EE 468
Project A Simple Shell
Total Points = 10.

Project A is an individual assignment, and it's attached as Project A Pipe Simple Shell.zip. Upload your solution code, and it must be able to run on wiliki

This project depends on knowledge of managing processes and communication between processes, e.g., pipes and dup2. A prerequisite for this project is EE 367L, and in particular the EE Client-Server lab assignment. If you have not taken EE 367L, then the client server lab assignment is provided in the Appendix of this document. The critical portion you should know is Part 1. Part 2 is also important because it's about sockets and networking. But you don't need that yet.

In Project A Pipe Simple Shell.zip, there is a simple shell program sshell.c. It currently can't do redirection, e.g., '<', '>' and '|'.

Rewrite the program so that it can do pipes ('|'), e.g.,

cat <file name> | grep and | hexdump -x

which takes the file as input to grep for the word 'and', which in turn is input to hexdump.

Note that the program must be able to

- Parse the input line
- Identify the pipe symbols ('|')
- Then launch each command as a process, using fork() and execvp()

Also the processes must be connected together with pipes.

$$\begin{tikzcd}
\text{<file name>} \arrow[r] & \text{grep and} \arrow[r] & \text{hexdump -x} \arrow[r] & \text{stdout}
\end{tikzcd}$$

These pipes become replacements for stdin and stdout.

For your reference, included in this directory is 'pipe.c', an example of implementing this using dup2. (Recall that this example came from EE 367L.)

You may assume that the maximum number of pipe symbols ('|') in a command line is 4.

Upload your updated version of sshell.c into laulima. Your code should work on wiliki, it compiles and will run.
Grading: Maximum points = 10

- 10 pts: The shell should be able to do up to 4 pipes. Note that the in[] and out[] arrays may have to organized into structs since you may need 4 of them. Your shell should not create zombie processes if it properly exits.
- 8 pts: The shell be able to do one pipe.
- 6 pts: For some effort is given. Significant changes in the code and it compiles.
- 0 pts: Little or no effort is given. It doesn't compile or very little or no changes to the code.

Hint: Here are some definitions.

size_t is a data type used to represent sizes of objects. It's part of the 1999 ISO C Standard.
char *buf_args[args_size] instantiation is possible because of C99 C Standard.
char *strsep(char *wbuf, const char *delim): https://man7.org/linux/man-pages/man2/wait.2.html

> Returns NULL if wbuf is an empty string
> Otherwise it returns the first "token" that terminates with a character in the string delim.
>
> > For example if delim is "` \n\t`", then there are three delimiter characters
> > `' '`, `'\0'`, and `'\t'`.
>
> strsep returns the pointer to the token, i.e., the original value of wbuf
> wbuf is updated to the point after the token
> For example, suppose wbuf = " Aloha World\n". After multiple calls, the string becomes
> "\0Aloha World\n\0"
> "\0Aloha\0World\n\0"
> "\0Aloha\0World\0\0"

execvp is like exec but with variable input parameters

Hint: The current version of sshell.c is a loop that that prompts the user, parses an input string into "tokens", then calls execvp using the tokens as input parameters. Note that the tokens is for one command which includes the command itself (e.g., 'ls', 'grep', or 'hexdump') and then the string of parameters (e.g., '-l', 'and', etc).

Modify the code so that it can handle pipes. This means that it must launch multiple commands (with their parameters) in its own process (use fork and execvp). The inputs and output of these processes may need to be redirected. In particular the output of one process is redirected as input to another process.

Hint: You can find tutorials on pipes and filters on the Internet. Here are some examples.

http://www.tuxradar.com/content/exploring-filters-and-pipes

http://www.ibm.com/developerworks/library/l-lpic1-v3-103-2/

Attached files

- sshell.c – You modify this to include pipes.
- EE367Lab3 – Lab in EE 367L that used pipes and the exec command
- EE367LOverviewClientServer – Overview of client server. This covers processes, etc.
- UNIX-Intro – Review of UNIX (or Linux) commands, which is optional if you're familiar with Unix/Linux
- pipe.c – example of how to use pipes
- almamater – a file to test

Testing: cat almamater | grep and | grep her | hexdump -x

Appendix

In the zipped folder EE 367L, is a lab from EE 367, which covers managing processes and communication between processes. There are two parts which are described below. As mentioned earlier, you need to know Part 1. Part 2 is optional. However, since it covers the important topic of sockets, it's left for reference.

Part 1 Background and Underlying Technologies:

Video overview of Part 1: https://drive.google.com/file/d/1pfx-ZNunQp-8S7pNIui0UWJZZn5dKAF9/view?usp=sharing

Attached files:

- Client-Server Lab-Part1.pdf (attached file): slides associated with the video
- Client-Server Lab-Part1-Markup.pdf: marked up version of the slides
- EE367L-ClientServer-Part1.pdf: Reading assignment covering the topic of Part 1
- Code-Part1.zip: the code

Part 2 Client Server System and the Assignment

Video overview of Part 2: https://drive.google.com/file/d/1gCQuK7Xxw-4Jdhs4p_LT9cFsU9d3eZKH/view?usp=sharing

Attached files:

- Client-Server Lab-Part2.pdf: slides associated with the video
- Client-Server Lab-Part2-Markup.pdf: marked up version of the slides
- EE367L-ClientServer-Part2.pdf: Reading assignment covering the topic of Part 2. This includes the lab assignment.
- Code-Part2.zip: the code