CSC375: Operating Systems

Assg. 1

Due: June 24, 2022

In this assignment, we practice writing a simple shell program that can interpret and execute simple command lines, including the following features (we will improve on this program to support more features such as pipeline, standard input/output redirection later).

1. Implement the function **ReadFillCommandArgs**

   If you have difficulty with parsing a command line into command line arguments, you can implement this function by prompting and reading each word of the command separately and storing them into the command line arguments array. Note that if you choose to read input this way, your shell script requires the user the have EOL at the end of each line.

```
Enter the name of the command:ls
Enter options/arguments (Enter EOL/# to end):-l
Enter options/arguments (Enter EOL/# to end):EOL
Running ls -l
total 484
drwxr-xr-x 12 pia staff  4096 Feb 22 21:54 .
drwxr-xr-x 43 pia ang staff  4096 Feb 12 17:50 ..
-rw-r--r--  1 pia staff 37184 Jan 11 14:09 330px-Operating_system_placement.png
-rw-r--r--  1 pia staff  3078 Jan 22 10:35 access.html
-rwxr-xr-x  1 pia staff  8959 Jan 29 12:27 a.out
-rw-r--r--  1 pia staff  5838 Feb 21 19:54 assignments.htm
drwx------  5 pia staff  4096 Jan 11 14:09 CISC3595

Enter the name of the command:....
```

   In this sample code, you can see how this is done in a program:

```
...
const int MAX_WORD_IN_CMD=20;
const int MAX_WORD_LEN=30;
char words[MAX_WORD_IN_CMD][MAX_WORD_LEN];

int main()
{
   char * argv[MAX_WORD_IN_CMD];

   //EZ: Below we demo how to read two words
   cout <<"Enter first word:";
   cin >> words[0];
   cout <<"Enter next word:";
   cin >> words[1];

   cout <<"first word " << word[0] <<endl;
   cout <<"next word " << word[1]<<endl;

   //set the arguments array, to point to each word...
   argv[0] = words[0]; //argv[0] points to the first word
   argv[1] = words[1];
   argv[2] = NULL;

   /*  now argv is ready to be used ...
   execvp (argv[0],argv);
   cout <<"Problems\n"<<endl;
   */
}
```

2. Support both interactive mode and scripting/batch mode

   Recall that if the shell program is run with a command line argument:

   ```
   ./myshell script.txt
   ```

   Then the shell program should read and execute the command lines from the file **script.txt** (instead of the standard input in the interactive mode). When it reaches the end of the file, the shell program ends. (In interactive mode, the shell program ends when user types **exit** command).

   Note on reading files: You can use higher-level library functions such as fopen, fread, fscanf, or C++ classes to read the file.

   ```
   //This code segment demonstrates how to open a file (to read), and how to read
   the file...

   ifstream scriptFile;
   scriptFile.open (argv[1]);

   if (!scriptFile.is_open())
   {
       //report errors
       _exit (1);
   }

   scriptFile.getline (cmdline,MAX_LINE_LEN); //read from the scriptFile until a
   newline character is
             //encountered, or until the MAX_LINE_LEN chars have been read
   (whichever happens first),
             //and save the bytes into the buffer cmdline, terminated by '\0'
             // i.e., strlen (cmdline) will tell you how many bytes have been read
   ...
   ```

3. Support running a command in background:

   When the last character in the command line is the &, the parent process running the shell does not **wait** for child process to exit, instead it will display the child process PID in bracket, and continue on to display prompt, read and execute next command line.

4. Ignore comments in script file (everything follows # sign)

Submission

Please upload your program to GitHub. Share with me your file.