**Programming Assignment 2**

Instructions: Write a report describing your unique design choices, algorithms (e.g., implemented file input/output redirection, piping, etc.), and any implemented bonus features along with its implementation technique. The report must be a minimum of a page. We are not

looking for generic items that are common to the entire class as part of this PA

instructions, but we are looking for your insights and unique contributions.

# Design of Code

### Simple commands with arguments

In order to run the simple commands with arguments, I created a variable num\_flags to hold the number of arguments. In order to populate this variable, I used the tknr file alongside the commands file. Tknr parsed the input from the terminal while commands formatted each argument as a command. Another design choice I needed to make was to convert the command’s argos to const\_char arrays. This was necessary in order to pass commands into execvp. There is a multitude of ways to incorporate this however I decided to convert commands’ args to const\_char arr. Then, I incorporated a null byte at the end of each index and a null ptr for the end of all the arguments.

### Background processes

In order to run the background processes I created a check in the parent for background processes. If any one of the commands passed in were background processes I would add them to a vector that I defined as bg\_pid. Then, within my “infinite for loop” / while(true) loop i used waitpid() to kill zombie processes. if (waitpid(bg\_pid[k], 0, WNOHANG) != 0 I would print background process is done.

### Input/Output Redirection

In order to implement input redirection, I opened the file and made use of dup2 to copy over the values into the new file or read them. I also implemented some error handling for debugging using perror.

### Piping

I implemented piping by first creating two variables to store the stdin and the stdout. I populated the stdin with fd 0 and stdout with fd 1. I then piped and forked accordingly and created a pipe for each command. I also incorporated dup2 to carry over the values and made sure to close unused end of pipes when finished.

### Directory processing

In order to implement cd and other directory processing I implemented multiple if statements that would consider each command. For example if only cd is present I use getcwd and chdir to obtain the previous directory. If cd - is present, I would make a choice based on what is occupied within the previous directory and current directory. This would allow me to go back and forth between the directories. Storing these variables ahead of time and in predetermined parts of the code i was able to easily implement cd-. For cd path i used a getcwd that obtains the path the user incorporated and went to that location.

### User Prompt

To get date/time for user prompt i used a clock variable of type time\_t and used ctime to convert it to a string. For the username I used getenv. To make sure they all ended on the same line i set the last index of current time to ‘\0’ instead of new line. I also used the curr\_directory I previously initialized utilizing getpwd. Finally, I printed all of those newfound variables on one line with proper color formatting.