Doruk Benli
29182

PA2 Report

Introduction:

We are asked to implement a command line interface in this programming assignment, This consists of several steps such as parsing, implementing command executions, implementing redirections and implementing the background jobs. While handling this we are also asked to do this with several process and since writing to the console after such steps are not atomic, we are also asked to implement this in a proper concurrent manner.

Implementation:

In this implementation, first thing we have to do is reading the command.txt file and parsing it. For this we use filestream library's getline() function. My implementation calls a function for parsing the given string lines. We call this function at each iteration of the while loop which will iterate the amount of line in the command.txt file and this function which is parseCommands(), also takes an ofstream object which will be our parse.txt file.
For making the parsing simple we use CommandValues struct which has 2 string fields. This kind of acts like a hashmap but since c++'s map library sorts the outputs alphabetically, i have used this struct with vectors to mimic it. We first initialize a vector of CommandValues objects which has 5 objects, each object's fields are Command, input, option, redirection and background job. We first initialize the background job to the string "n" for making things simple, then also initialize redirection to the string "-". Then in the given line we read each word with a stringstream object. If our word is equal to given 5 commands in the PA2 file then we change the vector's objects value to the given word. This is done by a very simple if check, which checks the given commands. For the input and options fields, we have no way of distinguishing input fields from normal strings but we can use the "-" char in the beginning of option string, since each of them has to start with dash char as it is stated in the PA2 folder. So we check if the given word's first char is dash, then we make the corresponding value of our data vector the that word, which in this case is option. Then we check for redirection. This is also very simple since this can only happen with "<" and ">" chars so we make a simple check to see wether a redirection exists or not. In this implementation we do not store the filename that is being redirected to, but for the later parts since we may need it we also store it in a string which is passed as a reference to this function. We make use of regex operations to find a pattern in the given line which ends with .txt, since this is the format of redirection. Importing thing is this regex operation does not affect the parse.txt output, this is for the later parts of the implementation. Last thing is checking for the background job, this can also be done with a single if check which checks for if word == "&" then we will make the "n" to "y", if not it will stay like that. For the input field which is the one that is left, we implement it in else part because it can't be any other thing. We also write 10 dashes before and after each line. For the rest of the

implementation, I had no time to implement them.

Conclusion:

If piping and fork operations are implemented correctly with locks, then executing each command in each iteration of while loop would print the correct results of the corresponding command being executed and this would mimic the command line interface of the operating system.