CSI 402 – Systems Programming

Programming Assignment V

Date given: Nov. 22, 2017 **Due date:** Dec. 07, 2017
Total grade for this assignment: 100 points
Note: Programs that produce compiler/linker errors will receive a grade of zero.

A. Purpose. A shell is a user interface built on top of an operating system. Most modern systems such as Windows, and MacOS include a graphical user interface shell. Many such systems also include a terminal emulator which grants access to a classic command–line shell. A command–line shell is a text based user interface in which the user types commands in order to interact with the system. Each shell (e.g., the bash shell used on most modern Unix systems) accepts a particular set of commands, i.e., has its own command language. For convenience and automation, many shells also accept scripts (a script is a sequence of commands stored in a file, to be executed one after another) as input instead of one command at a time. Your task in this assignment is to develop your own simplified command–line shell, similar to bash. Your shell will interpret script files, in addition to accepting one command at a time from the keyboard.

B. Description. The executable version of your program must be named `simpleshell`. Your `makefile` must ensure this. The `simpleshell` program must support the following usage:

```
simpleshell [scriptfile]
```

In the first case, when no arguments are given, simpleshell should enter a loop in which it accepts one command at a time as keyboard input (e.g., using `scanf`). The user types a command, and then presses 'return' on the keyboard to execute it (i.e., the command will end with a newline character). Your shell **must block** until the command completes and, if the return code is abnormal, print out a message to that effect. In the second case, your shell should open the given file and interpret the contents as a script. You may assume that each line of the script file corresponds to one command. You shell should accept and support the following commands:

- `list`: This is similar to the 'ls' command in bash. The list command prints (to stdout) information about the files contained in a directory. Note that list is a separate program from your shell. It can therefore be called from simpleshell, or can be called directly from bash, by manually executing its executable file. `list` should support the following usage:
  - `list pathname`, where pathname is an absolute or relative pathname to a directory. The list program should print the names of all non–hidden files contained in the specified directory, one file per line. Recall that non–hidden files are files whose name does not start with a dot.
  - `list -i pathname`, where the flag `-i` indicates that extra information should be printed. In this case, list should print each file's name (one file per line), size (in bytes), permissions (as an octal integer), and inode number (as a decimal integer).

  - `list -h pathname`, where the flag `-h` indicates that only the names of hidden files in the specified directory are to be printed.
  - `list`, `list -i`, and `list -h` respectively have the same usage as above, except that they are to be executed on the current working directory, instead of a specified path.

- `create`: This command is used for creating new files, directories, and links. Like the list program, create is a separate program that can be executed by simpleshell, or can be executed manually from bash by running its executable file. The create program should support the following usage:
  - `create -f filepath` creates an empty, ordinary file whose name is given in the specified path. The path can be an absolute pathname, a relative pathname, or just a bare filename, in which case the file should be created in the current directory. The new file should have permission 0640 specified in octal.
  - `create -d dirpath` creates a new directory whose name is given in the specified path. The new file should have permission 0750 specified in octal.
  - `create -h oldname linkname` creates a `hard` link. oldname is the path or name of an existing file, and linkname is the path or name of the hard link to be created.
  - `create -s oldname linkname` creates a `symbolic` (i.e., soft) link. oldname is the path or name of an existing file, and linkname is the path or name of the symbolic link to be created.

- `wd`: When given this command, simpleshell should print the absolute pathname of its current working directory. When simpleshell is first executed, its current working directory is the directory from which it was executed. This can later be changed by the `chwd` command.
- `chwd`: When given the command `chwd pathname`, simpleshell should change its current working directory to the specified path.
- `fileconverter`: This command is used for converting a text file into a binary file and vice versa. The fileconverter program should support the following usage:
  - `fileconverter flag infile outfile`: Parameters infile and outfile represent the names of the input and output files respectively. Parameter flag may be either -t or -b. When flag is -t, the input file is a text file and the output file must be the binary file containing the representation for each line as described above. Similarly, when flag is -b, the input file is a binary file containing the representations of the lines of a text file, and the output file must be the corresponding text file.

    You may assume that text files provided to fileconverter contain information about students at an imaginary university. Each line of a text file contains information about a single student, and will be in the following format: `firstname lastname id gpa`, where the first and last names are strings of at most 255 characters, the id is a 4–byte integer, and GPA is a 4–byte floating point number with precision of one decimal place (e.g., 3.5). The four items are separated by single whitespace characters.

    A binary file will contain one record for each line in its corresponding text file. The size of a record is $l_1 + l_2 + 10$ bytes, where l_1 and l_2 are the length of the first and last name strings respectively. The 10 remaining bytes consist of 1 byte to store l_1, 1 byte to store

l_2, 4 bytes to store the id number, and 4 bytes to store the GPA. The format of a record is l_1 `firstname` l_2 `lastname id gpa`.

  - `fileconverter -s infile`: Parameter infile represents the name of the binary input file which was created previously from a text file. The only flag that is permitted in this form is -s. Here, your program must process the input (binary) file and print the following information to stdout (i.e., should not produce an output file): (i)the full name of the student with the greatest combined name length (i.e., the student for whom l_1+l_2 is greater than that of each other student), (ii) the full name of the student with the least combined name length (i.e. the student for whom $l_1 + l_2$ is less that of each other student, (iii) the value of the highest id number, (iv) the value of the lowest id number, (v) the value of the highest GPA, and (vi) the value of the lowest GPA.

- `quit`: When this command is encountered, simpleshell should stop processing commands, print "goodbye", and exit.

Your simpleshell progtam should also allow output to be `redirected` to a file (e.g., executing the command `list -i > foo` should execute `list` in the current directory and print its output in the specified file).

Error Handling: For cases not covered by this specification, you may specify and implement a reasonable behavior. Additionally, your program must detect the following fatal errors.

- A wrong number of command line arguments is provided. In this case, your simpleshell program should print an error message and stop.
- The script file provided could not be opened. In this case, your simpleshell program should print an error message and stop.
- A pathname given to the list program could not be opened. In this case, the list program should print an error message including the reason for failure (e.g., doesn't exist, not a directory) and exit. Note that your simpleshell program should **not** exit in this case but instead continue executing commands. The only thing your simpleshell program should be aware of is whether list encountered an error or not.
- Wrong number of arguments given to the list program. The list program should print an error message and stop. However, your simpleshell program should not exit, but instead continue executing commands.
- A pathname given to the create program could not be opened. In this case, the create program should print an error message including the reason for failure (e.g., doesn't exist, not a directory) and exit. Note that your simpleshell program should **not** exit in this case but instead continue executing commands. The only thing your simpleshell program should be aware of is whether create encountered an error or not.
- Wrong number of arguments given to the create program. The create program should print an error message and stop. However, your simpleshell program should not exit, but instead continue executing commands.
- Wrong number of arguments given to the fileconverter program. The fileconverter program should print an error message and stop. However, your simpleshell program should not exit, but instead continue executing commands.

- The specified input or output file provided to fileconverter cannot be opened or the flag specified is not one of -t, -b or -s. In this case, the fileconverter program should print an error message including the reason for failure and exit. Note that your simpleshell program should **not** exit in this case but instead continue executing commands.
- The chwd command is called without specifying a directory or with an invalid pathname given. In this case, your simpleshell program should print an error message and continue executing commands.
- An invalid command is given to your simpleshell. In this case, simpleshell should print an error message and then continue processing commands.

Remarks:

- You may not utilize the `system()` function to complete any of the tasks.
- You may not have your simpleshell program execute any existing bash utility programs (e.g., ls, ln, mkdir). It must utilize system calls to execute either other programs. In turn, these programs must issue system calls directly (i.e., they too cannot use the `system()` function or execute any of the existing bash programs).
- The fileconverter program is **optional** and will be considered for an **extra credit** of 2% (i.e., the equivalent of Programming Assignment 1).
- The input/output redirection is also **optional** and will be considered for an additional **extra credit** of 1%.
- When your program is compiled and linked, it should produce **three** (3) executable files, namely, simpleshell, list, and create (similarly four executable files including the fileconverter program if you opt to submit that as well). The simpleshell program should **create child processes** to execute these commands, and wait for these programs to exit before continuing.
- The `list`, `create`, and `fileconverter` programs should accept their arguments via the `argv` parameter.
- For your fileconverter program, you may assume that (i) each text file will have at least one line, (ii) each string in a text file will have at least 1 and at most 255 characters, (iii) no string will contain white space characters, (iii) each integer and float in a text file will be non–negative and can be stored in 4 bytes, and (iv) each binary file will be non–empty; it will contain the correct representation of each line of the corresponding text file.
- Your shell program should execute each of the `list`, `create`, and `fileconverter` programs by creating a child process and having that child process execute the command's executable, while the parent (the shell)process waits for it to finish.
- If any part of your programs fails to compile, you will receive a zero on the entire assignment.

C. Structural Requirements. Your submission must *at least* contain the following files:

1. A C source file named `simpleshell.c` with just the `main` function for your shell program.
2. A C source file named `list.c` with just the `main` function for the list program.
3. A C source file named `create.c` with just the `main` function for the list program.

4. A C source file named `fileconverter.c` with just the `main` function for the fileconverter program.
5. A C source file named `shellFunctions.c`, which contains functions for handling the commands (i.e., wd, chwd, and quit) that the shell program must perform.
6. A C source file named `input.c` with the function `char* getLine(FILE* stream)`. This function reads one line from the specified file and returns the result as a null terminated string. Note that this can be used to read from stdin by calling it as getLine(stdin). This source file can additionally contain helper functions.

C. Submission Instructions. Make sure you are logged in to ITSUnix, and your present working directory contains only the files you wish to submit. Use the `turnin-csi402` command to submit all of the .c and .h files, and any file named 'makefile'. Make sure there are no extra .c or .h files in the current directory. If your makefile is called 'Makefile' (with a capital M), then make sure you use a capital M in the turnin command. Instructions for using the `turnin-csi402` command have been provided in Programming Assignment 0, which you should have already submitted.