

Project B

Due: Noon Wed, Dec. 13, 2006

Overview

The goal of this project is twofold: 1) Fix any bugs from Project A, and 2) extend your Project A to remove some restrictions and add new features. Furthermore, you will be expected to take more initiative in gaining an understanding of library functions required to implement features. Here are the changes to existing features and commands:

- `xssh` should continue to run even in the face of errors.
- All builtin commands should be checked for the proper number of arguments if possible.

This version of `xssh` should support pipelines, job control, a limited form of CPU limits, and run-away process detection. It should also respond to the terminal interrupt character (normally `ctrl-c`) as described below. Here are some specifics:

- A pipeline has the same syntax as in other shells; i.e., '`Command | Command | ... | Command`' is a foreground pipeline, '`Command | Command | ... | Command &`' is a background pipeline. Note that the Commands can have arguments and `stdin/stdout` redirection. Although there are a variety of ways to implement a pipeline, full credit will only be given if `xssh` is the parent of all processes in the pipeline.
- The terminal interrupt character (normally `ctrl-c`) generates the interrupt signal (`SIGINT`). `xssh` should terminate the foreground process(es) and return to the command line prompt. However, it should not terminate background processes or `xssh`.
- The terminal stop character (normally `ctrl-z`) generates the stop signal (`SIGSTP`). `xssh` should stop the foreground process(es) and return to the command line prompt. However, it should not stop background processes or `xssh`. The stopped processes can be continued by entering the `bg` command which puts the processes in the background where they continue running. The processes can be brought back to the foreground through the `fg` command.
- The user should be notified about any background jobs that have terminated right after the command line prompt is displayed.
- `xssh` should understand the `mytime` command which should return the real time, user time and system time spent by a command. For example, `mytime ls` should return the time to execute the `ls` command.
- `xssh` should understand the `mylimit` command which should limit the CPU usage of the command in the same way the `ulimit -S -t n` command can limit the CPU usage to `n` seconds for the Bash shell. For example, `mylimit 2; fib -n 100` should allow `fib` to use no more than 2 seconds of CPU time.

Note that you are not allowed to use the `system(3)` library function.

What to Submit

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 AND hardcopy is due by noon Dec. 13. The hardcopy should be submitted to my office.

Early Submission: You can submit a shortened documentation template if you can submit by noon Dec. 8. Here are the requirements for early submission:

- The program must be bug free (as far as you can tell).
- Schedule a demonstration of your program with me by noon Dec 8. You will be asked to demonstrate your program, and you may be quized about aspects of the program.
- There is no hardcopy submission.
- The shortened documentation template contains only instructions for running and testing your project and a full description of any extra credit work.
- Submit electronically by noon Dec 8. The README file contains the shortened documentation template instead of the full template.

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 B.shar containing the files xssh.c, in1.txt, out1.txt, and other files and then send mail to me:

```
shar README Makefile xssh.c ... in1.txt out1.txt ... > B.shar
mail -s B.shar kenw@arl.wustl.edu < B.shar      # mail is usually in /bin
```

The README file is the completed documentation template.

Late Policy

Late submissions will not be accepted except in extraordinary circumstances.

Extra Credit

Design, implement, test and document a local logging daemon. The logging daemon should be able to communicate with multiple `xssh` instances through one or more named pipes. All non-control messages sent to the daemon should be logged atomically to a single file. One such usage is to log the CPU usage of every command evaluated by `xssh`