PROJECT 1: DEVELOP YOUR OWN SHELL

Design Document

CS551: Operating Systems Design and Implementation

Anirudh Sunkineni(asunkine@hawk.iit.edu),

Lakshmi Karle(lkarle@hawk.iit.edu),

Vivek Pabani(vpabani@hawk.iit.edu)

Illinois Institute of Technology

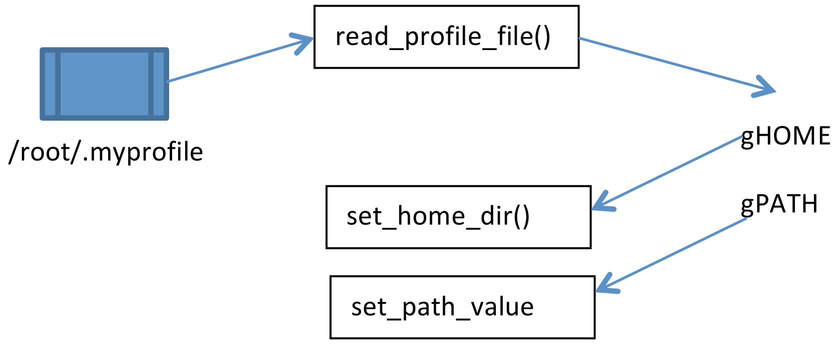
Contents

1. Read Profile File

1.1 Requirements

1. Your shell shall first execute a PROFILE file which defines the PATH variable that will allow you to access programs provided in /bin and /usr/bin. Once the PROFILE file is executed, you will be in a HOME directory specified by you in the PROFILE file. The PATH and HOME variables do not replace those of the Ash shell from which your shell is instantiated.
2. In a command line of your shell you will be able to exercise any executable programs including the utilities provided in /bin and /usr/bin.

1.2 Design



i. **read_profile_file()** function reads .profile file and populates global variables gHOME and gPATH
    i. Ignores comment lines
    ii. Takes the value of HOME and PATH when they are followed by "export"
    iii. Takes the value of HOME and PATH when they are not followed by "export"
    iv. Validates that there should be no space before and after to = sign in .myprofile file when assigning environment variables

ii. **set_home_dir()** gets the value of HOME environment variable as in .myprofile file from gHOME and sets the default directory of myshell to HOME directory
    i. Prints a warning of gHOME is not set
       Warning: HOME not SET
    ii. Changes current directory to gHOME using chdir() API with the following message
       Changed home directory to %s
    iii. Provides error message of chdir() API returns failure
       Error while setting home directory to %s errno:%s PAT
       H_MAX=%d

iii. **set_path_value()** gets the value of PATH environment variable as in .myprofile file from gPATH and sets the value of PATH environment variable to this value
    i. Uses setenv() API to perform this and prints following message on Success

Setting PATH variable to %s successful

ii. Prints Warning if API execution fails

Warning: Error setting PATH value

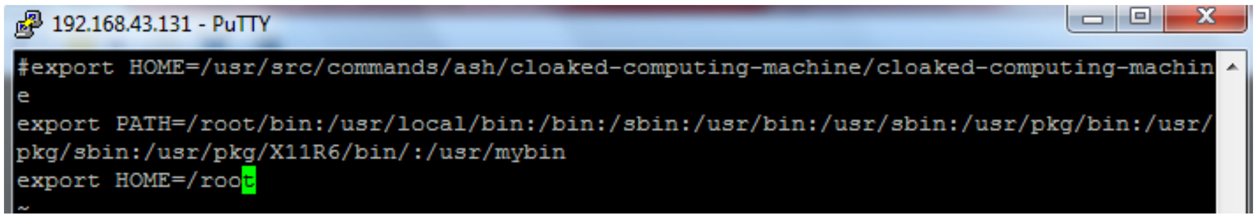
1.3 Test Cases

	Description	Input	Expected output	Result
1	Check if default home directory is set to what is specified in the .myprofile	Export HOME=/root	When HOME shell is invoked, pwd should yield /root	PASS
2	Check if myshell displays error message when default home directory is set to wrong value	Export HOME=/ro	An error message shall be displayed	PASS
3	Check if myshell displays error message when HOME env variable is not set in .profile file	HOME is not set here	An error message shall be displayed	PASS
4	Check myshell displays 'HOME not set' warning messages for all the cases where syntax to set HOME variable in .myprifile file is wrong	Cases: 1.export HOMEPATH=bin:\usr\bin:\usr\sbin 2. export HOME = "bin:\usr\bin:\usr\sbin" 3. export HOME = 4. export HOME = 5. export HOME 6. export HOME 7. export HOM	Warning message 'HOME not SET' shall be displayed	PASS
5	To check that comment lines are ignored	#export HOME=/root	Warning message 'HOME not SET' shall be displayed	PASS
6	Check if all the commands specified in the PATH as in the .myprofile can be executed	Export PATH=bin:\usr\bin:\usr\sbin	It shall be possible to execute ls. But should not be possible to execute myls copied	PASS

			in /usr/mybin	
		Export PATH=bin:\usr\bin:\usr\sbin:\usr\mybin	Now myls is also successful	PASS
7	Check if myshell displays ‘command not found’ error when PATH directory is set to wrong value	Export HOME=/ro	An error message ‘command not found’ shall be displayed	PASS
8	Check if myshell displays ‘command not found’ error when PATH directory is not set	PATH is not set in ./myprofile	An error message ‘command not found’ shall be displayed	PASS
9	Check if myshell displays ‘command not found’ error for all the cases where syntax to set PATH variable in .myprifile file is wrong	Cases: 1. export PATH = "bin:\usr\bin:\usr\sbin" 2.export PATH = 3.export PATH = 4.export PATH 5.export PATH	An error message ‘command not found’ shall be displayed	PASS

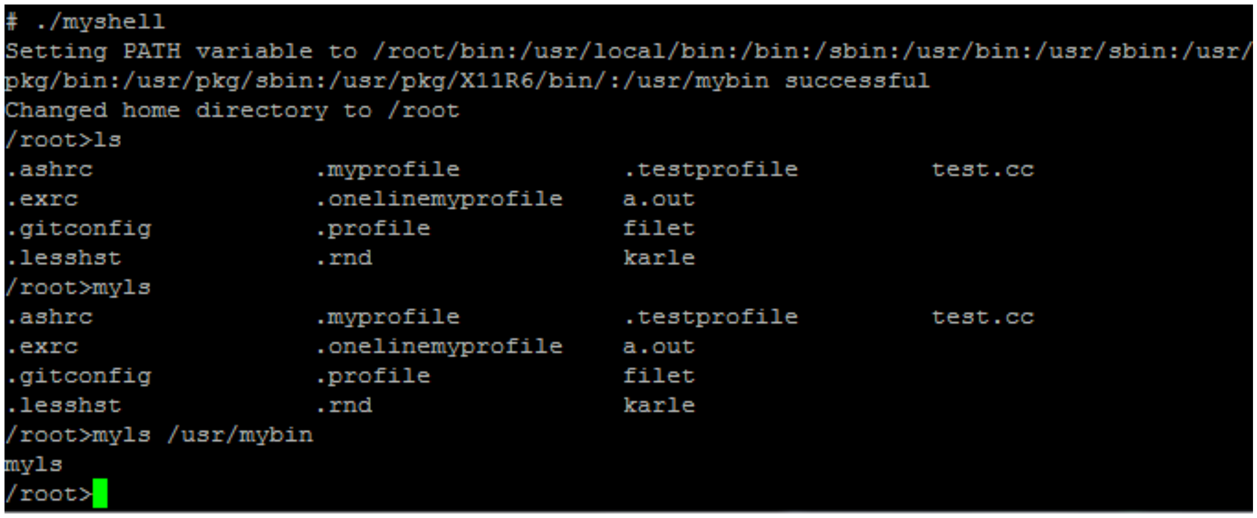
1.4 Sample output

Snapshot of .profile file



```
#export HOME=/usr/src/commands/ash/cloaked-computing-machine/cloaked-computing-machin
e
export PATH=/root/bin:/usr/local/bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/pkg/bin:/usr/
pkg/sbin:/usr/pkg/X11R6/bin/:/usr/mybin
export HOME=/root
```

Snapshot of myshell:



```
# ./myshell
Setting PATH variable to /root/bin:/usr/local/bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/
pkg/bin:/usr/pkg/sbin:/usr/pkg/X11R6/bin/:/usr/mybin successful
Changed home directory to /root
/root>ls
.ashrc                 .myprofile            .testprofile         test.cc
.exrc                  .onelinemyprofile     a.out
.gitconfig             .profile              filet
.lesshst               .rnd                  karle
/root>myls
.ashrc                 .myprofile            .testprofile         test.cc
.exrc                  .onelinemyprofile     a.out
.gitconfig             .profile              filet
.lesshst               .rnd                  karle
/root>myls /usr/mybin
myls
/root>
```

2. Set Prompt

2.1 Requirement

Your prompt shall display the current directory name.

2.2 Design

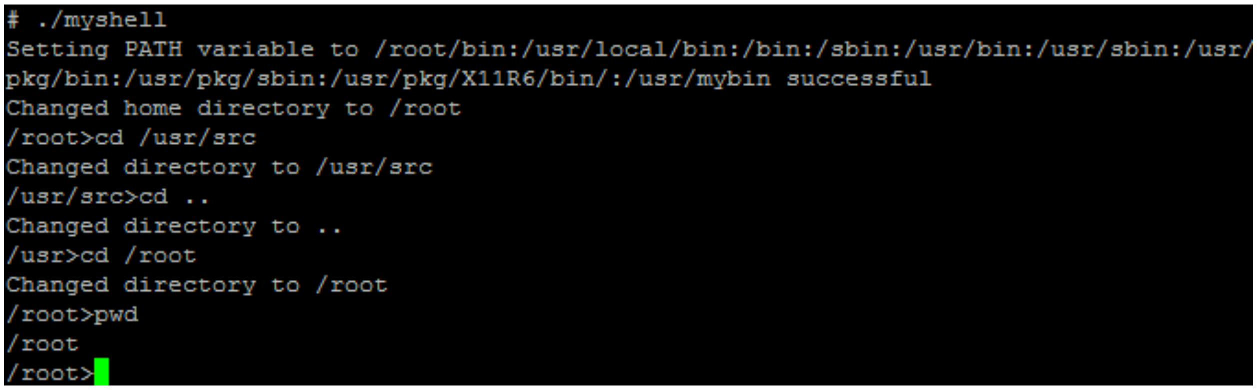
iv. From the infinite while loop of shell **emit_my_prompt()** is invoked before taking user input i.e. shell commands

v. emit_my_prompt() retrieves current directory using getcwd() API and sets it as prompt

2.3 Test Cases

	Description	Input	Expected output	Result
1	Login to shell and check prompt has HOME directory as in .profile file	Export HOME=/root	Prompt has HOME directory	PASS
2	Execute 'cd ..'	Execute 'cd ..'	Should go to parent directory and prompt should change to parent directory	PASS
3	Execute 'cd /root'	Execute 'cd /root'	'/root>' prompt shall be displayed	PASS
4	Execute 'cd /usr/src'	Execute 'cd /usr/src'	'/usr/src>' prompt shall be displayed	PASS

2.4 Sample output



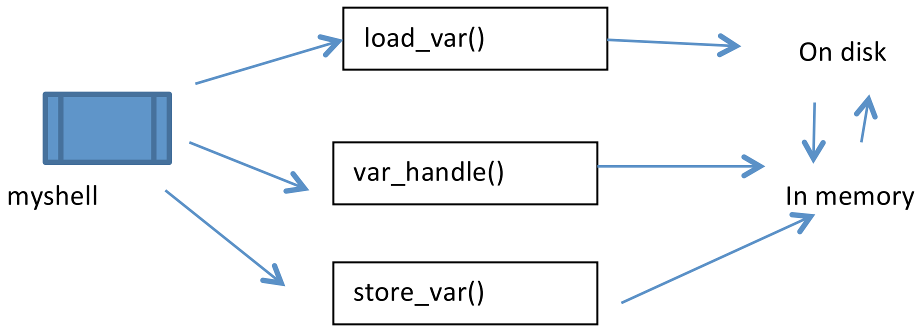
```
# ./myshell
Setting PATH variable to /root/bin:/usr/local/bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/
pkg/bin:/usr/pkg/sbin:/usr/pkg/X11R6/bin/:/usr/mybin successful
Changed home directory to /root
/root>cd /usr/src
Changed directory to /usr/src
/usr/src>cd ..
Changed directory to ..
/usr>cd /root
Changed directory to /root
/root>pwd
/root
/root>
```

3. Persistent Variables

3.1 Requirements

1. Your shell shall support integer variables (such as X whose value is $X).
2. The values of the variables will persist after your shell exits.

3.2 Design



Syntax : int <var_name> = <var_value> (space separated)

a. **load_var()** function gets called when the shell starts. It loads the variables from disk file to the memory.
   i. Check if the variable file exists.
   ii. If file found, read the variables into linked list.
   iii. If file not, just return with empty list.

b. **var_handle()** functions gets called when user input starts with ‘int’.
   i. Check if the syntax is correct. Generate error message otherwise.
   ii. Parse the command line and extract variable name and variable value.
   iii. Search in the list for the variable name. If already exist, update the value.

   iv. If variable does not exits, create a new node with variable data, and add to the list.

c. **store_var()** gets called when the shell exits.
   i. Check if the variable file exists. If not, create the file.
   ii. Iterate through the list, and write each variable in the file in the format:
   <var name> <var_value>

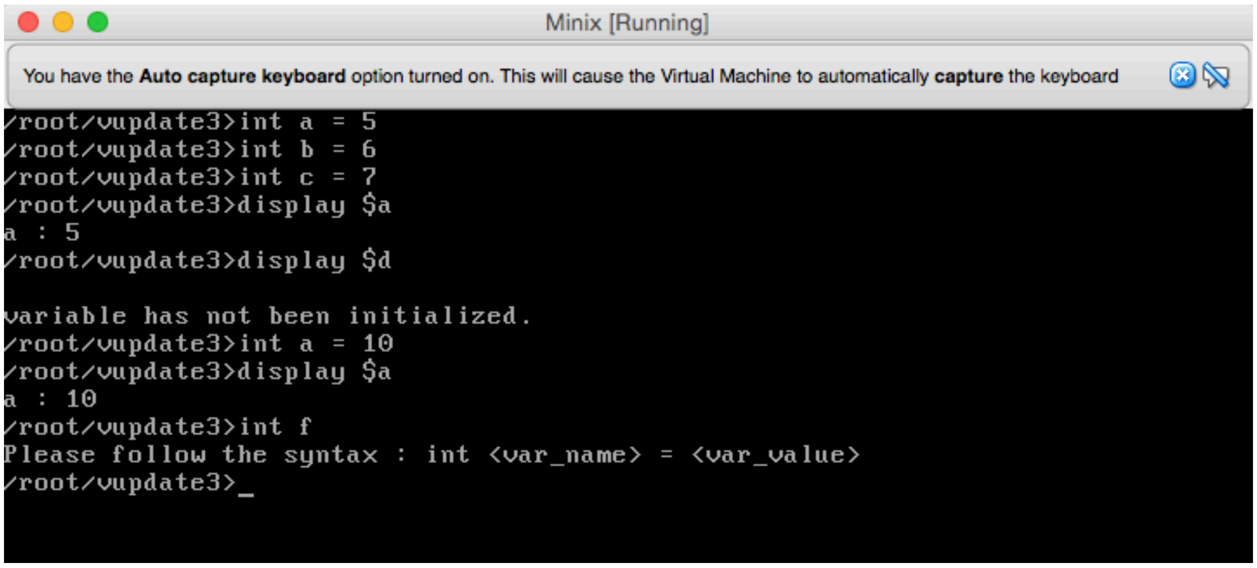
d. **display_var()** gets called when the shell exits. **Syntax : display $<var_name> (space separated)**
   i. Check if the variable exists in the list.
   ii. If exists, display the value. If not, generate error message.

3.3 Test Cases

	Description	Input	Expected output	Result
1	Check if variable file already exists in the path	./myshell	The list gets initialized with the variables in the file.	PASS
2	Check if variable file does not exist in the path.	./myshell	The list is initialized with no variables.	PASS
3	Check when a new variable is defined.	Int abc = 5	The variable gets added to the existing list.	PASS
4	Check when an old variable is redefined.	Int abc = 10	The variable gets updated to the existing list.	PASS
5	Check when syntax is wrong.	Case: 1. int xyz 2. int x=5 (no space) 3. int x 5	Warning message 'Please follow the syntax'	PASS
6	Check when non numeric value is provided.	Int z = fg	Warning message 'Please provide numeric value'	PASS

3.4 Sample output

Snapshot of myshell:



You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

```
/root/vupdate3>int a = 5
/root/vupdate3>int b = 6
/root/vupdate3>int c = 7
/root/vupdate3>display $a
a : 5
/root/vupdate3>display $d

variable has not been initialized.
/root/vupdate3>int a = 10
/root/vupdate3>display $a
a : 10
/root/vupdate3>int f
Please follow the syntax : int <var_name> = <var_value>
/root/vupdate3>_
```

4. Calculator Application

4.1 Requirement

Design a calculator application that can use the variables as input and output

4.2 Design

Syntax : cal <result_var> = $<first_var> <operation> $<second_var> (space separated)

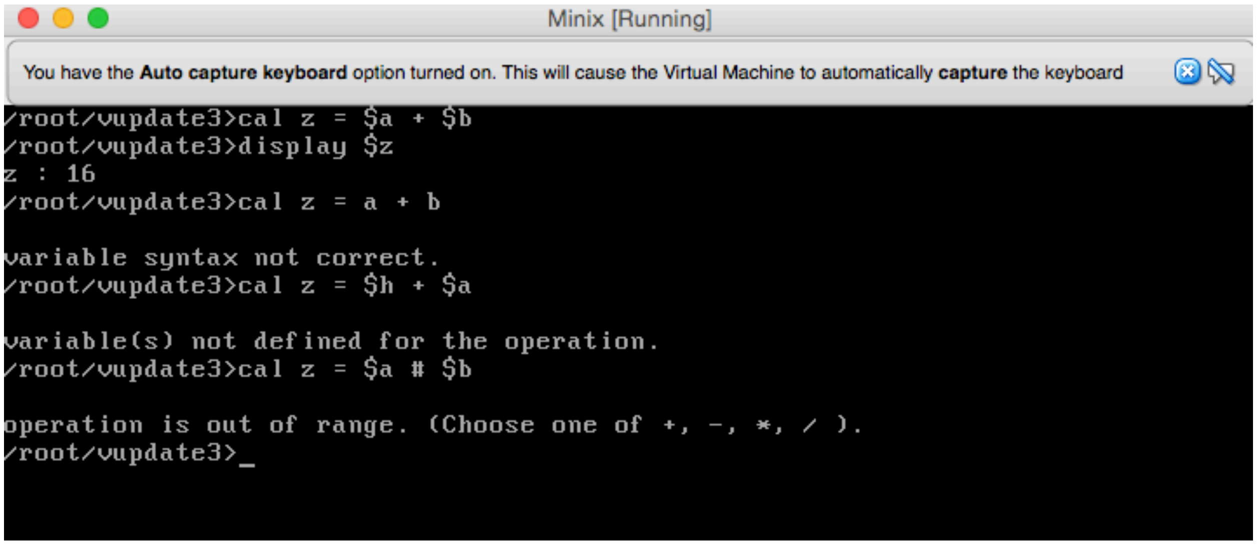
a. **cal_handler()** function gets called when the user input starts with 'cal'.
   - i. Check if the syntax is correct. Generate error message otherwise.
   - ii. Search for the variables in the list. If any of them is not defined, generate error message.
   - iii. If variables found, calculate the answer from the variable values.
   - iv. Search for the result variable in the list.
   - v. If found, update the value. If not found, generate the variable node and add to the list.

4.3 Test Cases

	Description	Input	Expected output	Result
1	Check when both variables exist. And result variable is new.	cal a = $b + $c	New variable is created, and result is stored in the list.	PASS
2	Check when both variables exist. And result variable is already defined.	cal a = $b * $c	The value of result variable gets updated in the list.	PASS

3	Check when at least one of the variables is not defined.	cal a = $b * $z	Warning message 'variables not defined'	PASS
4	Check operation is not defined.	Cal a = $b # $c	Warning message 'operation not defined'	PASS
5	Check when syntax is wrong.	Case: 1. cal a = b + c 2. cal a=$b*$c (no space)	Warning message 'Please follow the syntax'	PASS

4.4 Sample output


Minix [Running]
You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

```
/root/vupdate3>cal z = $a + $b
/root/vupdate3>display $z
z : 16
/root/vupdate3>cal z = a + b

variable syntax not correct.
/root/vupdate3>cal z = $h + $a

variable(s) not defined for the operation.
/root/vupdate3>cal z = $a # $b

operation is out of range. (Choose one of +, -, *, / ).
/root/vupdate3>_
```

5. Command Execution

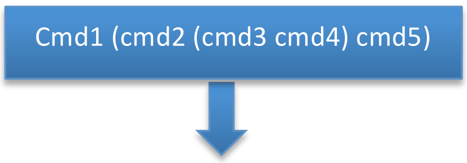
5.1 Requirement

Your shell shall support the execution of a list of commands in the following format: $ (command1, command2, (command3))

5.2 Design

USAGE: command1 (command2 (command3……))

- The parser initially calls a validate function which looks for illegal / unbalanced parentheses and if that is the case, then it rejects the command.
- After the initial inspection, if a valid command is detected, it then parses the command based on different levels of nesting and orders them into different queues based on the precedence (as shown in the figure below)
- Finally, it executes the commands from the highest-level queue (representing deepest nesting) to the lowest one. Since it is a queue, it also maintains the left-right precedence within each level.



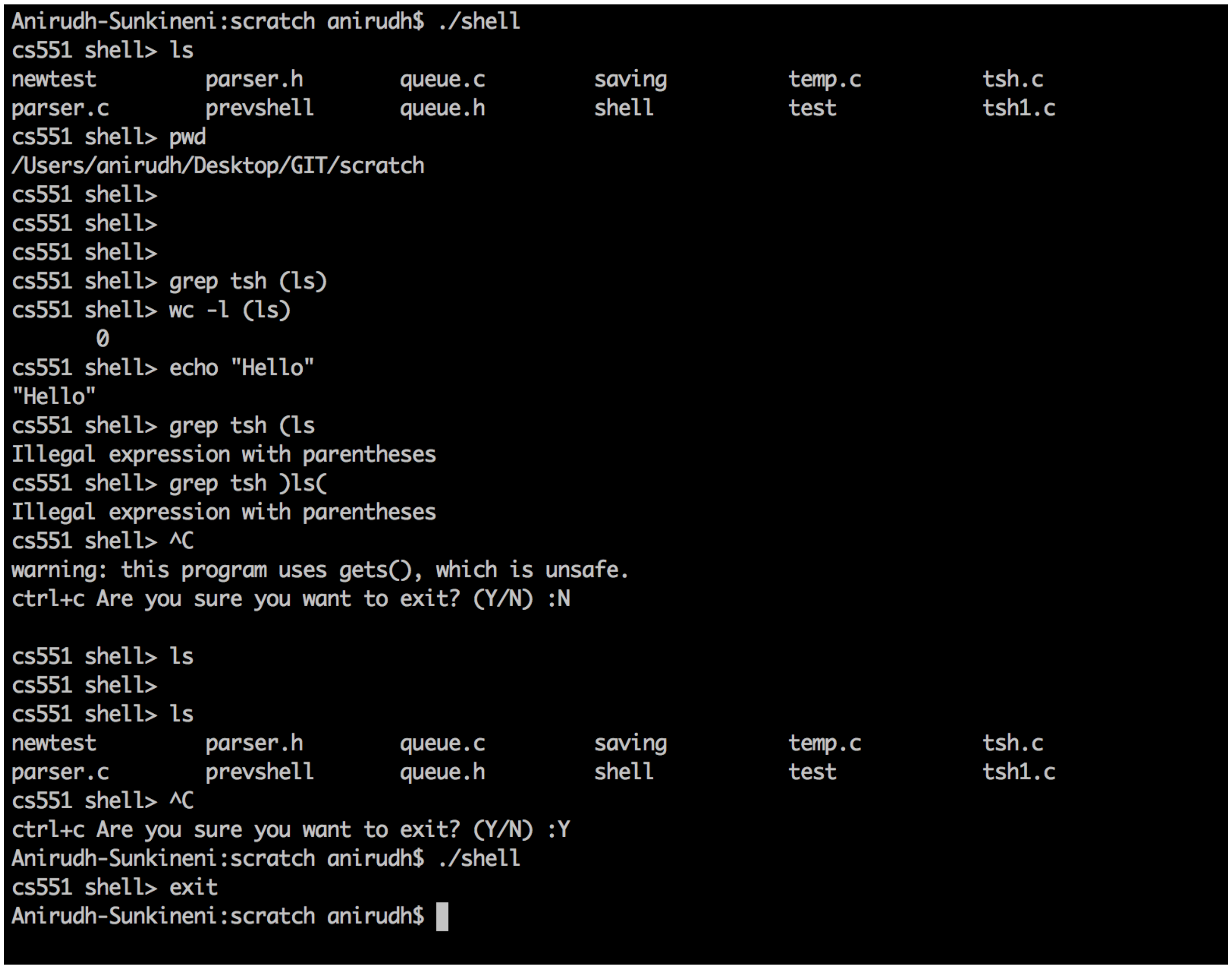
Queue0	Cmd1		
Queue1	Cmd2	Cmd5	
Queue2	Cmd3	Cmd4	
Queue3			

5.3 Test Cases

	Description	Input	Expected output	Result
1	When parentheses are not balanced	wc –l ls)	Illegal expression with parentheses	PASS
2	When parentheses are balanced but illegall formation.	wc –l)ls(	Illegal expression with parentheses	PASS
3	When correct formation	Wc –l (ls)	Resultant output	FAIL
4	More than one pipe	Wc –l (grep *.c (ls))	Resultant output	FAIL
5	Two commands at same level	Grep * (ls) wc -l	Resultant output	FAIL

The parser works correctly, it properly queues the commands to run, based on their precedence. However something seems to be failing in the piping part of the logic.

5.4 Sample Output



```
Anirudh-Sunkineni:scratch anirudh$ ./shell
cs551 shell> ls
newtest         parser.h        queue.c         saving          temp.c          tsh.c
parser.c        prevshell       queue.h         shell           test            tsh1.c
cs551 shell> pwd
/Users/anirudh/Desktop/GIT/scratch
cs551 shell>
cs551 shell>
cs551 shell>
cs551 shell> grep tsh (ls)
cs551 shell> wc -l (ls)
       0
cs551 shell> echo "Hello"
"Hello"
cs551 shell> grep tsh (ls
Illegal expression with parentheses
cs551 shell> grep tsh )ls(
Illegal expression with parentheses
cs551 shell> ^C
warning: this program uses gets(), which is unsafe.
ctrl+c Are you sure you want to exit? (Y/N) :N

cs551 shell> ls
cs551 shell>
cs551 shell> ls
newtest         parser.h        queue.c         saving          temp.c          tsh.c
parser.c        prevshell       queue.h         shell           test            tsh1.c
cs551 shell> ^C
ctrl+c Are you sure you want to exit? (Y/N) :Y
Anirudh-Sunkineni:scratch anirudh$ ./shell
cs551 shell> exit
Anirudh-Sunkineni:scratch anirudh$ 
```

6. Exception handling

Each section already talked about the exception handling for its part, they also had negative test cases to test them. For the program itself, we block most signals and for sigint (ctrl + C) , we have implemented a signal handler that asks for confirmation before exiting. The only other way to exit the program is through an inbuilt command 'exit'

7. Contribution

1. Vivek Pabani:
    - Persistent Variables.
    - Calculator Application.
2. Lakshmi Karle:

    - Read Profile File.
    - Set Prompt.
3. Anirudh Sunkineni:
    - Command Execution.