CS 377
Assignment 4
31/Jan/2014

Purpose
Understand piping, concurrency and redirection and implement them on the *jash* shell.
NOTE: Please read the complete assignment carefully and follow the submission instructions carefully, any discrepancies from the given submission instructions may be penalized.

Task
As stated before, you are expected to have completed the first part of the assignment (parts of the last assignment are highlighted in red)

1. **Interactive Operation** - After startup, *jash* should perform the following actions in a loop (described in detail in the following sections) :-
   1) Print a prompt consisting of a `$' sign followed by a space.
   2) Read a line from standard input.
   3) Lexically analyse the line to form an array of tokens.
   4) Syntactically analyse (i.e. parse) the token to form a command.
   5) Execute the command.

   To begin with you are provided with a file named 'jash.c' which does the first 3 parts for you. Use it as your starting point.

2. **Lexical Analysis** - The program breaks a single line taken from standard input into tokens. A token is simply a single word which does not need further breaking down. Usually, a token is a sequence of non-whitespace characters separated from other sequences by whitespace.

3. **Syntactic Analysis** - A command is a sequence of tokens, the first of which specifies the command name. The tokens following the command are to be passed to the executable. The fish should have the following built-in tokens: `<', `>', `>>', `&' and `|'. The `|' token separates two commands. The command is to be executed as per details given in the next section.
   i. The `<' token indicates that the STDIN stream should be redirected from a user specified file. Display an error message if this file does not exist or the redirection is done more than once, and discard the entire input line.
   ii. The `>' token indicates that the STDOUT stream should be redirected to a user specified file. If the file does not already exist, then fish should create it; if the file does already exist, then fish should destroy its contents and rewrite the file from scratch. Display an error if this token occurs more than once in a command.
   iii. The `>>' operator indicates that the STDOUT stream be redirected in APPEND mode. Display an error if this token occurs with `>' or occurs more than once in a command.
   iv. The `&' operator indicates that the program needs to be run in background. More details in the next section under Concurrent Executions.
   v. The `|' operator comes in between two separate commands. More details in the next section under piped executions.

4. **Execution** - The command can be either a file name which needs to be executed or a built-in *jash* command.
   i. **Built-in *jash* commands**

Implement the commands '**cd**' (change directory), '**prev**', '**run**', '**exit**' and '**children**' in the shell itself.

- '**cd**' takes one argument which is the absolute or relative path of the directory you wish to switch to. Display an error message if the directory does not exist.
- `**prev**' command which takes a string (*search_string*) as argument, and returns the latest command executed on the shell which contains *search_string* as a substring. For example, the following are valid search strings for the command "cd ~/Desktop":
  - ✔ "cd ~/Desktop"
  - ✔ "cd ~"
  - ✔ cd *(without quotes)*
  - ✔ Des

  You need not maintain the history in a file but do assume a reasonable number of commands in a session if you're storing it in memory.

- `**run**' takes one argument which would be a batch file name containing a list of commands separated by newline. Your shell program should execute them one by one and if one of them executes erroneously you should print the `error' and abort the batch without exiting the shell.
- '**exit**' command does not take any argument and exits the *jash* program.

ii. File executions

- You have to search for the file name in the current directory and the directories mentioned in the PATH variable inherited from the parent shell. Look for execv variants that can do this. Display an error message if the file is not present in any of the directories, and discard the entire input line.
- All the tokens following the file name are to be passed to the executable file.
- All the tokens following the file name are to be passed to the executable file.
- The child processes forked by *jash* could be running either in the foreground or in the background. The user should be able to kill the current child process by sending it a SIGINT (Ctrl-C) signal. SIGINT should not kill *jash* itself.

iii. **I/O redirection –** You need to redirect input/output streams from/to files as is done on Bash using appropriate symbols as mentioned in the syntactic analysis.

iv. **Concurrent Executions -** When a file execution command contains the `&' operator, you have to run the command in the background. The I/O redirections should be as per the other operators. *jash* should not wait for the command to finish and should immediately resume its interactive loop. *jash* must kill all child processes before exiting. When a process running in the background ends, you should display a message along with its pid before the next prompt display.

v. **Piped Executions** - This mode requires two `file execution' commands to be supplied. Both of these commands are to be executed concurrently with the STDOUT stream of the first command piped to the STDIN stream of the second command. Assume that this operator will be present only once in an entire input line, i.e. you do NOT have to handle multiple pipes like ` cmd1 | cmd2 | cmd3 | .. '. Assume that this operator will not co-exist with `&' operator in an input line. You can also assume that the piped streams are not redirected by I/O redirection operators. The shell's input and output streams are held by the first and

second programs, respectively, if no redirection was requested. Assume that jash built-in commands will occur in isolation, i.e. you should display an error if they are used in pipelines. Though the batch file provided to `run' command may contain piped executions.

5. **Exit** - *jash* should exit on receiving the Ctrl-D (EOF) character as input or the `exit' command. *jash* must kill all child processes before exiting.

Coding Directives

1. Follow the structure mentioned above to write *jash*. Each section should be implemented as a function, which may call on any number of helper functions you define. You can overload a function for a section if you want to pass a different number of arguments for different cases.
2. *jash* should handle an erroneous line gracefully by rejecting the line and writing a descriptive error message to the standard error stream (STDERR)
3. *jash* should contain no memory leaks. For every call of malloc(), eventually there should be a call of free().
4. Assume that no standard input line contains more than 1000 characters, the newline character included.
5. Document! Each function definition should have a comment stub above it to describe what it does. As a healthy programming practice, you should have ample number of in-line comments inside function definitions to guide someone who is reading your code for the first time.
6. Your code can be in any number of files. Make sure you provide a makefile which compiles your program into an executable called *jash*.
7. You can assume that the input/output redirections will be separated by white spaces i.e. your code will not be tested on separating the two tokens in `<file'.
8. *jash* need not support file redirection with its built-in commands.

Recommendations

1. Use a variant of the execv command to supply arguments and search for executable in the directories mentioned in the PATH variable. See the man page of execv for details.
2. Use the chdir UNIX system call to implement the `cd' command.
3. After printing anything to STDOUT, flush the output. In C, you can do it by fflush(stdout).
4. Once a command executes, your shell should wait for its activities to finish. You should be using the wait function to achieve this.
5. Read the error codes returned by various functions carefully, especially exec and its variants, malloc, fopen etc. and use them to handle and display any errors.
6. After printing to standard output, immediately flush the standard output stream by calling **fflush(stdout)**. Doing so eliminates complications with output buffering in the presence of concurrent processes.
7. Your shell should only wait for a command not executing concurrently. You should be using the wait function to achieve this.

Resources

Use man pages for description of system calls. Alternatively, you can use http://www.opengroup.org/onlinepubs/007908799/xsh/unistd.h.html

Submission Instructions

You must follow these instructions for automated grading by a script.

1. Your submission folder **must** contain a MAKEFILE which will compile your program into an

executable *jash*. The name of your folder MUST be your <rollno1>_<rollno2>_assign4

2. Your program must be able to run with gcc (not g++) and must not give warnings on compilation.
3. **[Optional] You can have a test folder which contains test cases which worked for your program. This will help us grade incomplete assignments.**
4. **README file** containing your details, implementation status, and anything you need to tell us. Actual documentation should be within the source code itself.
5. Tar your folder with
   tar -zcvf <rollno1>_<rollno2>_assign3.tar.gz <rollno1>_<rollno2>_assign4
6. Submit on Moodle before the deadline.