Colton Sellers

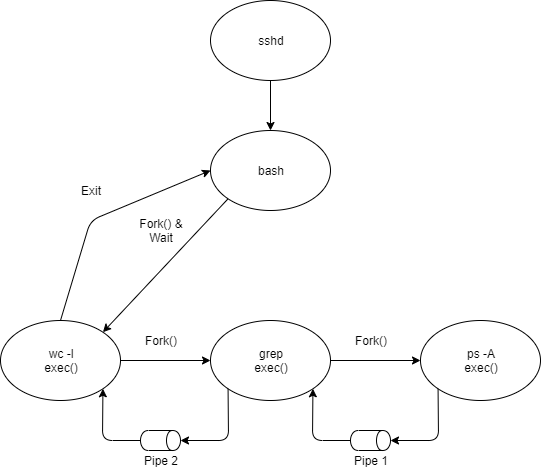
CSS 430 B

Assignment 1

# Part 1: Processes.cpp

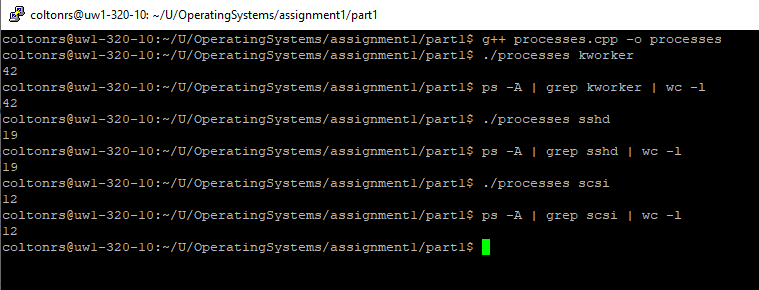
Processes.cpp is meant to demonstrate a pipe system of commands, passing the output of one command to the next through pipes. In order to this it is necessary to break down the commands and assign them the proper pipes to either read or write to. I used a switch statement to fork the process three times to create a child, grandchild and great grandchild as shown in the flowchart below.

*1.1 Flowchart of processes.cpp*



As you can see the output of ps is passed into Pipe 1 which is then read by grep as an input, then grep executes and writes it output to Pipe 2, which is then read my wc and its output written back to the bash shell which initially called this program.

*1.2 Testing of processes.cpp*



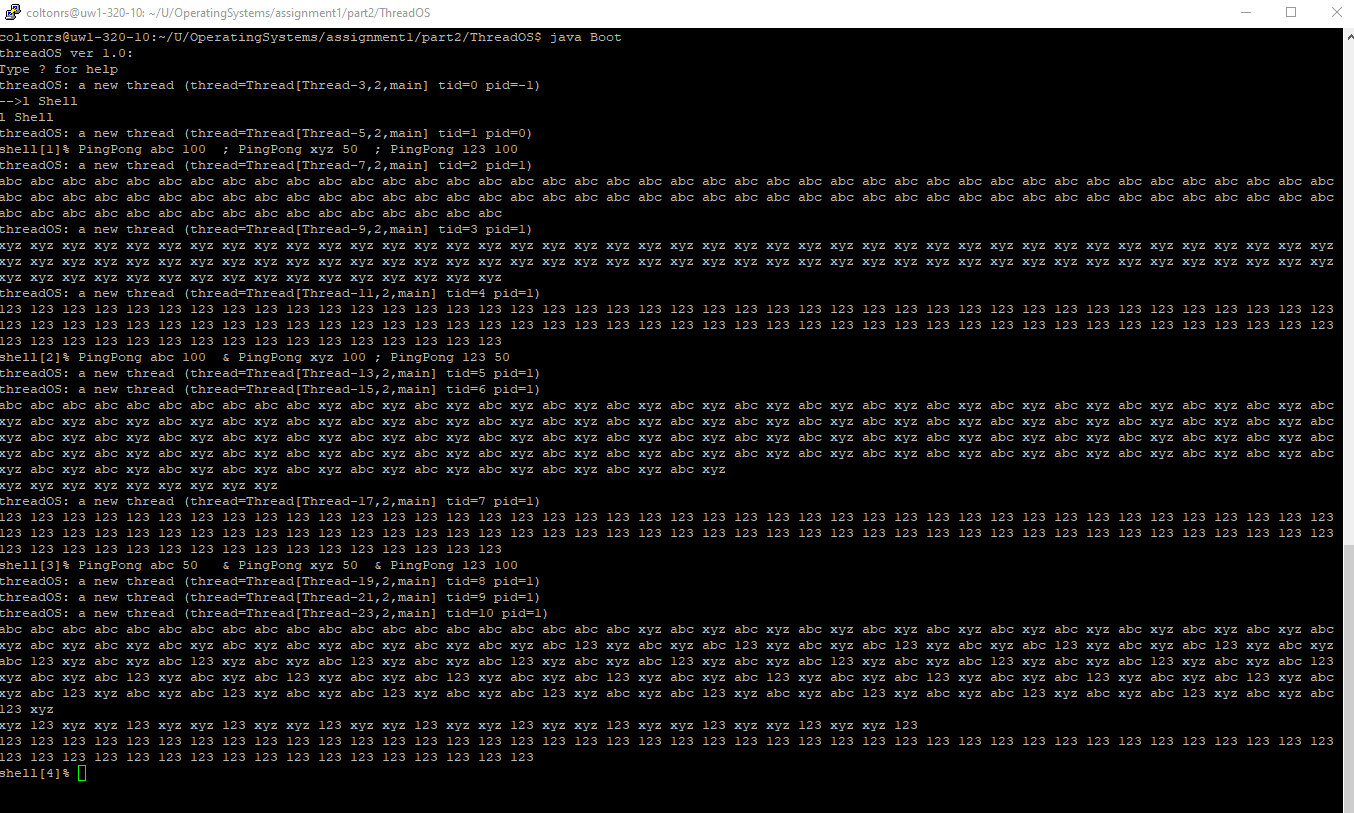
Testing for this process was done by comparing the output of my C++ program against the exact same Linux command. In figure 1.2 you can see the processes program and Linux commands provide the same output thus proving its ability to properly execute the function from C++.

# Part 2: Shell.java

Shell.java is designed to give ThreadOS a shell terminal to execute programs by utilizing ThreadOS syslib functions. It is capable of running multiple processes off of one command line. This can be done by breaking commands up by either “;” or “&”. When using “&” after a command the shell will execute the command and continue processing the rest of the arguments. When using “;” after a command the shell will wait until the command is finished before moving on to the rest of the arguments.

In implementation I used a while loop with a conditional Boolean that is only altered if the command issued is “exit". This way the shell continues running and taking commands until the condition is changed. In this while loop the shell prints out the current command number and waits for a input using syslib.cin(), once the line has been entered it ensures the string contains some and proceeds into processInput(). This function takes in the string input and uses another syslib function stringToArgs() to convert it to an array. Once separated processInput() iterates through each argument looking for the end of a command, meaning either it reaches the end of the arguments or it finds a “;” or “&” and handles each appropriately. This command is then extracted by buildCommand() and executed by syslib.exec().

*2.1 Test cases of Shell.java*



I tested Shell.java by using the test cases given in the assignment, calling other given test programs in various commands, and checking its ability to exit back to the kernel successfully. In figure 2.1 it shows the shells ability to run multiple threads in one entry with “&” as well as properly waiting for the thread to finish when defined with “;”.

# Code Appendix

## processes.cpp

|  |
| --- |
| /\*  Colton Sellers  CSS 430 Assignment 1  Part 1: processes.cpp  1/20/2020  A program meant to demonstrate system calls and piping through C++  \*/  #include <iostream> //for cout, endl  #include <unistd.h> //for fork, pipe  #include <stdlib.h> //for exit  #include <sys/wait.h> //for wait  using namespace std;  int main(int argc, char\* argv[]) {  enum {Read, Write}; //Give file descriptors values, Read=0, Write=1  int filedes1[2], filedes2[2]; //Pipe file descriptors  if(pipe(filedes1) < 0 || pipe(filedes2) < 0){ //create the pipes  cerr << "Pipe Error" << endl;  }  switch(fork()){  case 0: //this is the child -> wc-l  close(filedes2[Write]);  close(filedes1[Write]);  //read from pipe 2  dup2(filedes2[Read],0);  close(filedes2[Read]);  execlp("wc", "wc", "-l", NULL);  break;    default:{ //parent process  switch(fork()){  case 0: //this is the grandchild -> grep  close(filedes2[Read]);  close(filedes1[Write]);  //read from pipe 1 write to pipe 2  dup2(filedes1[Read],0);  dup2(filedes2[Write],1);  close(filedes1[Read]);  close(filedes2[Write]);  execlp("grep", "grep", argv[1], NULL);  break;    default: //parent process  switch(fork()){  case 0: //this is the great grandchild -> ps -A  close(filedes1[Read]);  close(filedes2[Write]);  //write to pipe 1  dup2(filedes1[Write],1);  close(filedes1[Write]);  execlp("ps", "ps", "-A", NULL);  break;    default: //parent process  wait(NULL);  exit(EXIT\_SUCCESS);  break;  }  break;  }  break;  }  }  } |

## Shell.java

|  |
| --- |
| /\*  Colton Sellers  CSS 430 Assignment 1  Part 2: Shell.java  1/23/2020  A shell program to used by ThreadOS  \*/  class Shell extends Thread{  public void run(){  boolean running = true;  int shellCmdCount=1;    //Start the shell loop  while(running){  SysLib.cout("shell[" + shellCmdCount + "]% ");  StringBuffer buffer = new StringBuffer();  SysLib.cin(buffer);  String input = "";  input = buffer.toString();  if(input.length() != 0){  shellCmdCount++;  running = processInput(input);  }  }  SysLib.exit();  }  //Takes input and processes it. Returns true if shell is still live.  private boolean processInput(String input){    //Split up the input and track where the current command begins and ends  String[] argArray = SysLib.stringToArgs(input);  int cmdStart = 0;  int cmdEnd;  //Loop through the args to determine the next command  for (int argIndex = 0; argIndex < argArray.length; argIndex++) {  //If the current argument is the last argument in the array or signals execution then execute the command.  if (argIndex == argArray.length - 1 || argArray[argIndex].equals(";") || argArray[argIndex].equals("&")) {    //Determine where the end of the command is.  if(argArray[argIndex].equals(";") || argArray[argIndex].equals("&")){  cmdEnd = argIndex - 1;  } else {  cmdEnd = argIndex;  }  //Get us our command  String[] commandArray = buildCommand(argArray, cmdStart, cmdEnd);  //If its an exit we need to get out of the shell  if (commandArray[0].equals("exit")) {  return false;  }    //Execute and collect the thread ID.  int tid = SysLib.exec(commandArray);  //If the thread hasn't failed and the job isn't background wait for it to complete  if (tid != -1 && !argArray[argIndex].equals("&")){  while (SysLib.join() != tid);  }    //Update counter to the beginning of the next command  cmdStart = argIndex + 1;  }  }    //Continue to take inputs  return true;  }    //Builds the commands for execution from array. Returns the command in a string array  private String[] buildCommand(String[] argArray, int commandStart, int commandFinish) {  String[] command = new String[commandFinish - commandStart + 1];    for (int index = commandStart; index <= commandFinish; ++index) {  command[index - commandStart] = argArray[index];  }  return command;  }  } |