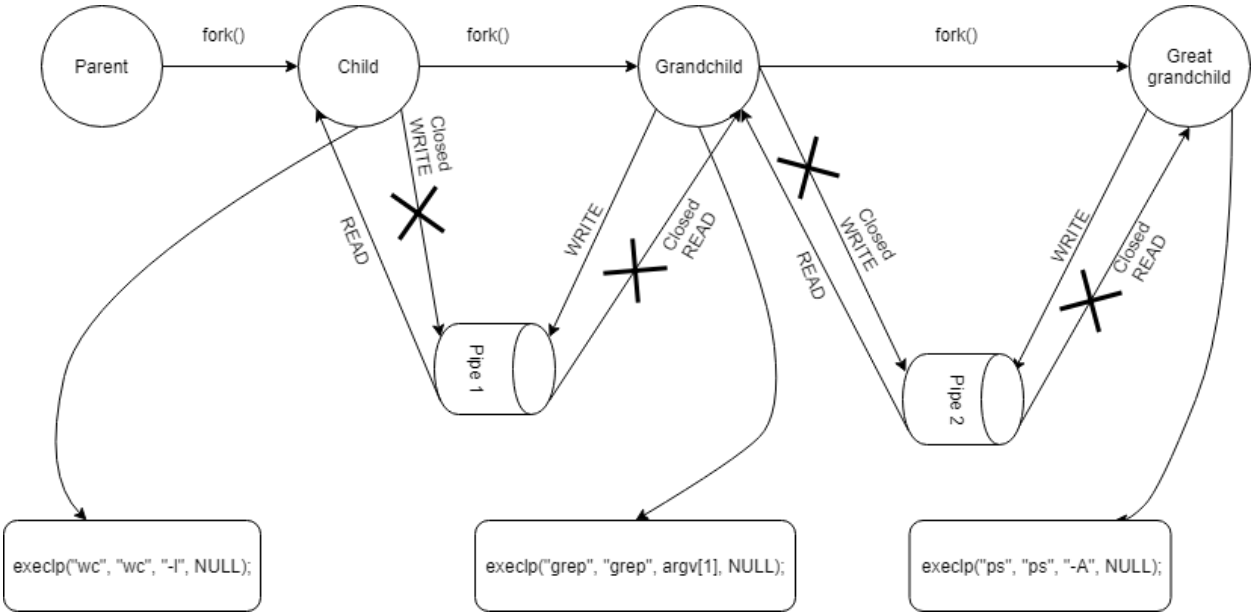
CSS 430 Program 1 Report

Process.cpp



The parent will fork, creating a child process. After that, the parent invoke wait(&status) to wait for all child status to finish. Closed READ & WRITE for pipe 1 and pipe 2 since parent process will not be using them.

The child will fork, creating a grandchild process. After that, the child use dup2(pipe1[READ], 0) to duplicate READ end of pipe 1 to the stdout of the child process. Closed WRITE for pipe 1 and closed READ & WRITE for pipe 2. The child process executes execlp("wc", "wc", "-l", NULL), then invoke wait(&status) to wait for grandchild and greatgrandchild process to finish.

The grandchild will fork, creating a greatgrandchild process. After that, the grandchild uses dup2(pipe1[WRITE], 1) to duplicate WRITE end of pipe 1 to the stdin of the grandchild process, and uses dup2(pipe2[READ], 0) to duplicate READ end of pipe 2 to the stdout of the grandchild process. Closed READ for pipe 1 and closed WRITE for pipe 2. The grandchild process execlp("grep", "grep", argv[1], NULL).

The greatgrandchild does not fork. The greatgrandchild uses dup2(pipe2[WRITE], 1) to duplicate WRITE end of pipe 2 to the stdout of the greatgrandchild process. Closed READ for pipe 2 and closed READ & WRITE for pipe 1. The greatgrandchild process executes execlp("ps", "ps", "-A", NULL).

Shell.java

Testing:

1) Go to the directory that has ThreadOS by using command “cd” in Unix.
2) Add Shell.java to the ThreadOS directory.
3) Type javac Shell.java to compile the Shell.java
4) Type java Boot to boot the threadOS
5) Type l Shell. That is (a lower-case L) Shell.
6) Utilized some test cases below
    a. PingPong abc 100 ; PingPong xyz 50 ; PingPong 123 100
    b. PingPong abc 50 ; PingPong xyz 100 & PingPong 123 100
    c. PingPong abc 100 & PingPong xyz 100 ; PingPong 123 50
    d. PingPong abc 50 & PingPong xyz 50 & PingPong 123 100
7) Enter exit as the first argument (Ex: exit & PingPong abc 50, …) to exit the shell.
8) Enter q to end the thredOS and return to the Unix command prompt.

Code (please refer to the Code comments for further details):

Method run():

Method prompt the user to enter a line of input.

Method uses SysLib.cin to accept input, and SysLib.cout to print to the console.

Method uses String Buffer to store the input.

Method convers the String Buffer into string array.

Checks for empty argument. If so, return to the start of the for loop and prompt for new command lines.

Checks if the first element of the string array is “exit”. If so, break from the for loop, sync then exit the shell using SysLib.sync() and SysLib.exit() respectively.

Then invoke method execute() with a parameter as the string obtained from converting the String Buffer.

After invoked execute() method, increase the thread counter and start a new thread. (Ex: shell[1] -> shell[2]). Run until exit condition is invoked.

Method execute():

This method works as the execution for arguments which will run sequentially. These arguments are separate with a delimiter “;”. Accept a String input as parameter.

Use the split() method on the String input to separate based on the delimiter ";" into string array. Ex: PingPong abc 50 & PingPong xyz 50 ; PingPong 123 100 will be split into array[0] = "PingPong abc 50 & PingPong xyz 50", and array[1] = "PingPong 123 100".

If array[index] is empty, then iterate to the next array[index++].

Check for any delimiter "&" in each array element. Ex: checkConcurrent(array[0]) will return true.

If it is not true then execute the command as a sequential command.

Invoke SysLib.exec(SysLib.stringToArgs(sequential)). Get the processId (pid).

If the pid is -1 meaning the program failed to create a child thread based on the given string to string array.

Otherwise, the method will run in a while loop until the child thread created above is complete. This is the part that makes this sequential. A created thread must finish before the next iteration, next thread creating, or reading the next command line.

Method checkConcurrent():

This method is called by the execute() method to check if there is concurrent delimiter "&" within the array element. Accept a String input as parameter. Use the split() method on the String input to separate based on the delimiter "&" into string array. Ex: arrayfromExecuteMethod[0] = "PingPong abc 50 & PingPong xyz 50" will be split into arrayinCheckConcurrentMethod[0] = "PingPong abc 50", and arrayinCheckConcurrentMethod[0] = "PingPong xyz 50", then the program returns true later on. arrayfromExecuteMethod[1] = "PingPong abc 123 100" however will return false since there is no delimiter "&".

If input contains "&" then create a counter that will be used to keep track of created thread.

Similar to how the execute() method use split and iterate through the string array. This method however, split based on the delimiter "&".

If array[index] is empty, then iterate to the next array[index++].

Invoke SysLib.exec(SysLib.stringToArgs(sequential)), which returns -1 if failed

If value is -1 then print out that the method failed to create a child thread.

Else increase the count by 1, which keep track of how many child thread is created and running.

Create a for loop which will iterate through the counter, waiting for all child thread to finish before exiting the method, returning true that there is a concurrent delimiter “&”.

Overall

My Shell class handles delimiter by first separate them based on the sequential delimiter “;” and put them into String array. Then iterate through the array and check for the concurrent delimiter “&”. If found any concurrent, it will execute the concurrent before returning true to the sequential method. The sequential method will only continue to the next iteration or create a sequential child thread after receiving a Boolean from the concurrent method. For the sequential method, the important part is that the method will wait until the created sequential child thread is finished. For the concurrent method, create as many child threads as required by the argument provided, then wait for all child thread to terminate via a for loop iterating through the number of concurrent child thread created. The assumption for this class is that the user will input exit as the first argument in the command line. Putting it in any other order will result in a failed to execute command error. Another assumption is that there will be no orphaned delimiter “&” or “;”, meaning they will not appear at the last character of the command line input.