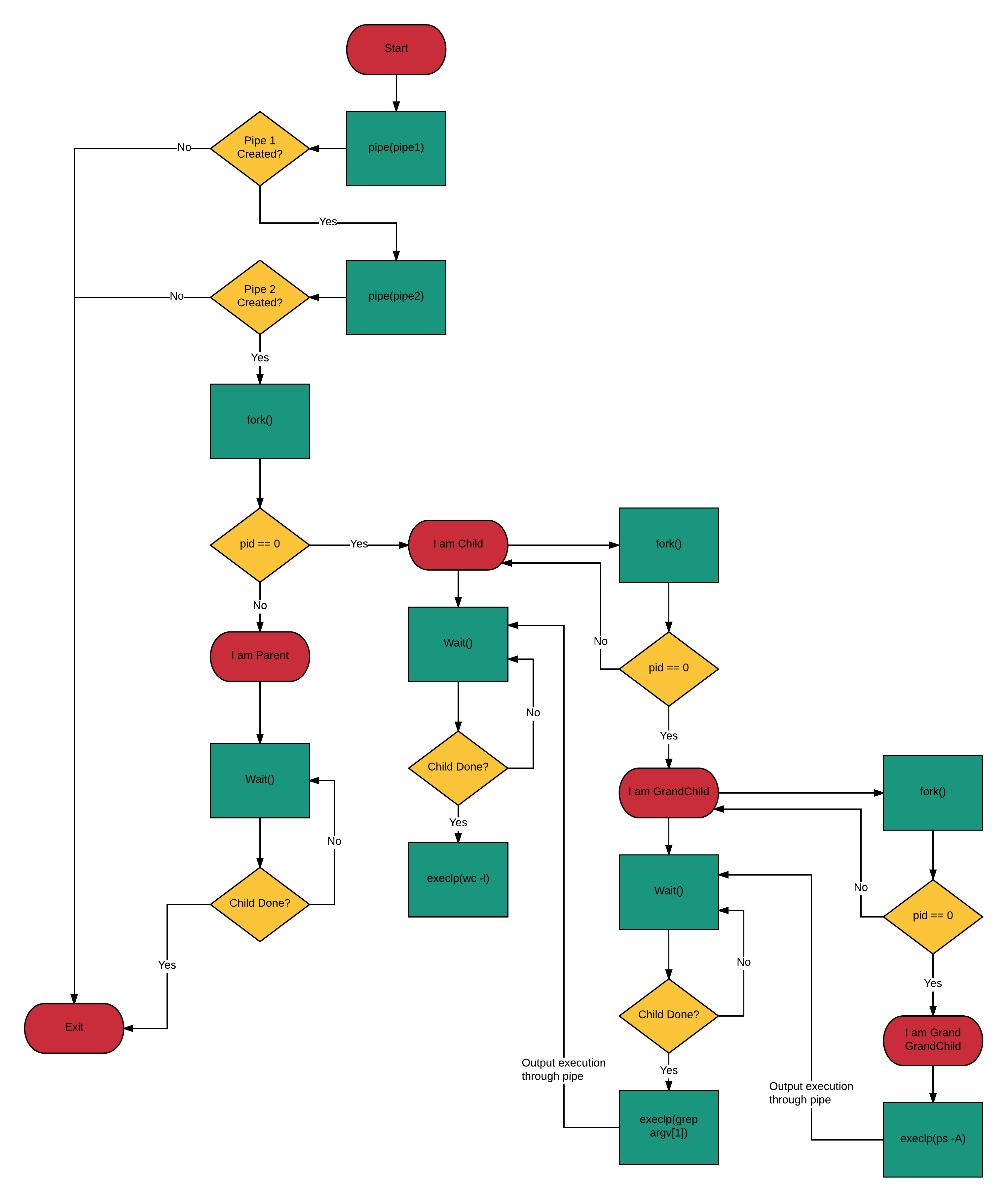
|  |
| --- |
|  |
| Program 1: System Calls and Shell |
|  |
| CSS 430 – Operating Systems |

**Nenad Bulicic**

## Part 1 – System Calls

Part 1 of the programming assignment required to code a C++ file named processes.cpp which would emulate the following Linux shell command line:  
$ ps -A | grep argv[1] | wc –l  
The execution is done left, being first, to right, and each ‘|’ delimited command is a separate process. The algorithm is closely based on the specification table on the assignment page.  
The way the algorithm works is it first creates two separate pipes for child-parent communication. Then the program (parent) creates a new child process by forking, this child process spawns its own new child process (or grandchild), and lastly this child (grandchild) process spawns its own new child process (grand grandchild). Each parent waits for its child to finish execution before proceeding with its. The grand grandchild process establishes communication to parent (grandchild) by duplicating pipe 2’s write side, closes pipe 1’s read and write, and pipe 2’s read side, and executes “ps –A” command and sends output through the pipe. The grandchild process, establishes communication to parent (child) by duplicating pipe 1’s write side, establishes communication to child by duplicating pipe 2’ read side and reads, closes pipe 2’s write and pipe 1’s read side, and executes “grep argv[1]” command and sends output through the pipe. The child process establishes communication to child by duplicating pipe 1’s read side and reads, closes pipe 2’s read and write, and pipe 1’s write side, and executes “wc -l” command. The complete program execution flowchart can be seen in in figure 1.

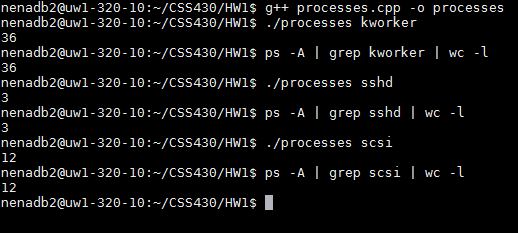
Figure 1: Processes.cpp execution flowchart



To compile and execute the processes.cpp program use the following syntax:  
g++ processes.cpp –o processes  
./processes argv[1]  
where argv[1] is substituted by kworker, sshd, or scsi.

The program output with kworker, sshd, and scsi can be found in figure 2.

Figure 2: Processes.cpp vs. actual system outputs



## Part 2 – Shell and ThreadOS

Part 2 of the programming assignment required to code a Java file named Shell.java which would be invoked from the ThreadOS Loader and emulate a shell command interpreter.  
The way the algorithm works is it executes the run() function and enters a while(true) loop which is used to take in user commands, parse them and pass along to executeCommands() function. But before the command strings are passed to execute they are checked if they match “exit”, at which point the Shell program calls SysLib.exit() and shuts down. If the user string does not match “exit” it is then parsed with the ‘;’ delimiter. These delimited strings, if not empty, are then passed to executeCommands(), which first performs the same kind of string parsing but with the ‘&’ delimiter. Then, again, if the string is not empty the command is executed by passing it to SysLib.exec() function. This function returns -1 on failure or a positive integer on successful execution, which is then inserted into a hashset. Once the input string has been traversed the function then waits for all child processes to finish by calling SysLib.join() and removing the successfully returned child ID from the hashset.  
This processes is then repeated until the entire initial user string has been traversed and parsed.

To test the program:  
- Download a clean version of the ThreadOS, and hopefully you have full permission for all files, so just and add the Shell.java file into the downloaded folder and compile with syntax: javac Shell.java. Otherwise the compiling might have to be done somewhere else and both Shell.java and Shell.class copied into the ThreadOS folder.  
- Run ThreadOS with the syntax: java Boot, and once it boots and displays “-->”, run syntax: l Shell to load the Shell program. This will load the Shell program and Shell[1]% will appear if everything went correctly.  
- Run as many concurrent and sequential programs, such as PingPong, as needed using ‘&’ and ‘;’.

Figure 3, 4, and 5 represent the full execution output of the required tests and some extra. For a continuous test execution output in text format please refer to the included Shell\_Output.txt included in the .tar bundle.

Figure 3: Shell.java test execution output

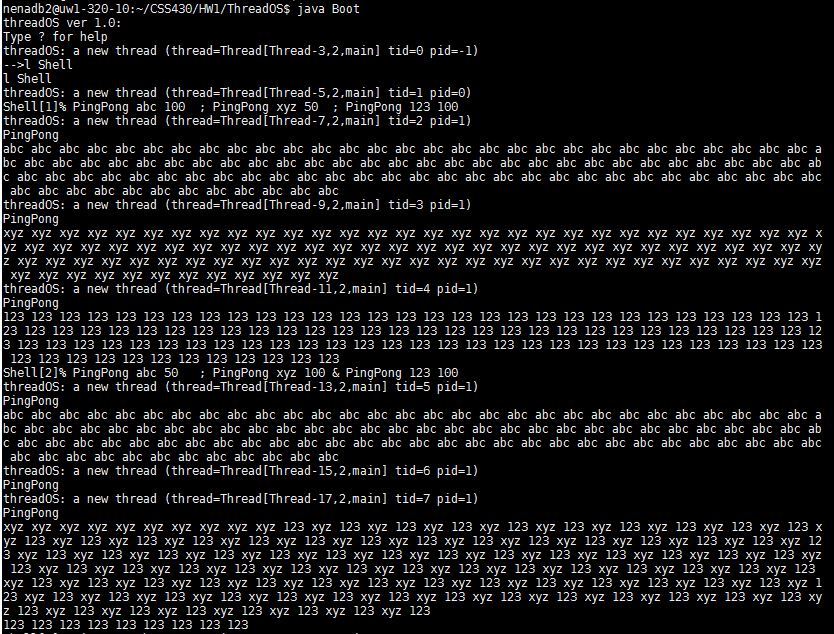


Figure 4: Shell.java test execution output



Figure 5: Shell.java test execution output

