

Project A

Reading: Tanenbaum, Sections 2.3, 2.4

Due: Feb. 27 and Mar. 6, 2006

Overview

You are to write and test a C/C++ program that implements a simple shell whose executable is called `xssh` and is a modified form of `xssh0`. This document specifies a basic `xssh`. The follow-on to this project will add a few more features and lift some simplifications assumed here. It has an optional `-x` flag:

```
xssh [-x]
```

`xssh` normally reads commands from `stdin`, one command per line. The `-x` flag indicates that the command line *after* variable substitution should be displayed before the command is evaluated. In the description below, a proper word indicates a metasymbol, square brackets (`[]`) indicate an optional word(s), and `"..."` indicates 0 or more words. It has the following builtin (internal) commands (**CAVEAT:** There are some small differences between these commands and those in Homework 3.):

- `echo [Word] ...`: Display the arguments followed by a newline. Multiple spaces/tabs should be reduced to a single space.
- `quit [N]`: Quit the shell with an exit status of *N*. If *N* is omitted, the exit status is that of the last command executed.
- `"bg Cmd ..."`: The remainder of the line should be run in the background with the word following `bg` treated as a non-builtin command.
- `"wait Pid"`: The shell should wait for child process *Pid* to complete. If *Pid* is `-1`, the shell should wait for any child process to complete.
- `"pause Pid"`: The process with PID *Pid* should be put to sleep.
- `"resume Pid"`: The process with PID *Pid* should be allowed to continue to run.
- `"set Name Value"`: Set a variable name (which could be an environment variable) to a value. A user-defined variable name is a sequence of letters, digits or underscores. There is one special single-character variable name described later: question mark (`?`). The value of a variable is indicated by preceding the name with the dollar sign. For example, `'set XYZ 32'` sets the variable `XYZ` to the string `32`. The value of `XYZ` is denoted by `$XYZ`. Note that the variable name is the longest possible name (e.g., if there are variables `X` and `XY`, `$XY` refers to the variable `XY`).
- `"export [Name] ..."`: The names are exported to the current environment (and subsequent children). If there are no arguments, list the exported names and their values.
- `chdir [Pathname]`: Change the current directory to *Pathname* which can be an absolute or relative Unix pathname. If *Pathname* is not given, change the current directory to the path given by the environment variable `HOME`. The current directory should be maintained in an environment variable called `PWD`.

Here are the other features of `xssh`:

- a) Multiple spaces/tabs are reduced to a single space during the substitution and line scanning phase.
- b) The command line prompt should be the three character sequence '`>>>`' (i.e., `>`, `>`, space).
- c) A child process inherits all of its parent's environment variables.
- d) `XYZ` is the string resulting from the concatenation of the values of the variables `XY` and `Z`.
- e) A non-builtin command is assumed to be a Unix executable that can be found in a directory listed in the `PATH` environment variable.
- f) All undefined variables have a value of the null string. By *convention*, we use `$.` as a special variable whose value is always the null string and is used to terminate a variable name. (e.g., `X.Y` is the value of the variable `X` concatenated with the string `"Y"`).
- g) The `#` character signifies the beginning of a comment. All other characters following and including the `#` should be ignored during interpretation.
- h) `xssh` understands the shell variables `$$`, `%?` and `#!`, and these variables have the same meaning as in `sh` and `bash`.
- i) Blank lines should be ignored.

Note that there is simple variable substitution, but there is no filename substitution nor command substitution. See `fork(2)`, `waitpid(2)`, `execvp(2)`, `sh(1)`, `gettimeofday(2)`, `exit(3)`, `getenv(3)`, `putenv(3)`, `chdir(2)`, `kill(2)`, `fgets(3)`.

Implementation Notes

In this implementation, you can assume that the system will be small and therefore simple data structures are appropriate (i.e., there is no need at this time for sophisticated data structures). It is up to you to determine what you will need, but remember that simplicity will be a virtue in this assignment. Furthermore, for this part of Project A, you can assume that an error should terminate your shell with an error message.

The main processing loop of the `xssh` interpreter looks like this:

```
Initialize;
Process command-line args;
while ( (line=getcmd()) != EOF ) { // prompt&get cmd until EOF
    (nwords, word[]) = parse(line); // word[i] points to ith word
    word[] = do_subst(word[]);      // do var substitution
    quit = eval_builtin(nwords,word[]) if (is_builtin(word[0]));
    status = eval_nonbuiltin(nwords,word[]) if (!is_builtin(word[0]));
} until (quit);
Cleanup;
```

Additional Guidelines

- Code readability is of the utmost importance. The Web page will contain a summary of coding guidelines that you should follow in spirit. I am not rigid about these guidelines, but unreadable code will be penalized.
- All system calls that should not fail (e.g., `fork(2)`) should be wrapped so that any fatal errors will cause an error message to be displayed followed by an `exit`. If you can recover from a system call error (e.g., `execvp(2)`), output an error message and continue. I will not examine any solutions that do not follow this rule. By convention, the wrapped system call name will be the same as the actual system call name except the first character should be capitalized (e.g., `Fork` is the wrapped version of `fork`).

What to Submit by 4 PM, Feb. 27

Electronically submit the following **PLAIN TEXT** files by 4 PM, Feb. 27 (instructions follow):

- 1) The **PLAIN TEXT** file *Feb27-questions.txt* which contains answers to the questions in the file *Feb27-questions.txt*. Leave the questions in place. The first question asks about the status of your code. You should indicate which features are working, which have buggy or incomplete code, and which have not even been attempted. For buggy or incomplete code, give an indication of its incompleteness.
- 2) A single-page, **PLAIN TEXT** design guide file called *Design.txt* containing the most important conventions, ideas and implementation notes related to your project. Do not use any font size less than 10 points.
- 3) The output of your interpreter for the test case labeled *Feb27-test.in* on the course Web page. Name this file *Feb27-test.out*. (It is not expected at this time that your interpreter will be able to handle all of the cases.)
- 4) The **SOURCE FILES** for your latest version of the `xssh` interpreter. **NOTE: I do NOT want object code or executables.** As a minimum, this code should contain the control structure for handling all built-in commands with stub code for each evaluation function. The stub code should display a message that says that the function was entered.

There will be only three grades assigned: 0, 5 or 10 points. The grade assigned will be subjective and based on perceived effort.

Feb. 27 Electronic Submission: The end result should be that you mail to `kenw@arl.wustl.edu` a single *shar* (shell archive) file containing your files. Do **NOT** submit object code or executables. The following commands will create a *shar* file named `A.shar` containing the required files and then send mail to me:

```
shar Feb27-questions.txt Design.txt Feb27-test.out ... source files ... > A.shar
mail -s A.shar kenw@arl.wustl.edu < A.shar      # mail is usually in /bin
```

If you prefer, send the *shar* file as an attachment and use whatever mailer you are comfortable with. **NOTE:** The *shar* file should be relatively small (try `ls -l`) and make sure it is not more than a few hundred thousand bytes.

What to Submit by Mar. 6

The CS422S Web page contains a link to the documentation template. You should complete the template and submit it in both hardcopy AND electronic form. Submit the completed documentation template AND a listing of the source code. The electronic submission (described below) should include the completed documentation template, the source code, the Makefile, test scripts, and test output. The electronic copy is due by 2400 hours Mar. 6. The hardcopy can be submitted in class on Mar. 6 or to my office by 4 PM Mar. 7. This submission is worth 100 points.

Words of Caution

Here are some observations from my years of experience with projects like this:

- **The documentation template is non-trivial.** Do not expect to complete it in less than an hour. Furthermore, working code but no documentation is almost useless. So, don't forget to fill out the documentation template.
- Trivial bugs can consume tens of hours of time. Yes, I said tens of hours, not just hours. You need to start small, test the control structure, and incrementally add features on top of what seems like rock solid code.
- Keep different versions of your "rock solid code versions" so that you can rollback to and recover from a stable version. This also helps if you mistakenly delete your latest source code!!!
- Don't ignore error messages and think they will disappear on their own. They don't. They just come back and bite you when you least want to be bitten.
- Try to understand the origins of your bugs rather than always doing trial and error changes. (Some trial and error may be appropriate in small test cases.)
- If the approach you are taking seems like it will be a nightmare to implement, then don't implement it. Find a better way or better understand the system calls you are trying to use.
- The more lines of code you have, the more chances for bugs to appear.
- Have a plan. Don't try to do everything at once.

Electronic Submission

The end result should be that you mail to kenw@arl.wustl.edu a single *shar* (shell archive) file containing your files. Do **NOT** submit object code or executables. The following commands will create a shar file named A.shar containing the files xssh.c and other files and then send mail to me:

```
shar README Makefile xssh.c ... other files ... > A.shar
mail -s A.shar kenw@arl.wustl.edu < A.shar      # mail is usually in /bin
```

The README file is the completed documentation template. If you prefer, send the shar file as an attachment and use whatever mailer you are comfortable with. **NOTE:** The shar file should be relatively small (try `ls -l`) and make sure it is not more than a few hundred thousand bytes.

Late Policy

There is no late submission date for the first deadline. You must submit in class on that date. The final submission can be one week late for a 20% penalty. Note that you should submit something even if the final version still has bugs.