Istanbul Technical University March 5, 2024

BLG312E Computer Operating Systems
Spring 2024
Homework 1

Regulations

Due Date: Sunday, March 24, 2024, 23.59

Submission: Compress all your codes (C source code, other files) as NameSurname_StudentID.tar.gz and send it via Ninova before the due date. Late submissions will not be graded.

Team: There is no teaming up. The homework has to be done individually.

Cheating: All kinds of cheating is forbidden. All parts involved (source(s) and receiver(s)) get zero.

If you have any questions, please e-mail teaching assistant **Esin Ece Aydın (aydinesi16@itu.edu.tr)**.

Overview

In this assignment, you will implement a command-line shell. Your shell can be run in batch mode. It is initiated by specifying a batch file on its command line; the batch file contains the list of commands, each on its own line, that should be executed. In batch mode, you should echo each line you read from the batch file back to the user before executing it; this will aid in debugging your shell. Your shell stops accepting new commands when it encounters the 'quit' command on a line or reaches the end of the input stream (i.e., the end of the batch file or the user types 'Ctrl-D'). The shell should then exit after all running processes have terminated.

Implementation Details

- Each line may contain multiple commands separated by the ; or | characters.
  - i.e, a line of input might look like this: `command1 ; command2 | command3`
- Each of the commands separated by a ; should be run simultaneously or concurrently.
  - Note that this behavior differs from standard Linux shells, which run these commands one at a time, in order.
- The shell should not print the next prompt or take more input until all of these commands have finished executing
  - the wait() and/or waitpid() system calls may be useful here.
- To exit the shell, the user can type "quit." This should simply exit the shell and terminate the program:
  - The exit() system call will be useful here.
  - Note that "quit" is a built-in shell command and should not be executed like other programs the user types in.
  - If the "quit" command is on the same line with other commands, ensure that the other commands execute and finish before the shell exits.
- Your program should support the "cd" command and the "history" command:
- The "cd" command affects all commands after it.
- Change the working directory when the "cd" command is executed.
  - The "cd" command does not run with other commands.
  - Assume that the "cd" command appears alone in the command line.
- The "history" command only shows previous commands you have run in the program.
- You can assume that length of one line in batch file is maximum 250 character.

For example, the commands "`ls -a ; grep foo file2`" should all be running at the same time; as a result, you may see that their output is intermixed. Each group of commands separated by a pipe (|) should be run similarly to standard Linux; however, the result of all commands before the | must be transferred as a parameter to all commands after the |. For instance, in the command "`cat f1.txt ; cat f2.txt | grep hw ; grep odev`", the "cat" operation must be applied to both files simultaneously, and the "`grep`" operation must be applied to their contexts for both "`hw`" and "`odev`".

Example Command line output when executing script.sh batch file in Linux (f1.txt, f2.txt and script.sh is given in Ninova):

```
$ bash script.sh
hw f1
odev f2
odev f1
hw f2
```

Additional Notes:
Look into `fork()`, `execvp()`, and `wait/waitpid()`. Read the man pages for more details.

The `fork()` system call creates a new process. After this point, two processes will be executing within your code. You will be able to differentiate the child from the parent by looking at the return value of fork; the child sees a 0, the parent sees the pid of the child.

You will note that there are a variety of commands in the exec family; for this project, you **must** use `execvp()`. Remember that if `execvp()` is successful, it will not return; if it does return, there was an error (e.g., the command does not exist).

The most challenging part is getting the arguments correctly specified. The first argument specifies the program that should be executed, with the full path specified; this is straight-forward. The second argument, `char *argv[]` matches those that the program sees in its function prototype:

```
int main(int argc, char *argv[]);
```

Note that this argument is an array of strings, or an array of pointers to characters. For example, if you invoke a program with:

```
foo 205 535
```

then `argv[0] = "foo", argv[1] = "205" and argv[2] = "535"`.
Important: the list of arguments must be terminated with a NULL pointer; that is, `argv[3] = NULL`.

We strongly recommend that you carefully check that you are constructing this array correctly!