CS 452 – Lab1
C Shell Implementation

Objective

Experience in systems programming with C making use of the system call interface and programming tools provided in the Unix environment. Using C and the incomplete set of the code provided, you are to implement a simple 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 are provided with two main files called `lex.c` and `myshell.c`. Download the files here. The first file `lex.c` contains some code that uses `getaline()`, a function provided by `lex.c` to get and parse a line of input. `getaline()` 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: ( ) < > | & ;.

`myshell.c` contains the code that implements the shell. Obviously, some important pieces are missing and your goal is to insert more functionality into the existing functions to make it a more comprehensive shell. These files will be emailed to you at an appropriate time.

To compile `lex.c`, you have to use flex: "`flex lex.c`". This will produce a file called `lex.yy.c`.

`lex.yy.c` and `myshell.c` must then be compiled and linked in order to get a running program. In the link step you also have to use "`-lfl`" to get everything to work properly. Use gcc for the compilation and linking.

It is imperative that you understand the code thoroughly, especially with respect to the types of communication between processes and the way `stdin` and `stdout` are manipulated.

If you run the program and do not get any arguments in the arg arrays, you might want to make sure that you do a `dos2unix` on the files.

Requirements

Since the provided code is incomplete, you are required to ensure that you successfully parse each command and execute them. Forking a process to execute the commands is typical.

The provided code implements some but not all of the following:

1. **The internal shell command "exit" which terminates the shell.**
   Concepts: shell commands, exiting the shell
   System calls: `exit()`

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
   System calls: `fork(), execvp(), exit(), wait()`

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, whose output is redirected to a file**
   Example: `ls -l > foo`
   Details: This takes the output of the command and put it in the named file
   Concepts: File operations, output redirection.
   System calls: `freopen()`

5. **A command, with or without arguments, whose output is appended to a file**
   Example: `ls -l >> foo`
   Details: This is an append, which is a variation of the output redirect (see above)

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
   System calls: `freopen()`

7. **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: `vi &`
   Details: In this case, your shell must execute the command and return immediately, not blocking until the command finishes. The distinction must be made between backgrounding a process that does not need interactive input and one that does, e.g., the ***`who`*** command vs. the ***`vi`*** command.
   Concepts: Background execution, signals, signal handlers, process groups, asynchronous execution.
   System calls: `sigset(), sigaction()`
   Signals: `SIGTTOU`

8. **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 System calls: **pipe()**
   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.

A Simple Solution

The command that was just parsed is passed to the system call function `system()`. This bypasses all the code that is provided in `myshell.c`. This does not comply with the specifications of this assignment and is not acceptable.

Demo Utilities

Examples of some UNIX commands/utilities to be used in the demo are: `sort`, `grep`, `parenthesis`, `&&`, `||`, `tr`, `sleep`, `sed`, `;` (semicolon command separator), in addition to basic UNIX commands.

Distribution of Scores

Based on the items described above by numbers:
Item# 1 through 6 = 40%
Item# 7. Background processing 30%
Item# 8. Pipes 30%

What to submit

1. W drive:
   A directory consisting of the source code, makefile, and README files tar'ed or zipped and put under the cs452/lab1 directory on the W drive. Name your directory with both your email names, e.g., wagnerpj_tanjs_lab1.tar.

2. Email as attachment (tar'ed or zipped project) and percentage workloads.

3. Print, fill, and submit the **lab cover page** and **honor code** shown below.

4. Make an appointment to demo your project.

Project is only considered completely submitted, i.e., to get a grade, if **ALL** items mentioned above are accomplished.

CS 452 – Operating Systems Lab Assignment Cover Page

Name(s) ____ Lab Assignment No.: 1

1. Include a copy of your source code, a design document (if required), and a sample run
2. Fill in the table of contents including the names of routines and the corresponding pages.
3. Indicate the status of your program by checking one of the boxes.
4. Submit the assignment at the beginning of the class on the due date.

Program Status: (check one box)
□ Program runs with user-defined test cases. □ Program runs with some errors.
□ Program compiles and runs with no output. □ Program does not compile.

Workload:	Written By			
main.c				
Paper				
Test cases				
Program modules (classes)	Module name	Written by	Module name	Written by
List only major modules				
that are typically more				
than 50 lines of code				

Grading:	Points	Score
Stress Test	40	
Correctness	50	
Documentation	10	
Late Penalty (15% off per day)		
Total	100	

Honor Code

Department of Computer Science

University of Wisconsin – Eau Claire

As members of the University of Wisconsin – Eau Claire community and the computer science discipline, we commit ourselves to act honestly, responsibly, and above all, with honor and integrity in all areas of campus life. We are accountable for all that we say, read, write, and do. We are responsible for the academic integrity of our work. We pledge that we will not misrepresent our work nor give or receive illicit aid. We commit ourselves to behave in a manner which demonstrates concern for the personal dignity, rights and freedoms of all members of the community and those that depend on the expertise we possess.

For all course work in the Department of Computer Science, students will write and sign (if printed) the following: "**I have abided by the Department of Computer Science Honor Code in this work**."

I accept responsibility to maintain the Honor Code at all times.

Name ______________________________

Signature ______________________________

Date ______________________________