Leo Le

Professor Robert Dimpsey

CSS 430

April 9, 2018

Program 1

**Processes.cpp algorithm explanation:**

The processes.cpp is a program that will do 3 commands: ps -A, grep, and wc -l at the same time as well as piping their result together in order to get the desired output. The program will first look for an appropriate argument from user input. If not, the program will show an invalid syntax to notify the user. When the program takes the argument it required, the bash will create 2 pipes in order to connect the data from each process it is about to create. The program will check for invalid pipe when creating them. After that, the bash process will do a system call fork() to create a child process for “wc -l” command. The “wc -l” child process will make another fork() call to create a grand-child process for “grep”. Lastly, the “grep” grand-child process will call fork() for a great grand-child process that will process “ps -A”. All three processes: child, grand-child and great grand-child will process each command concurrently. The data flow will move from “ps -A” process to pipe 1; from pipe 1 to “grep” process and write data to pipe 2; process for “wc -l” will read the data from pipe 2 in and finish the process. The bash process will wait until all of its 3 child processes to finish their job and collect all the data which has been processed from each child.

**processes.cpp output:**

phucle@uw1-320-01:~/430/p1/Program\_1$ ps -A | grep kworker | wc -l

35

phucle@uw1-320-01:~/430/p1/Program\_1$ ./processes kworker

35

phucle@uw1-320-01:~/430/p1/Program\_1$ ps -A | grep sshd | wc -l

3

phucle@uw1-320-01:~/430/p1/Program\_1$ ./processes sshd

3

phucle@uw1-320-01:~/430/p1/Program\_1$ ps -A | grep scsi | wc -l

12

phucle@uw1-320-01:~/430/p1/Program\_1$ ./processes scsi

12

**Shell.java algorithm explanation:**

The Shell.java program is created to take in and run different program in the ThreadOS given. This shell is an imitation of the bash shell in terminal. The Shell will firstly prompt for the command from the user after it greets the user. The user will have to put in the program/command that they want the Shell to run for them. For example, instead of using: l HelloWorld to run the HelloWorld program/command, all we need to do is: l Shell in order to run the Shell program and start typing in the name of the program/command we want to run. The output below is the perfect example for this situation.

When the Shell get the arguments from the user, it will start processing the command made by the user. The argument comes in as a string buffer and will go through the command handler. If the argument is different from “exit” and “q”, the program will continue the process. If not, the Shell program will be terminated as the user wants to exit Shell program. If there are nothing in the argument, the shell will count it as a command and start increment the shell count without doing anything. If the argument is an actual program/command and more than 1, it will be split into 3 different conditions.

1. The argument contains both “;” and “&” symbol: The program will use regex string split to split the argument into small chunks of string with the delimiter “;” or “&” at the end of string. After that, depending on what the last character it is in that string(“;” or “&”), the program will execute the command concurrently or sequentially. If the string itself has no character that is “;” or “&”, the program will do a sequential execution on it.
2. The argument only has “;” symbol: The program will use regex string to split the delimiter “;” and execute each argument sequentially.
3. The argument only has “&” symbol: The program will use regex string to split the delimiter “&” and execute each argument concurrently.

If there is only 1 argument, the program will execute it sequentially. The sequential execution is an execution that will hold the program doing any other command/program until the current is done. The concurrent execution is an execution that will be executed along with other program/command and do not have to wait for anything. When the user wants to quit, they will have to type in “exit” or “q”. The Shell program will then sync all on-memory data back into disk and exit the program.

**Shell.java output:**

phucle@uw1-320-01:~/430/p1/Program\_1/ThreadOS$ java Boot

threadOS ver 1.0:

Type ? for help

threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)

-->l Shell

l Shell

Shell initiated!

To terminate the Shell, type (q) or (exit).

threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)

Shell[1]% PingPong abc 50 ; PingPong xyz 100 & PingPong 123 100

threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)

abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc

threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=1)

threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=1)

xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123 xyz 123

123 123 123 123 123 123 123 123 123

Shell[2]% PingPong abc 100 & PingPong xyz 100 ; PingPong 123 50

threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=1)

threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=1)

abc abc abc abc abc abc abc abc abc abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz

xyz xyz xyz xyz xyz xyz xyz xyz xyz

threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=1)

123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123

Shell[3]% PingPong abc 50 & PingPong xyz 50 & PingPong 123 100

threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=1)

threadOS: a new thread (thread=Thread[Thread-21,2,main] tid=9 pid=1)

threadOS: a new thread (thread=Thread[Thread-23,2,main] tid=10 pid=1)

Shell[4]% abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc 123 xyz abc xyz abc 123 xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz abc xyz 123 abc xyz

xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123 xyz xyz 123

123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123

Shell[5]% PingPong abc 100 ; PingPong xyz 50 ; PingPong 123 100

threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=1)

abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc abc

threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=1)

xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz

threadOS: a new thread (thread=Thread[Thread-29,2,main] tid=13 pid=1)

123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123 123

Shell[6]% q

Superblock synchronized

-->q

q

Superblock synchronized

How to Compile processes.cpp:

* Run “g++ -std=c++14 processes.cpp” on terminal in order to compile the program.
* Do “./processes <name for argument>” the name inside of the bracket is for the process you want to see.

How to run Shell.java in ThreadOS:

* Boot threadOS by doing: “java Boot”.
* Use: “l Shell” to run the shell program. (If Shell.java is not compiled yet, do “javac Shell.java”)
* The Shell will prompt you for the command you want to use. From here, you can type in the test case shown in the program 1 documentation given by the professor. (The output above is a great explanation for this)