CS377 Assignment 3

Purpose

Understand UNIX processes, input/output streams of processes, and signals by constructing an interactive UNIX shell.

Background

A UNIX shell is a simple textual interface to the operating system. Most shells expose the file system to the user who can execute various utilities and commands present in the file system to interact with the operating system. There are several popular UNIX shells: sh (the Bourne shell), csh (the C shell), zsh (the Z shell), and bash (the Bourne Again shell) are a few.

Task

Create a program named *JASH* (Just Another Shell) which will be your first interactive UNIX shell. This shell will expose the file system to the user and the user can run executable files present in the file system. The shell will also control the input and output for these executables. The following sections guide you through the process of creating a shell. It is highly advised that you read and implement one section at a time. Also read the coding directives and recommendations before you begin.

NOTE:

First submission should have till section 5. Sections 6 and 7 have to done before second deadline. The exact time for these two deadlines will be announced separately.

Next lab will extend on this assignment. Therefore, It is important that this assignment be completed in order for you to expand it for the next lab assignment. Hence, even if you are unable to complete it in the given time frame, you must complete it before next lab to hope to do the next lab well.

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

   e. 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. [Note: This part is already implemented!]

3. **Syntactic Analysis** - A command is a sequence of tokens, the first of which specifies the command name. For this assignment you can assume that there would be only one command per line, hence the tokens following the command name are to be passed to the executable. The command name is to be executed as per details given in the next section.

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

   Built-in *jash* commands.
   Implement the commands 'cd' (change directory), 'run' in the shell itself.

   a. ‘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.
   b. ‘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.

5. **File executions**

   a. 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.
   b. All the tokens following the file name are to be passed to the executable file.
   c. All child processes forked by jash should run in the foreground; jash need not support background process control (i.e. commands ending in ampersand `&'). However, the user must be able to kill the current child process by sending it a SIGINT (Ctrl-C) signal. SIGINT should not kill jash itself.

6. **cron command**

Syntax:
cron <filename>.txt

Cron command is used to execute commands at predefined time. It takes a file as argument. The file contains the information about when a particular command is to be executed. When it is time for a command to be executed, JASH should create a new process and run the command.

The format of input file is as given below:

a. Each line denotes one command
b. Each line is of the form given below

```
* * * * * command to execute
┬ ┬ ┬ ┬ ┬
│ │ │ │ │
│ │ │ │ │
│ │ │ │ └───── day of week (0 - 6) (0 to 6 are Sunday to Saturday)
│ │ │ └────────── month (1 - 12)
│ │ └─────────────── day of month (1 - 31)
│ └──────────────────── hour (0 - 23)
└───────────────────────── min (0 - 59)
```

c. '*' in a particular field denotes that there is no restriction on that field. A value in that field means a restriction is there in that field. For the command to be executed at a particular time, all the fields should satisfy that time instant.

For example the file is as given below:

```
15 * * * * ls
0 0 1 * * pwd
```

'ls' command will be executed every hour at 15th minute.
'pwd' command will be executed at midnight on 1st of every month.

7. **parallel command**

Syntax:
parallel job_1 ::: job_2 ::: job_3 ::: …….. ::: job_n

GNU **parallel** is a shell tool for executing jobs in parallel using one or more computers. For this assignment, we consider only those commands as above. Each job in the given list of jobs (n>0) is to be executed parallely in child process (use fork()).

For example,
parallel ls ::: pwd ::: echo abc
then 3 child processes must be created for each of the jobs (ls, pwd and echo abc).

BONUS: (Attempt only if you have done the other questions.)

- Implement the command ‘**exit**’ – this command does not take any argument and exits the *jash* program. *Jash* should exit on receiving the Ctrl-D (EOF) character as input or the ‘exit’ command. It must kill all child processes before exiting.
- Ctrl-C should kill a child process, if there is one, else it should kill the parent process.

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.

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.

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 <rollnumber1>_<rollnumber2>
2. [Optional] You can have a test folder which contains test cases which worked for your program. This will help us grade incomplete assignments.
3. README file containing your details, implementation status, and anything you need to tell us. Actual documentation should be within the source code itself.
4. Tar your folder with
   tar -zcvf <rollnumber1>_<rollnumber2>.tar.gz <rollnumber1>_<rollnumber2>
5. Submit upto Section 5 in the first submission link and complete assignment in the second link before the respective deadlines on moodle.