Programming Assignment 1: Minix Shell

Due April 6, 2024

The Basics

The goal of this assignment is to get each group up to speed on system programming and to gain some familiarity with the system call interface. A secondary goal is to use some of the programming tools provided in the Minix environment. In this assignment you are to implement a Minix shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on your favorite shell to see what it does.

You will need to write a lexical tokenizer that uses getline() to parse a line of input at a time. getline() returns an array of pointers to character strings. Each string is either a word containing the letters, numbers, '.', and '/', or a single character string containing one of the special characters: ( ) < > | & ;.

To compile you can use "clang source.c -o my_prog".

Replace: source.c with your source file and my_prog with your program name.

The Details

Your shell must support the following:

1. **The internal shell command "exit" which terminates the shell**.
   Concepts: shell commands, exiting the shell
2. **A command with no arguments**
   Example: ls
   Details: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect.
   Concepts: Forking a child process, waiting for it to complete, synchronous execution
3. **A command with arguments**
   Example: ls -l

Details: Argument 0 is the name of the command
Concepts: Command-line parameters

4. **A command, with or without arguments, executed in the background using &.**
   For simplicity, assume that if present the & is always the last thing on the line.
   Example: ls &
   Details: In this case, your shell must execute the command and return immediately, not blocking until the command finishes.
   Concepts: Background execution, signals, signal handlers, processes, asynchronous execution
5. **A command, with or without arguments, whose output is redirected to a file**
   Example: ls -l > foo
   Details: This takes the output of the command and puts it in the named file
   Concepts: File operations, output redirection
6. **A command, with or without arguments, whose input is redirected from a file**
   Example: sort < testfile
   Details: This takes the named file as input to the command
   Concepts: Input redirection, more file operations
7. **[Extra credit] A command, with or without arguments, whose output is piped to the input of another command.**
   Example: ls -l | more
   Details: This takes the output of the first command and makes it the input to the second command.
   Concepts: Pipes, synchronous operation

Note: You must check and correctly handle all return values. This means that you need to read the man pages for each function to figure out what the possible return values are, what errors they indicate, and what you must do when you get that error.

Tips:

- Remember that the 0 file descriptor is for standard input, and the 1 file descriptor is for standard output. You need to keep this in mind when you want to override the descriptor for a child process if its input is to be directed from a file or its output is to be directed to a file.
- Remember that when your shell is started (which, confusingly, you'll start from within another shell), it enters into the main() function defined in your program. Technically, in Unix, the main() function takes three arguments.

```
int main(int argc, char **argv, char **envp);
```

The first, argc, tells how many arguments there are. The second, argv, lists the arguments. (Your shell program doesn't need to process any arguments, so you can ignore these first two parameters.) The last, envp, points to the first element of an array of pointers to the first character of a environment variable

definition. The definition is a string containing the variable's name and value separated by an equal sign.

- Useful system calls include, but are not limited to:

read()	for reading commands from the user (used indirectly via fgets())
write()	for printing to the display (used indirectly via printf())
chdir()	for changing the current working directory in cd built-in
stat()	for identifying executable files within the path
fork()	for creating a child process
close()	for closing standard I/O for redirection (cf textbook, bottom p~109)
open()	for opening files for redirected I/O
execve()	for executing an executable file within the child process
exit()	for terminating the child process on an error
waitpid()	for waiting for the child process to terminate

What to turn in

A compressed tar file of your project directory, including your design document. You must do "make clean" before creating the tar file. In addition, include a README file to explain anything unusual to the TA — testing procedures, etc. Your code and other associated files must be in a single directory so they'll build properly in the submit directory.

REMEMBER: *Do not* submit object files, assembler files, or executables. Every file in the submit directory that could be generated automatically by the compiler or assembler will result in a 5 point deduction from your programming assignment grade.