Chandler Ford | January 23rd, 2019

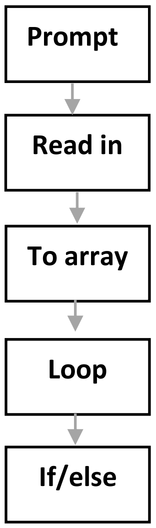
CSS 430 Operating Systems | Program 1 Report

Testing Shell.java

I connected to the Linux Lab with PuTTY to test Shell.java. ThreadOS is needed to run this program. The file must be placed inside of the folder that has Boot.java, Disk.java, Kernel.java, etc. After using javac Shell.java to compile the file, run java Boot and type l Shell. From there it is possible to run the test cases. For example, typing PingPong abc 100 ; PingPong xyz 50 ; PingPong 123 100 after the Shell[1]% will yield output. If it freezes the user may have to push Ctrl + c and go through the process again.

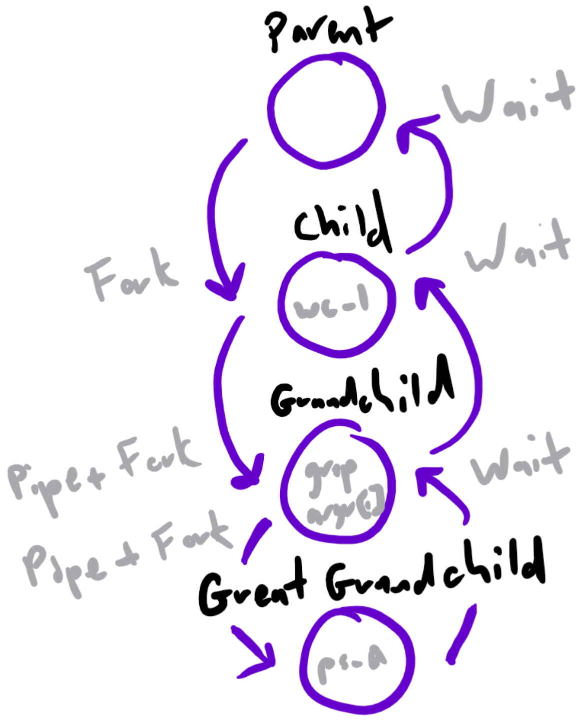
Algorithm Discussion: Shell.java

This program uses a constructor to initialize the x counter variable used in the cout statement. The rest of the program is mainly found in the run() function. This function begins by printing out the Shell[x]% prompt before reading in the contents of the buffer. This is then split into a string array via a ThreadOS system call. An if/else if structure inside of a loop through the array controls the sequential/concurrent thread creations with the help of a sub array that takes in commands before the delimiters. If the sequential, the program waits until the returned ID from the join system call matches the ID of the newly executed thread. See flowchart below.



Algorithm Discussion: Processes.cpp

This algorithm uses the creation of multiple processes to demonstrate the same behavior as *ps -A | grep argv[1] | wc -l*; this command views all processes running on the system, processes the text line by line, and counts the number of lines. The program creates multiple processes that send input/output around through the use of forking and pipes. A series of forks followed by if statements create the child, grandchild, and great-grandchild. Pipes are created between the child and grandchild as well as between the grandchild and great grandchild. Once the center of the structure (the great-grandchild) is reached, the ps -A command is run with the help of execlp. All of the processes above this wait as the algorithm starts to traverse through the else statements, which represent the parent of the current process. The commands grep argv[1] and wc -l are reached in this fashion. The system calls dup2 and close are used to pipe input/output around and close pipes. See diagram below.


Parent
Wait
Fork
Child
wc -l
Wait
Grandchild
Pipe + Fork
grep argv[1]
Wait
Pipe + Fork
Great Grandchild
ps -A