CSCI 340 - Operating Systems I

Assignment 2 (Part I of II) Total Points 100

Objectives

In this assignment you will develop a simple *command line interpreter* (or *shell*) for the Linux kernel using the C programming language. This assignment will allow you to gain experience in the following areas:

- **More Basic Programming**: This includes variable declaration, data types, arrays, pointers, operators, expressions, selection statements, looping statements, functions, structs, and header files.
- **System Calls**: Become familiar with system calls that are capable of executing external programs, getting and changing the directory, working with environmental variables, and working with processes (i.e. fork, exec, and wait).

System and Standard Lib Functions

This assignment will use the system and standard library functions listed below. Please ensure your familiar with the syntax, and usage of them. Detailed information about each function listed below can be found using the `man` command from the console: i.e. `man chdir`, will show the man page (short for manual page) for the `chdir` function.

- **Change Directory**: `int chdir(const char* path)`
- **Execute External Command**: `int execv(const char* file, char* const argv[])`
- **Get Environmental Variable**: `char* getenv(const char* name)`
- **Error Mapping**: `void perror(const char* string)`
- **Fork a Process**: `pid_t fork(void)`
- **Memory Allocation**: `void* malloc(size_t size)`
- **Free Memory**: `void free(void* ptr)`
- **File/Directory Status**: `int stat(const char* path, struct stat* buffer)`
- **Blocking Operation**: `pid_t wait(int *status)`

Provided Files

The three files listed below are provided to you.

- **shell.h**: Header file that defines the command *struct* used in this assignment, constants, and the function prototypes to be completed by you. **Please note**: You may not add new function definitions to this header file.
- **shell.c**: The file containing the implementation of the functions listed in *shell.h*. Having a different file for the implementation separates interface (the include file) from the implementation (the .c file).
- **hw2.c**: Source code file that includes a stubbed out version of the main function, and defines the libraries and constants used in this assignment. **Please note**: You may not remove, modify, or add (i.e. #include) additional libraries. The ones that are provided, are the only libraries needed for this assignment.

The *hw2.c*, *shell.h* and *shell.c* files contain many comments that provide basic definitions and step-by-step instructions. Please read the comments carefully and follow the instructions.

Todo

Using the *shell.h* file, you must provide working implementations within the corresponding *shell.c* file for the following function prototypes:

1. *void parse(char* line, command_t* p_cmd)*
2. *int execute(command_t* p_cmd)*
3. *int find_fullpath(char* fullpath, command_t* p_cmd)*
4. *int is_bultin(command_t* p_cmd)*
5. *int do_builtin(command_t* p_cmd)*
6. *void cleanup(command_t* p_cmd)*

For each function prototype listed above, numerous comments are provided in the header file to guide you in this assignment. Please read them carefully, they either provide basic step-by-step instructions, or basic calculations.

In the *hw2.c* file, you must fully implement the main function, i.e., *int main(int argc, char** argv)*. In the main method, pseudocode (high-level step-by-step procedure) is provided to you that outlines how your code should function. Please read the comments carefully, and follow their instructions.

Collaboration and Plagiarism

This is an **individual assignment**, i.e. **no collaboration is permitted**. Plagiarism will not be tolerated. Submitted solutions that are very similar (determined by the instructor) will be given a grade of zero. Please do your own work, and everything will be OK.

Submission

Create a compressed tarball, i.e. *tar.gz*, that only contains the completed *hw2.c*, *shell.h* and *shell.c* files. The name of the compressed tarball must be your last name in lower case. For example, *ritchie.tar.gz* would be correct if the original co-developer of UNIX (Dennis Ritchie) submitted the assignment. Only assignments submitted in the correct format will be accepted (no exceptions). Submit the compressed tarball to the appropriate Dropbox on OAKS by the due date. You may resubmit the compressed tarball as many times as you like, only the newest submission will be graded.
To be fair to everyone, late assignments will not be accepted. Exceptions will only be made for extenuating circumstances, i.e. death in the family, health related problems, etc. You will be given a week to complete this assignment. Poor time management is not excuse. Please do not email assignment after the due date, it will not be accepted. Please feel free to setup an appointment to discuss the assigned coding problem. I will be more than happy to listen to your approach and make suggestions. However, I cannot tell you how to code the solution. Additionally, code debugging is your job. You may use the debugger (gdb) or print statements to help understand why your solution is not working correctly, your choice.

Grading Rubric

For this assignment the grading rubric is provided in the table shown below.

Program Compiles	10 points
Program Runs with no errors	10 points
parse() function implementation	20 points
execute() function implementation	20 points
is_builtin() function implementation	10 points
do_builtin() function implementation	10 points
cleanup() function implementation	10 points
main() function implementation	10 points

In particular, the assignment will be graded as follows, if the submitted solution

- does not compile: 0 of 100 points
- compiles but does not run: 10 of 100 points
- compiles and runs with errors: 15 of 100 points
- compiles and runs without errors: 20 of 100 points
- all functions correctly implemented: 100 of 100 points

Additional Guidance

Executing a Command

When a command is executed using the `execv` function, a child process must be created. The following incomplete code segment can be used in the `execute` function defined in `shell.c`.

```
if (fork() == 0) {
      // This is the child
      // Execute in same environment as parent
      execv( ... );   // determine proper arguments...
      perror("Execute terminated with an error condition!\n");
      exit(1);
}

// This is the parent - wait for the child to terminate
wait( ... );    // waitpid() could be used instead
```

File/Directory Status

The `stat` function can be used to determine the existence of a file, or directory, on the file system. The following incomplete code segment can be used in the `find_fullpath` or `do_builtin` functions to determine if the file or directory is on the file system.

```
struct stat buffer;
int exists;

// string that represents the fully qualified
```

```
// path of a file or directory on the file system
char* file_or_dir;

exists = stat(file_or_dir, &buffer);

if (exists == 0 && (S_IFDIR & buffer.st_mode))  {
     // Directory exists
} else if ( exists == 0 && (S_IFREG & buffer.st_mode)) {
     // File exists
} else {
    // Not a valid file or directory
}
```