Homework Assignment 5
COP4342 Unix Tools (100 points)
Due: 12/3/2019

---

You must implement all programs using Perl.

**Part I (50 points).** In this problem, you will need to sort a list of lowercase character strings according to the following ordering rules. In the following, let str1 and str2 be two strings.

- Let vow1 be the substring of vowels appearing in str1, and vow2 the corresponding substring of str2. (Characters in vow1 and vow2 appear in the same order as they appear in str1 and str2, respectively.) If the substring vow1 appears before vow2 in the ascending ASCII order, str1 appears before str2 in the output.
- If, after considering the above rule, there are ties among strings, break ties using the regular ascending ASCII order of the original strings str1 and str2.

Your program should read the strings from a file (given as the command-line argument). In this file, each line contains one string. Your program should print the results to the standard output, one string per line. An example run of the program is given below.

```
duan@linprog4 (~...homework/solutions) % perl sort_strings.pl
sort_string.input
apparate
fanfare
panacea
albatross
albacore
false
parmesan
candelabra
beans
vermeil
duan@linprog4 (~...homework/solutions) %
```

**Part II.** (50 points) In our lecture, we discussed a simple echo server and echo client, where the client can only send a single line to the server, and the server is implemented as a sequential server. In this problem, you need to extend the client and server programs in the following way.

- The client program should continue read the user input line by line, until the user types "quit", or end-of-file is encountered, or the program is killed.
- Instead of arbitrary strings, what a user types will be Unix command-line commands, for example, "ls", "who", and "ps". In our test, we will only test simple commands like these, which will produce all outputs and then terminates. In particular, we will not test programs such as "top", which will continue run.

- At the server side, instead of simply echoing back the received string, it should treat it as a Unix command-line command, and run it. The output of the command will be sent back to the client.
- The server should be implemented as a concurrent server; that is, upon each connection being accepted, a child process will be created to interact with the client (read command from client, run the command, and send back the result, and so on). The parent process should continue to accept new connection requests.

An example run of the program is given below.

```
duan@linprog2 (~...homework/solutions) % perl con_server.pl 6789
One connection is established.
One connection has been disconnected.
```

// the server continue runs

```
duan@diablo (~...homework/solutions) % perl con_client.pl linprog2.cs.fsu.edu 6789
linprog2.cs.fsu.edu 6789
MSG TO CLIENT: ps
MSG FROM SERVER:   PID TTY          TIME CMD
23709 pts/3    00:00:00 reg-tcsh
23955 pts/3    00:00:00 perl
23958 pts/3    00:00:00 perl <defunct>
23967 pts/3    00:00:00 perl
23968 pts/3    00:00:00 ps
MSG TO CLIENT: who
MSG FROM SERVER: duan     pts/3        2011-10-24 11:37 (diablo.cs.fsu.edu)
MSG TO CLIENT: quit
```

**NOTE (IMPORTANT):** This program does not have any security features. It will allow anyone to connect if they know the port number of your program. Remember to kill the server after you are done with testing --- do not let your server run over a long time period. In real situation, we will verify some password before running any commands.

**Submission:** Tar your programs into a single file and submit online via Canvas. Make sure you tar your programs correctly. You are responsible for incorrect submissions (for example, empty tar file). You can untar the file **under a different directory** to make sure that you do include both programs in the tar file. Same late policy applies if the submission is incorrect and you need to submit a new version.