CS307 2022 SPRING PA1 – COMMAND LINE INTERFACE REPORT

UĞUR ÖZTUNÇ 28176

After opening the ***commands.txt*** file and while loop that reads line by line has started, at the very beginning of every while loop, ***clearEscape(char*)*** function clears the string, where the whole line is stored, from escape sequences and blanks. ***clearEscape*** function starts from the last character of the string, and changes the character to '\0' if it is '\r' , '\n' or ' ' initially. A while loop continues doing this task until the last character of the string is anything but an escape sequence or a blank. This operation made easier to do some other operations on the line string.

i.e: `"grep description -i < output1.txt \r\n" → "grep description -i < output1.txt"`
`"ls -a " → "ls -a"`

An integer named "***iswait***" is decleared with initial value of '1' (wait), and an if condition determines to set iswait to '0' (do not wait) or not by checking whether the line ends with ampersand (&) or not. If so, ***iswait*** is set to '0' and the last 2 characters of the line string is changed to '\0', since they have no purpose from here on. All the things related with background status will be operated according to the value of ***iswait*** integer. These decisions were made according to ***iswait*** integer:

- printing 'y' or 'n' at command information part
- waiting youngest child process or do not wait

Following pieces of code handles the 'wait' command:

```
int rc1 = wait(NULL);
while(rc1 != -1) {
  rc1 = wait(NULL);
}
```

In this part it will wait a single time first and if the ***rc1*** value is equal to -1 it wont wait anymore. If ***rc1*** is any other value, then it will wait until ***rc1*** equals to -1, which is returned by wait function when there is no process terminated, no child left. By doing this, when current command is 'wait', program will wait for all child processes to be terminated.

Background jobs are managed by "***waitpid***" function which waits for a specific child process to be terminated, unlike "wait" function. This prevents waiting an older process that is running in background instead of newly created child process. If '***iswait***' equals to '1', which means current command line does not have '&', ***waitpid*** function runs with parameter of '***rc***' which is the process ID of latest child process.

If current command is not '***wait***', if condition part comes which determines the arguments array initialization size and ***redir*** integer by checking the " < " , " > " substrings in the line string. If there is " < " then ***redir*** is set to -1, if there is " > " ***redir*** is set to 1 and if no redirection character exist in line string ***redir*** keeps the initial value of 0. After this part, by looking the value of ***redir*** parsing tokens of the line string starts. With the help of the '***strtok***' function, commands are parsed and command information are printed with proper format, tokens are cleared from quotation marks (") and apostrophes (') with '***clearSpecial***(char*)' function and they are put in the arguments array in order to be executed later. If ***redir*** is -1 or 1 (there will be redirection), parsing part works same except when there is redirection last token, which is the file name, is added to "`./`" string with '***strcat***' function and stored in '***filepath***' variable in order to make it's format proper.

After parsing and storing tokens in array completed, program forks. For child process condition, if there is redirection out: *“**close**(STDOUT_FILENO)”* and *“**open**(filepath, O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU)”* functions run in order to change output description of the child process to filepath. If there is redirection in: *“**close**(STDIN_FILENO)”* and *“**open**(filepath, O_RDONLY)”* functions run in order to change input description of the child process to filepath. And finally ‘***execvp***’ function executes with the arguments array.