CS250 - Operating Systems **Winter-2021-22**

Linux Shell in C

Name: Vemuganti Sesha Satvik **Roll No :** 12041710 **email:** vemugantisesha@iitbhilai.ac.in

Introduction

The submission folder consists of this report and 3 C-programme files. This report doesn't contain detailed explanation of the programmes. Such an explanation has been provided in extensive comments of the code files.

Execution Instructions :

- Parts A, B and C can be found in the files parta.c, partb.c and partc.c respectively.
- To compile, run either of the following:
  - gcc -lreadline <filename.c>
  - clang -lreadline <filename.c>
- And finally, to execute :
  - ./a.out
- Remember to compile each file before running the last command or risk executing the previously compiled file.

PART A

```
[satvikvemuganti@Satviks-MacBook-Pro] ~/Desktop/satvik  master  ⚡
❯ clang -lreadline parta.c

[satvikvemuganti@Satviks-MacBook-Pro] ~/Desktop/satvik  master  ⚡
❯ ./a.out
satvik_vemuganti@12041710 (^_^)$ pwd
/Users/satvikvemuganti/Desktop/satvik
satvik_vemuganti@12041710 (^_^)$
```

On compiling and running parta.c with the above commands, we see that a shell is initiated in the current working directory. In the above image, this has been demonstrated with the help of the pwd command, to print the current working directory.

Below is a demonstration of the working of a few basic commands :

```
satvik_vemuganti@12041710 (^_^)$ pwd
/Users/satvikvemuganti/Desktop/satvik
satvik_vemuganti@12041710 (^_^)$ ls
a.out   abc     adfadf. gg.c    nigga   parta.c partb.c partc.c
satvik_vemuganti@12041710 (^_^)$ ps
  PID TTY           TIME CMD
57857 ttys001    0:00.19 -zsh
44673 ttys002    0:01.10 -zsh
78269 ttys002    0:00.00 ./a.out
76216 ttys003    0:00.21 /bin/zsh -l
76230 ttys004    0:00.55 /bin/zsh -l
78057 ttys004    0:00.00 ./a.out
76277 ttys005    0:00.28 /bin/zsh -l
76309 ttys006    0:00.26 /bin/zsh -l
satvik_vemuganti@12041710 (^_^)$ echo THiS sHeLl is AwesOME !¡!
THiS sHeLl is AwesOME !¡!
satvik_vemuganti@12041710 (^_^)$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/satvikvemuganti/.ssh/id_rsa): from-my-shell
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in from-my-shell
Your public key has been saved in from-my-shell.pub
The key fingerprint is:
SHA256:Bejw62sfmjKvvIqddpuASJpCeB9EXQLnSw7YDrLoEOw satvikvemuganti@Satviks-MacBook-Pro.local
The key's randomart image is:
+---[RSA 3072]----+
|    ooooo        |
|.  +.oo. .       |
|o.o ++o   .      |
|++ + +o. .       |
|=E. o o.S        |
|B+ . ..          |
|*.. ..  .        |
|.o.++.oo .       |
|..+o*O=o.        |
+----[SHA256]-----+
```

Please note that the ssh-keygen command was previously installed and is displayed purely for demonstration purpose.

The shell is also able to execute commands with options (like ls -l, ps -a), as shown below.

```
satvik_vemuganti@12041710 (^_^)$ ps -a
  PID TTY           TIME CMD
57856 ttys001    0:00.07 login -pf satvikvemuganti
57857 ttys001    0:00.19 -zsh
44672 ttys002    0:00.02 login -fp satvikvemuganti
44673 ttys002    0:01.10 -zsh
78269 ttys002    0:00.01 ./a.out
78447 ttys002    0:00.00 ps -a
76216 ttys003    0:00.21 /bin/zsh -l
76230 ttys004    0:00.55 /bin/zsh -l
78057 ttys004    0:00.00 ./a.out
76277 ttys005    0:00.28 /bin/zsh -l
76309 ttys006    0:00.26 /bin/zsh -l
satvik_vemuganti@12041710 (^_^)$
```

```
satvik_vemuganti@12041710 (^_^)$ ls -al
total 128
drwxr-xr-x@ 11 satvikvemuganti  staff    352 Mar 14 12:09 .
drwx------@ 28 satvikvemuganti  staff    896 Mar 13 18:44 ..
drwxr-xr-x@ 15 satvikvemuganti  staff    480 Mar 13 21:59 .git
drwxr-xr-x@  3 satvikvemuganti  staff     96 Mar 14 00:22 .vscode
-rwxr-xr-x   1 satvikvemuganti  staff  50736 Mar 14 11:54 a.out
-rw-------   1 satvikvemuganti  staff   2635 Mar 14 12:04 from-my-shell
-rw-r--r--   1 satvikvemuganti  staff    595 Mar 14 12:04 from-my-shell.pub
-rw-r--r--   1 satvikvemuganti  staff   8838 Mar 13 18:44 gg.c
-rw-r--r--   1 satvikvemuganti  staff   4682 Mar 14 11:53 parta.c
-rw-r--r--   1 satvikvemuganti  staff   6105 Mar 14 11:28 partb.c
-rw-r--r--   1 satvikvemuganti  staff   7539 Mar 14 11:28 partc.c
satvik_vemuganti@12041710 (^_^)$
```

Changing directory with cd works as :

```
satvik_vemuganti@12041710 (^_^)$ cd a-new-directory
Directory changed to : /Users/satvikvemuganti/Desktop/satvik/a-new-directory
```

However, as of part A, the shell is unable to execute pipe commands and “&&” commands, as shown below.

```
satvik_vemuganti@12041710 (^_^)$ ls -al | grep \.c
ls: \.c: No such file or directory
ls: grep: No such file or directory
ls: |: No such file or directory
satvik_vemuganti@12041710 (^_^)$ ls -al && echo "&& ain't workin'"
ls: "&&: No such file or directory
ls: &&: No such file or directory
ls: ain't: No such file or directory
ls: echo: No such file or directory
ls: workin'": No such file or directory
satvik_vemuganti@12041710 (^_^)$ echo "&& ain't workin'"
"&& ain't workin'"
satvik_vemuganti@12041710 (^_^)$
```

These are solved in part B.

The shell exits elegantly with the exit command or the ctrl + C shortcut as shown below. The latter was implemented with the help of the <signal.h> library included in the header files list.

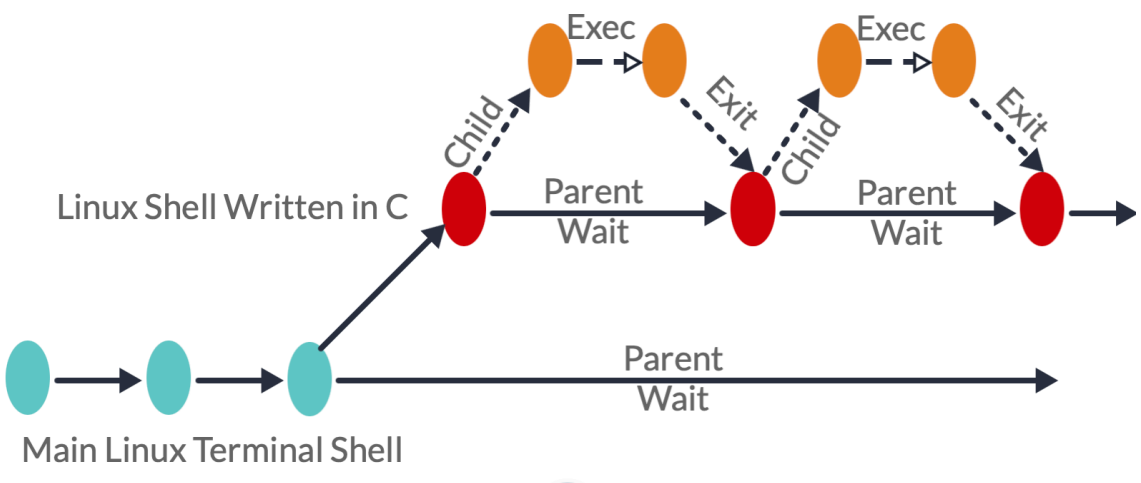
```
satvik_vemuganti@12041710 (^_^)$ exit

Exiting shell . . .

[satvikvemuganti@Satviks-MacBook-Pro] ~/Desktop/satvik  master  ⚡
❯
```

The following table describes the system calls that were important in implementing the linux shell using C. These are :

- Fork
- Wait
- Execute
- Exit

System Call	Description	Usage
Fork	Function causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process) with a different process ID.	In our shell, to execute a given command, we first fork our shell into a child process and execute the command in the child shell using exec[]( ).
Wait	Function suspends execution of its calling process until stat_loc information is available for a terminated child.	This is used to suspend execution of the parent process until the command is executed in the child process.
Exec	The exec family of functions replaces the current process image with a new process image. Replaces the process with another process with same PID. There are many variations.	The **execvp( )** variant of exec is used in the C programme to implement this linux shell. Essentially, the execvp system call is made from the child process in the fork of the current shell. This executes the command, replacing child shell process with the process of the invoked command.
Exit	exit functions terminate a process.	Implemented when exec has completed successfully and exits. It is also implemented for the exit command in our shell, which should terminate the C programme which is executing our linux shell. Also called by the **Ctrl + C** shortcut.

Here is a flow chart describing running of the written C shell in the inherent linux shell (bash or zsh being used in the terminal) as well as the execution of commands in the C shell for linux written for part A.



PART B

Following is a demonstration of the working of the **&&** operator in our linux shell programmed in **C**.

```
satvik_vemuganti@12041710 (^_^)$ ls -al && pwd
total 128
drwxr-xr-x@ 12 satvikvemuganti  staff    384 Mar 14 12:10 .
drwx------@ 29 satvikvemuganti  staff    928 Mar 13 18:44 ..
drwxr-xr-x@ 15 satvikvemuganti  staff    480 Mar 13 21:59 .git
drwxr-xr-x@  3 satvikvemuganti  staff     96 Mar 14 00:22 .vscode
drwxr-xr-x@  2 satvikvemuganti  staff     64 Mar 14 12:10 a-new-directory
-rwxr-xr-x   1 satvikvemuganti  staff  50464 Mar 14 13:35 a.out
-rw-------   1 satvikvemuganti  staff   2635 Mar 14 12:04 from-my-shell
-rw-r--r--   1 satvikvemuganti  staff    595 Mar 14 12:04 from-my-shell.pub
-rw-r--r--   1 satvikvemuganti  staff   8838 Mar 13 18:44 gg.c
-rw-r--r--   1 satvikvemuganti  staff   4682 Mar 14 12:31 parta.c
-rw-r--r--@  1 satvikvemuganti  staff   4423 Mar 14 13:34 partb.c
-rw-r--r--   1 satvikvemuganti  staff   7539 Mar 14 11:28 partc.c
/Users/satvikvemuganti/Desktop/satvik
satvik_vemuganti@12041710 (^_^)$
```

We can notice that the pwd command is executed only after the successful execution of the first command, which is ls -al, and fails otherwise, printing a line break (“\n”) to standard error.

```
satvik_vemuganti@12041710 (^_^)$ cd abc && pwd
/usr/bin/cd: line 4: cd: abc: No such file or directory

satvik_vemuganti@12041710 (^_^)$
```

Piping has also been implemented as required in part B.

```
satvik_vemuganti@12041710 (^_^)$ ls -al | grep \.c
-rw-r--r--   1 satvikvemuganti  staff   8838 Mar 13 18:44 gg.c
-rw-r--r--   1 satvikvemuganti  staff   4682 Mar 14 12:31 parta.c
-rw-r--r--@  1 satvikvemuganti  staff   6105 Mar 14 13:47 partb.c
-rw-r--r--   1 satvikvemuganti  staff   7539 Mar 14 11:28 partc.c
satvik_vemuganti@12041710 (^_^)$ ls -al | grep \.c | grep a\.c
-rw-r--r--   1 satvikvemuganti  staff   4682 Mar 14 12:31 parta.c
satvik_vemuganti@12041710 (^_^)$
```

W.R.T partb.c :
The simple tokeniser creates tokens of the input buffer using whitespaces [“ “] as a delimiter. The pipe_tokeniser does the same but with pipe operator [“|”] as a delimiter.
In the flow of the programme, the input buffer is first tokenised with the pipe delimiter, if exist, before execution of the individual tokens. The whitespace-tokeniser is part of the execution process that follows the pipe tokeniser.

The above is an overview abstraction of the control-flow of the program in implementing a pipe operator. Please refer to the code (and comments) for explanation of the implementation at the code level abstraction.

PART C

Demonstration of the implemented redirection feature of command outputs to files in our shell is shown in the screenshots below :

```
satvik_vemuganti@12041710 (^_^)$ ls -al
total 128
drwxr-xr-x@ 12 satvikvemuganti  staff    384 Mar 14 12:10 .
drwx------@ 31 satvikvemuganti  staff    992 Mar 13 18:44 ..
drwxr-xr-x@ 15 satvikvemuganti  staff    480 Mar 13 21:59 .git
drwxr-xr-x@  3 satvikvemuganti  staff     96 Mar 14 00:22 .vscode
drwxr-xr-x@  2 satvikvemuganti  staff     64 Mar 14 12:10 a-new-directory
-rwxr-xr-x   1 satvikvemuganti  staff  51520 Mar 14 13:59 a.out
-rw-------   1 satvikvemuganti  staff   2635 Mar 14 12:04 from-my-shell
-rw-r--r--   1 satvikvemuganti  staff    595 Mar 14 12:04 from-my-shell.pub
-rw-r--r--   1 satvikvemuganti  staff   8838 Mar 13 18:44 gg.c
-rw-r--r--   1 satvikvemuganti  staff   4682 Mar 14 12:31 parta.c
-rw-r--r--@  1 satvikvemuganti  staff   6105 Mar 14 13:47 partb.c
-rw-r--r--   1 satvikvemuganti  staff   7578 Mar 14 13:59 partc.c
satvik_vemuganti@12041710 (^_^)$ ls -al > list.txt
satvik_vemuganti@12041710 (^_^)$ cat list.txt
total 128
drwxr-xr-x@ 13 satvikvemuganti  staff    416 Mar 14 14:02 .
drwx------@ 31 satvikvemuganti  staff    992 Mar 13 18:44 ..
drwxr-xr-x@ 15 satvikvemuganti  staff    480 Mar 13 21:59 .git
drwxr-xr-x@  3 satvikvemuganti  staff     96 Mar 14 00:22 .vscode
drwxr-xr-x@  2 satvikvemuganti  staff     64 Mar 14 12:10 a-new-directory
-rwxr-xr-x   1 satvikvemuganti  staff  51520 Mar 14 13:59 a.out
-rw-------   1 satvikvemuganti  staff   2635 Mar 14 12:04 from-my-shell
-rw-r--r--   1 satvikvemuganti  staff    595 Mar 14 12:04 from-my-shell.pub
-rw-r--r--   1 satvikvemuganti  staff   8838 Mar 13 18:44 gg.c
-rw-r--r--   1 satvikvemuganti  staff      0 Mar 14 14:02 list.txt
-rw-r--r--   1 satvikvemuganti  staff   4682 Mar 14 12:31 parta.c
-rw-r--r--@  1 satvikvemuganti  staff   6105 Mar 14 13:47 partb.c
-rw-r--r--   1 satvikvemuganti  staff   7578 Mar 14 13:59 partc.c
satvik_vemuganti@12041710 (^_^)$ ls
a-new-directory         from-my-shell           gg.c                    parta.c                 partc.c
a.out                   from-my-shell.pub       list.txt                partb.c
satvik_vemuganti@12041710 (^_^)$
```

The file list.txt contains the contents of the redirection. It is highlighted in blue text.

The shell supports only the ‘>’ redirection operator. Syntax is as demonstrated in the screenshot above, command > filename.extension .

In our program, there is a skipwhite function that returns a temporary state. This temporary state holds the output contents of the command. The C-program uses malloc to expand and allocate memory to temp state, which is basically the file to which the content has to be redirected.
This way, redirection is implemented successfully.
A limitation of my program is that the operator “>” and the operator “>>” give the same output and completely write a new file even if there’s an existing file of the same name, in both the cases.
In the linux shell, “>” is analogous to overwriting and “>>” is analogous to appending to the file.

Keeping this flaw aside, the shell functions well.

Added a bit of design that shows on start of the shell :