jbobotek_lab1report

Part 1 - process.cpp algorithm:

The algorithm used to execute the process chain was fairly simple, and pretty much exactly following the table given in the project description. After creating/forking each process to the smallest/grandest child (child process relationship necessary for direct pipe communication), processes had their process image changed (using execlp) to the command required to complete their third of the command chain. Note that because the grandest child is run first, it must process the first command before it can pipe the result to its parent and so on until it reaches the absolute top. Before each exec, the pipes created by the original parent are closed and dup2'ed until the child's process image can only go in the intended direction/pipe.

Part 2 - Shell.java algorithm:

The algorithm used to impliment Shell.java was sort of twofold. I knew that the shell would need to wait for user commands, and would have to use SysLib's cin and cout to do all of this, and it would obviously need to be able to accept unspecified multiple lines of commands, so a loop would be needed, and would only end (and end Shell) when the user told it to exit. As far as processing the commands on one line, I am familiar with processing delimited commands in Java a bit, so I took the clever step of realizing that the ';' delimiter was the more absolute command of the two, meaning that one to many commands could be processed after split by ';'.

Simply stated - Shell reads in one line at a time, splits commands by semicolon, passes this smaller String to another method to split the line again by the '&' delimiter and process the command(s) that result. Of course, to execute these commands it must be put in the format that SysLib likes, and SysLib.stringToArgs was used for this purpose. Threads were then waited/joined before returning to the main Shell loop.

Part 3 - How to test Shell.java:

1. Obtain the ThreadOS project and add Shell.java to the project.
2. Run ThreadOS (Boot) and wait for ThreadOS to get ready to accept commands.
3. type 'l Shell' into the console and wait for shell[1]% to appear.
4. enter series and permutations of ;, &, and PingPong (insert any word you wish) (a number) - to test.