**Abstract**

A Linux-Shell is the command line interpreter which allows user to direct the operations of the computer by entering command as text for a command line interpreter to execute it. The main objective behind the shell design in C will be so simple that there will not be any configuration file and there will not be any shutdown command. So, we will just call the looping function and then terminate. As a traditional Linux-Shell this shell will also contain few builtins. It will handle all the Linux commands in three basic steps those are read the command form standard inputs in the form of string, parse the command string into program and arguments and execute the parsed command.

**Introduction**

A shell is a program that provides the traditional, text-only user interface for Linux and other Unix like operating systems. So, this a walkthrough on how I designed my first simplistic Linux shell in C, in the hopes that it makes other people feel that way too.

Basically a shell does three main things in its life time. **Initialize:** a typical shell would read and execute its configuration files. **Interpret:** a shell read command from standard inputs and execute them. **Terminate:** after its commands are executed the shell executes any shut down command, frees up memory and terminate. These steps are so general that they could apply to many programs, but my shell will be so simple there will not be any configuration file and there will not be any shut-down command. So, we will just call the looping function and then terminate.

What does the shell do during its loop? Well, it handles command in three steps read the command form standard inputs in the form of string, parse the command string into program and arguments and execute the parsed command.

Now, we’re really at the heart of what a shell does. Starting processes is the main function of the shell. There are only two ways of starting processes on Unix. The first one is by being Init. Since most of the programs are not Init, that leaves only one practical way for processes to get started that is by using fork() system call.

We know that most of the commands a shell executes are programs, but not all of them. Some of them are built right into the shell. These commands are called shell builtins. So, it makes sense that we need to add some commands to the shell itself. The ones I have added to my shell are cd, exit and help.

The most missing piece of the puzzle is to put together builtins and processes. That’s all the code that goes into the shell. If you’ve read along, you should understand completely how the shell works. To try it out on a Linux based machine, you would need to copy code into a file(filename.c) and compile it.

**Tools and Requirements**

* **Software Requirements:**

1. Operating System: Any Unix based operating system.

Example: Ubuntu

1. Programming Language: C

* **Hardware Requirements:**

1. Processor: Intel
2. RAM: Min 1GB
3. Hard Disk: Min 80 GB

* **Tools:**

1. Text Editor: Any text editor for writing source code.
2. GitHub: GitHub is a web-based hosting service. It is used for source code management and version control.

**Analysis**

**Existing Shell:**

A shell does three main three things in its lifetime.

* Initialize: A typical shell would read and execute its configuration files.
* Interpret: The shell reads commands from standard inputs and execute them.
* Terminate: After its commands are executed the shell executes any shutdown commands, free up memory and terminates.

These steps are so general for any Linux shell.

**Proposed Shell:**

This shell will be so simple that there won’t be any configuration file and there won’t be any shutdown command. So, we’ll just call the looping function and then terminate. So, we’ve taken care of how the program should start up. Now, what does the shell do during its loop? Well, a simple way to handle commands is with three steps:

* Read: Read the command from standard input.
* Parse: Separate the command strings into a program and arguments.
* Execute: Run the parsed command.

We know that most of the commands a shell executes are programs, but not all of them. Some of them are built right into the shell. So, it makes sense that we need to add some commands to the shell itself. The ones I added to my shell are cd, exit, and help.

**Architectural Design**

The basic architecture of the shell is similar to pipeline, where input is analysed and parsed, symbols are expanded and finally commands are executed (using shell builtins and external commands).

Shell

User space

User

Command ls, cd, etc

Execution Builtins

Expansion

Lexical analysis and parse

Kernel space

Kernel

**Basic shell architecture**

**Modules**

Software is divided into separately named and addressable components called modules that are integrated to satisfy problem requirements. Modularity is the single attribute of software that allows a program to be intellectually manageable.

They are divided into four different modules.

1. **Read the command.**
2. **Parse the command.**
3. **Start the process.**
4. **Shell Built-ins and Execution.**
5. **Read the command:**

In this module, the shell can read the command from standard inputs. It uses a function within the (apparently infinite) while loop. In the loop, we read a character and store it as an int. If it’s newline or EOF, we null terminate our current string and return it. Otherwise we add character to our existing string. Next, we see whether the next character will go outside of our current buffer size. If so, we reallocate our buffer before continuing.

1. **Parse the command:**

It parses the line into list of arguments and programs. All we need to do is **“tokenize”** the string using whitespace as delimiters. We are using the same strategy of having a buffer and dynamically expanding it. But this time, we’re doing it with a null-terminated array of pointers instead of a null-terminated array of characters. Finally, we reallocate array of pointers if necessary. The process repeated until no token is returned and at this point we null-terminate the list of tokens. So, now we an array of tokens, ready to execute.

1. **Start the Processes:**

Starting processes is the main function of shells. There are only two ways of starting processes in Linux. The first one is by init. Since most of the programs aren’t init, that leaves only one practical way for processes to get started: the **fork()** system call. First, an existing process forks itself into two separate ones. Then, the child uses **exec()** to replace itself with a new program. The parent process can continue doing other things, and it can even keep tabs on its children, using system call **wait().**

1. **Shell Built-ins and Execution:**

You see, most commands a shell executes are programs, but not all of them. Some of them are built right into the shell. These commands could only change the shell’s operation if they were implemented within the shell itself. So, it makes sense that we need to add some commands to the shell itself. The ones I added to my shell are cd, exit, and help.

**Module Diagram**

1. **Read the command:**

Read the command

Lexical Analysis and Parsing

Input

1. **Parse the command:**

Read the command

Lexical Analysis and Parsing

Input

Argument

Program

Expansion

1. **Shell Builtins and Execution:**

Read the command

Lexical Analysis and Parsing

Input

Program

Expansion

Argument

Loop Termination

Command Execution

**Conclusion**

A Linux-shell is the command line interpreter which allows user to direct the operations of the computer by entering command as text for a command line interpreter to execute it. This shell has been written in C language because it doesn’t contain any configuration file and any shutdown command. We are just calling the looping function and then terminate. During its loop shell reads the command from standard input, parse the command into program and argument and run the parsed command. Starting processes is the main function of the shell. So, I have used fork() system call to start the process and exec() system call to run the process. Most commands a shell executes are programs but not all of them, some of them are built right into the shell. So, I have added some commands to the shell itself, those are called built-ins. The ones I added are cd, help, and exit. That’s all the code which goes into the shell and makes it so simple.

**References**

* <https://github.com/rathirakesh/Linux-Shell>
* <https://www.kernel.org/doc/Documentation/filesystems/proc.txt>
* <https://www.thegeekstuff.com/2011/11/linux-101-hacks-2nd-edition-download-free-ebook/>

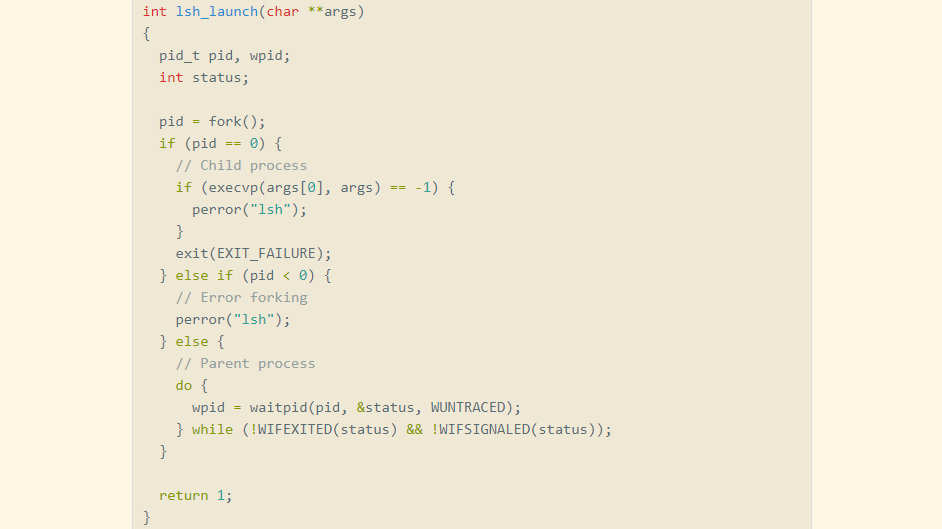
**Future Enhancement**

This shell has few limitations, including:

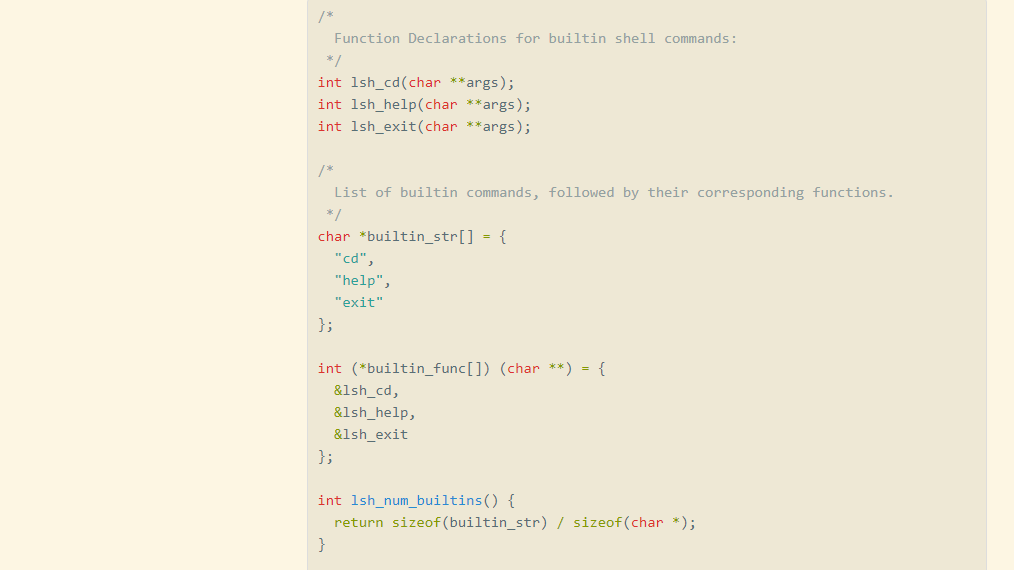
* Commands must be on a single line.
* Argument must be separated by whitespace.
* No quoting argument or escaping white space.
* No piping or redirection.
* Only built-ins are: cd, help, and exit.

All the above limitations can be overcome in future. Many more built-ins can be added like logout, break, date etc.

**Code snippet for starting processes:**

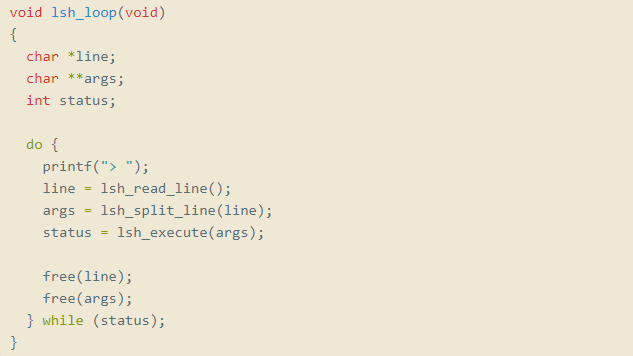


**Code snippet for built-ins:**



**Appendix**

**Code snippet for looping function:**



**Sample output:**

