SY204 – Sys Prgm & OS Fund Name(s): _____

Alpha(s): _____

Assignment Type:	Programming – Laboratory	Collaboration Policy:	Default
Assignment Title:	0x06: myShell		

1. General.

Due	
Portion of Assignment	Course Server Time / Date
Complete	1800 Friday, 07 Apr 2017

a. Description. Read through this entire assignment before starting work. Beginning with this lab you may use Descriptor I/O or Stream I/O as you see fit. In this assignment you will create a simple shell. Your shell program will take in commands and arguments from a command line and execute the specified command passing in the associated arguments.

2. Explore (10).

a. [3 / __ / 0] Read the man page for the `fgets(3)`, and complete the following.

Regarding the Explore 2.a. `if` statement in the given source code, in your own words explain what the `if` statement is doing and what user input would cause the condition of the `if` statement to be false.

b. [3 / __ / 0] Read the man pages for `exec(3)` and `strtok_r(3)`, and complete the following.

Why does it make sense that the last element in the `strSingleCmdLnArgs` array is `NULL`?

c. [4 / __ / 0] Read the man pages for `getenv(3)` and `setenv(3)`, and complete the following, recall: `$ set` .

What does `getenv` return if the specified environment variable is not set? _____

What character should not be included in the `name` argument to `setenv`? _____

3. Plan (10).

a. [5 / __ / 0] Complete the table below using the system calls and library routines from class meetings: Creating Processes, Monitoring Child Processes, and Program Execution; that you will use in Part 1. Note: you only need to list one `exec` family library routine. You may and are encouraged to discuss this question with classmates.

Order in Source Code	System Call / Library Routine	Used in Process (Genealogy)

Remainder of Page Intentionally Blank

b. [5 / __ / 0] Starter code has been given to you in `myShell.c`. Draw a general block diagram that includes the given blocks and blocks for the sections of code for the system calls and library routines from Paragraph 3.a. Depict the process genealogy in a clear fashion by including genealogy terms for each of the blocks.

4. Implement (40).

a. [0 / -5 / -10] Sound and Secure Cyber Design

- You shall minimize the use of global variables other than `errno`, `opt*`, and for error reporting.
- You are encouraged to use `#define`, and other preprocessing directives.
  - Your program will not be compiled using the `-D` compiler directive.
- Source code is poorly or inconsistently formatted, but can be compiled (readability).
- Source code uses poorly named functions or poorly named variables (readability).
- Source code does not use associated common libraries (reusability).
- Source code does not use library defined constants (portability).
- Source code does not use deprecated language features (design for the future).
- Implementation does not violate Principle of Least Privilege.

b. Usage (All Parts)

- If your `myShell` is started with any command line arguments (other than the command itself), then output a simple usage message.
- After you output the usage message, exit the program with a normal return value.
- Implement the outputting of the usage message as a function your program calls.

c. [5 / __ / 0] Error Handling (All Parts)

- You may assume output to standard output and standard error will succeed (no error checking required).
- For any error that occurs from a system call or library routine (except as amended in above Implementation requirements), output a simple error message to standard error (recall `perror(3)`) with a unique error number for each of the system calls and library routines you use.
- After detecting and reporting an error, your shell shall return to the command prompt of your shell program.
- In the given starter code some error points have been identified for you. Replace generic error messages with detailed error messages; i.e. you final solution shall not say "Error detected by _h0ff", and should instead describe the issue the error handling detected.
- Add error cases as you see fit at given error points and in other places.

e. [5 / 0] Awkward Blinking Cursor (All Parts)

- Create a simple text prompt that will be displayed on standard output when your program is ready to receive a new command line.
- The prompt shall have a single blank space at the end.
- The prompt shall not be a single space character.

- Your prompt may be a static string (non-dynamic).

f. [5 / __ / 0] `0x90` (All Parts)

- If a blank command line is entered, do nothing and display a new prompt.
- For all parts, the prompts shall be in the original `myShell` process.

g. [10 / __ / 0] Part 1 – Single Command Line

- Complete *Awkward Blinking Cursor* and *`0x90`* before beginning Part 1.
    - You should retest previous specifications as you implement additional specifications.
- Execute the command line the user entered.
- After the single command line is executed, display a new prompt.

h. [5 / __ / 0] Part 2 – Secure the `PATH`

- Complete all of Part 1 before beginning Part 2.
- Revisit Activity 2 and Activity 3 from the `a-new-me.c` activity file from the Program Execution discussion.
- If the `PATH` environment variable is unset when your program starts, set the `PATH` environment variable to only search standard UNIX binary locations in this order: `/bin`, `/usr/bin`, `/usr/local/bin`.

i. [10 / __ / 0] Part 3 – It's a Party Line [History is Important]

- Complete all of Part 2 before beginning Part 3.
  Note: Part 3 is more of a challenge, and further develops your skills at reading code that you did not create and seeing how you can reuse and modify that code in other situations. There is enough in the given starter code for you to reuse and modify to complete Part 3.
- In most shell environments the "`;`" is used to separate multiple commands on a single line.
- Parse the full command line entered for "`;`" separated multiple commands.
- Execute each of the commands with arguments entered on the full command line.
- After all the commands are executed, display a new prompt.

5. Test (40).

a. [0 / -20] Code Compilation and Running

- Source code compiles (`gcc -Wall ...`) under Ubuntu 16.04 LTS x86-64 with no warnings and no errors.
- Compiled program runs without crashing or hanging on test cases (no SEGFAULTs, no unintentional infinite loops).

b. [0 / -5] Usage (All Parts)

- Solution correctly outputs a usage message when any command line arguments are entered.
- Solution correctly returns a normal exit value.

c. [5 / __ / 0] Error Handling (All Parts)

- Errors are correctly reported on standard error.
- Error reports include a unique number.
- Detected errors due not cause your program to exit.
- Solution does not output starter code identified generic error messages.

d. [5 / __ / 0] Awkward Blinking Cursor

- Prompt displays on standard output.
- Multiple prompts never display on same line of standard output.

e. [5 / __ / 0] `0x90`

- Prompt is always displayed from original `myShell` process.

f. [10 / __ / 0] Part 1 – Single Command Line (No Arguments)

- Solution correctly executes commands in `/bin`.
- Solution correctly executes commands in `/usr/bin`.
- Solution correctly executes commands using absolute or relative reference.

g. [5 / __ / 0] Part 1 – Single Command Line (With Arguments)

- Solution correctly executes commands in `/bin`.
- Solution correctly executes commands in `/usr/bin`.
- Solution correctly executes commands using absolute or relative reference.

h. [5 / __ / 0] Part 2 – Secure the `PATH`

- Solution correctly protects against an unset `PATH` environment variable.

i. [5 / __ / 0] Part 3 – It's a Party Line

- Solution correctly executes multiple commands on a single command line.
- Solution correctly executes multiple commands on a single command line in the correct order.

6. Submit. Turn in this worksheet and submit source code per the following specifications.

- Turn in this worksheet to your instructor after you have submitted your final source code.
- Submit your final code as a single file named `myShell-#.c`, where # is the part of the implementation that you completed (e.g. `myShell-3.c`), following the submission directions on the course information web page.
- `submit` Assignment (project): *lab06*

[0 / -10 / -15 / -20] Submitted source code that demonstrated insufficient effort (lack of attempt to solve problem).
[0 / -5] Submitted assignment using incorrect assignment type or incorrect assignment number.
[0 / -5] Submitted incorrect number of files in submission.
[0 / -5] Submitted file(s) named incorrectly.
[0 / -5] Submitted file(s) do not include name and alpha.