Lab2- Implement a shell

What is a shell?

Shell is a UNIX term for the interactive user interface with an operating system. The shell is the layer of programming that understands and executes the commands a user enters. The shell is also called a command line interpreter. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute, or by creating text scripts of one or more such commands. This project is divided into three phases as explained below.

Phase 1

The most generic sense of the term *shell* means any program that users employ to type commands. A shell hides the details of the underlying operating system and manages the technical details of the operating system kernel interface, which is the lowest-level, or "inner-most" component of most operating systems.

A shell does three main things:

- Start up: After the system boots, the shell process starts up by reading and executing its configuration files.
- Scan: The shell scans the commands from standard input which could be direct input or in the form of a script and executes them.
- Stop: After execution, the shell performs some clean-up procedure, and terminates.

Out of the above 3 things, we are mostly concerned with scanning and would build our shell on that component. Now what does shell do as a part of scanning and executing is as follows:

1. Reading the input
2. Parsing the input to separate it into commands and arguments
3. Executing the commands from step 2.

Out of the above steps, how would a shell execute a program which is taken as input and then parsed. Below is an example of the world's simplest shell and the steps followed:

The World's simplest shell

```
int main(int argc, char **argv)
{
  char buf[1024]; pid_t pid;
  int status;

  while (getinput(buf, sizeof(buf)))
  { buf[strlen(buf- 1] = '¥0';
    if((pid=fork()) == -1) {
      fprintf(stderr, "shell: can't fork: %s¥n",  strerror(errno));
      continue;
    }
    else if (pid == 0) {
      /* child */
      execlp(buf, buf, (char *)0);
      fprintf(stderr, "shell: couldn't exec %s: %s¥n", buf, strerror(errno));
      exit(EX_DATAERR);
    }

    if ((pid=waitpid(pid, &status, 0)) < 0)
     fprintf(stderr,"shell: waitpid error: %s¥n",strerror(errno));
  }

  exit(EX_OK);
}
```

The steps followed by it:

1) Taking input (command) from user

2) Forking a child process

3) Calling exec to replace the child process with the input taken in the first step.

4) Waiting for the child process to finish execution

5) Exiting

Some more things to care of: When you execute a command you have to set up the environment variables for the shell to know where to take the values of variables eg PATH, CWD, SHELL.

For e.g. exec(/bin/ls, ls, NULL)

In this case environment variable:

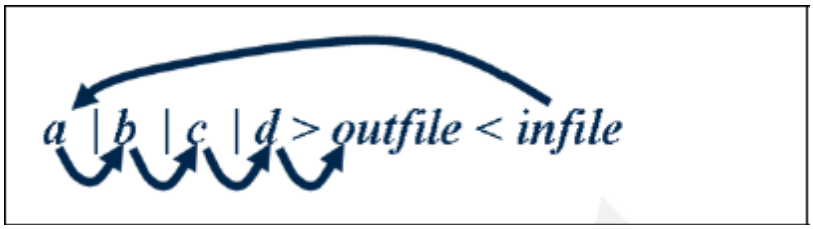
- PATH should specify the path for the command ls to run
- CWD should specify the current working directory for the ls command to list files and directories
- SHELL should specify which shell to execute

At this point, if we convert every step we have discussed above into code, we would get a very basic shell which would take input and execute the commands. And this is something which is readily available online. (I would encourage you to write your own version as this is something which would help develop your skills and can be added in your resume.)

Phase 2

In Phase 2, we would like to add more functionality to the basic shell. Here is a list of functions which your shell should perform:

1. **Piping and I/O redirection**: The strategy for your shell is to have the parent process do all the piping and redirection before forking the processes. In this way the children will inherit the redirection. The parent needs to save input/output and restore it at the end.



In this figure the process a sends the ouptut to pipe 1. Then b reads its input from pipe 1 and sends its output to pipe 2 and so on. The last command d reads its input from pipe 3 and send its output to outfile. The input to a comes from infile. You can implement this with the help of ***pipe*** function and ***dup*** system call.

2. **History feature**: You are supposed to implement history feature in your shell which is able to give last 25 commands used with there PID's. Here are the other features for the history command:
    - List last 25 commands used with date and timestamp
    - Search specific commands in history using grep

3. **Editor:** Implement a customized editor in which you can write your code and execute using the shell.

4. **Aliasing:** An alias is a (usually short) name that the shell translates into another (usually longer) name or command. Aliases allow you to define new commands by substituting a string for the first token of a simple command. They are typically placed in the ~/.bashrc (bash) or ~/.tcshrc (tcsh) startup files so that they are available to interactive subshells. Under bash the syntax of the alias builtin is
   alias [name[=value]]
   You have to build this feature for your shell.

Phase 3

Apart from the above features, I would like you to implement **2 custom functions**. These functions can be some new functionality which you think is not already there or an enhancement of the already available ones. Now for these functions I am keeping them open for you to think as of now and we can discuss the same in subsequent mails. Here are a few ideas for the same:

1. Implement a function let's say "sgown" which takes a keyword in the command line and searches for all the instances of this keyword in given directory, sub directory and files and display them.

$ sgown keyword

Displays the no of occurrences with line numbers in various files and directories.

2. Enhance the functionality of ls command: to create a ls command which sorts the output on time of last inode modification/ type of file/ UID's given different options.

3. Implement ps command for your shell

Now this is something where you can score marks for creativity and come up with some innovative ideas which are not already there.

We can divide the entire task into phases now:

Phase 1: Implementation of basic shell

Phase 2: Implementation of features defined above

Phase 3: Implement custom functions

You are also required to make a design document which documents your design ideas.

Dates of submission:

Phase 1: 19-24 March 2018

Phase 2: 2-7 April 2018

Phase 3: 23-28 April 2018

Note: You might understand that unlike last time this deadline is something which can not be extended because 28th April is your last working day.