 **Lab 1 Report**

**Linux Shell Commands、C programming**

**Student ID: 2018380039**

**Student Name: Dikshya Kafle**

**Report Subject: Computer OS Experiment - Laboratory 1**

**Teacher Name: Tianhai Zhao**

**Computer Operating System**

**TABLE OF CONTENTS**

[1. Objectives](#_Toc44237009)

[2. Equipment](#_Toc44237009)

[3. Methodology](#_Toc44237010)

# **Objectives**

* To familiar with Linux Operating system.
* To practice the usage of Linux shell commands.
* To practice with the creation of a Makefile.
* To get reacquainted with C programming.
* To practice your code reading skills.

# **Equipment**

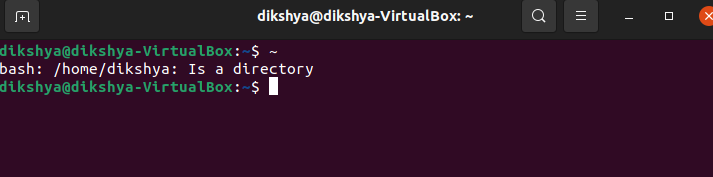
* VirtualBox with Ubuntu Linux 20.04.

# **Methodlogy**

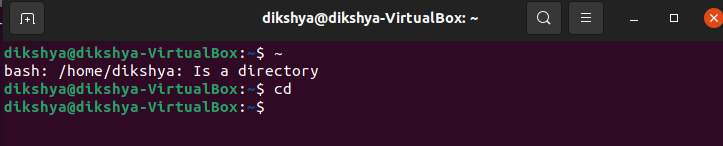
1. **Practise and answer the questions**
2. How do you find your home directory quickly? And change the directory to your home.

**Answer:**

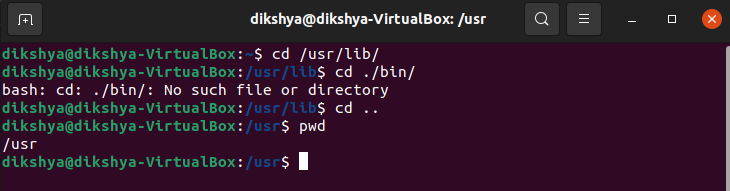
* In linux, the tilde (~) symbol stands for the home directory. In my case my user name is dikshya, so the tilde (~) stands for /home/dikshya



* To directly change from the directory to the home directory we simply use the “cd” command.



1. Type these commands “cd /usr/lib/”, “cd ./bin/”, “cd ..”, and “pwd”. Give the current directory which you are located. What are the meanings of . and .. ?



The current directory is “/usr”. The single dot “.” means the current directory which actually is a hard link to its containing directory. The double dot “..” means one step back i.e. the parent directory, which is the only directory where that directory is referenced from.

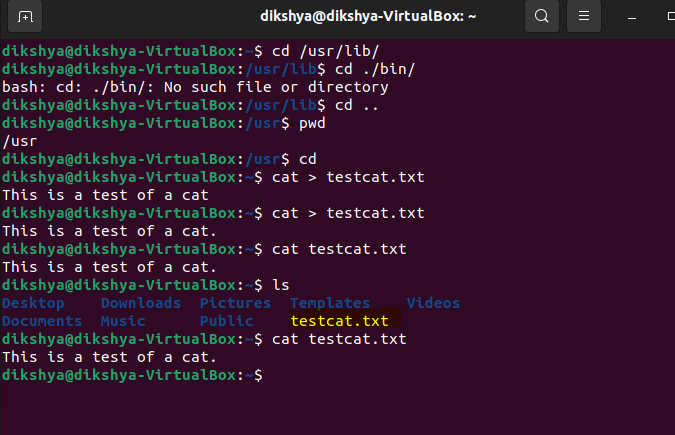
1. Type "cd" to go back to your home directory. Type “cat > testcat.txt” in the command line. After pressing return, type the following line of text “This is a test of cat.”, and then press crtl-d. Type “cat testcat.txt” again. What do you see?

**Answer:**

* The command “cat > testcat.txt” creates a testcat.txt file (in the current directory). After this text file is created we need to type the text we want to include in the text file, once we are done we press CRTL+D to save the file. Then, using the command “cat testcat.txt” will display the text file content.



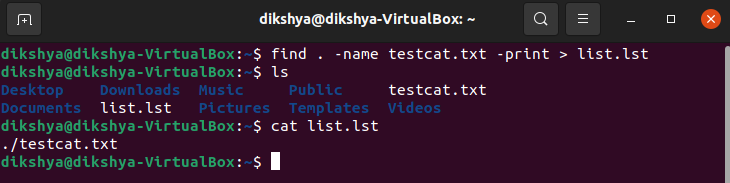
If we check the home directory, we can see the text file is there!



1. Type “find . -name testcat.txt –print > list.lst” in the command line. You will find a file “list.lst” in your current directory. Use cat commands to show its contents. What is the result? What is the meaning of > list.lst?

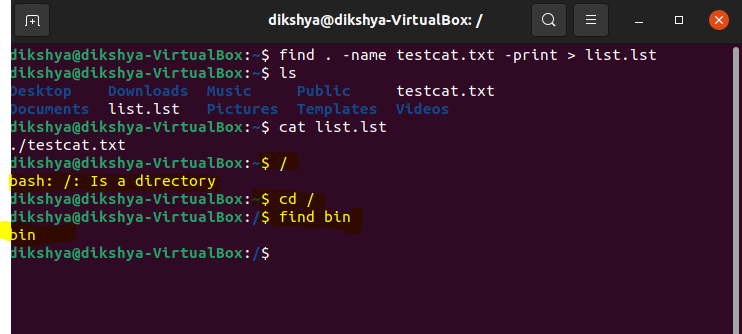
**Answer:**

* The command “find . -name testcat.txt -print” is used to find the file testcat.txt in the current directory and the -print option is meant to print the path name of the file found. The part “> list.lst” means redirect the output to a newly created file called list.lst (text file containing a list of data). If the file already exists, replace/override it. (If the file existed already and we only want to append the result instead of overriding the existing file, we could use >> instead of >)



1. Go to “/” directory and type the command to find the bin directory.

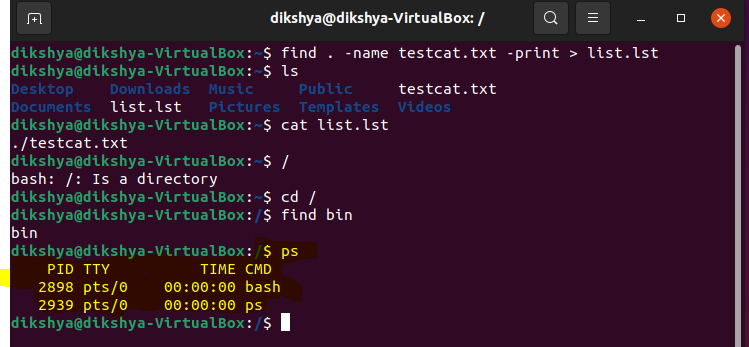
**Answer:**



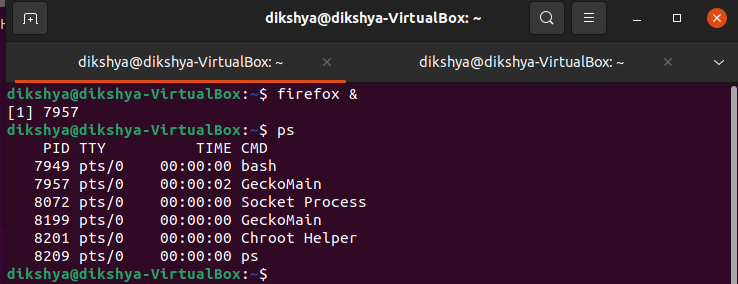
1. Open two terminals on Linux. In each terminal, type the command “ps”. Give the process number (PID) of the bash process in both terminals. Why they are different?

**Answer:**

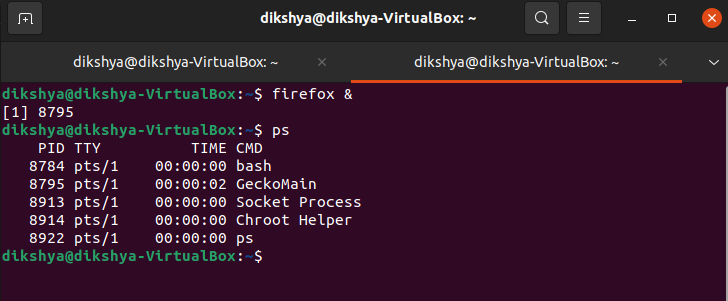
* Each bash process is a different process eventhough the program is the same. They are both two different instances of the same program. There for they are two separate process and this is why they have different process numbers (PID)



1. Start the Fire-Fox web browser by typing "firefox &" at the command line and type “ps” in the terminal. Can you see the process of Fire-Fox web browser? Find the parent process and child process.



**Each terminal has its own bash process hence the two terminals have different PID.**



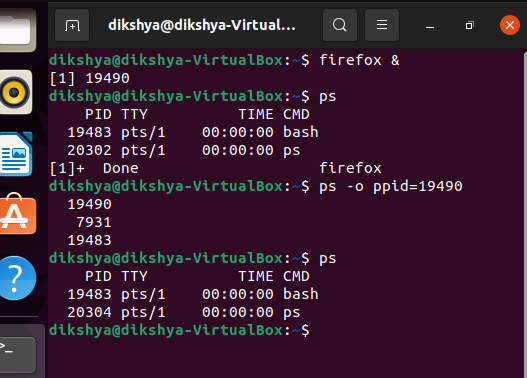
**Yes I can see the process of firefox, the parent process is the second one while the child process is web content.**

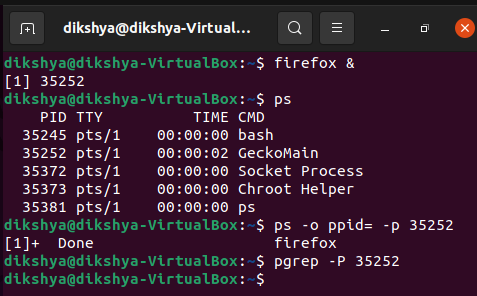
**Answer:**

* To run firefox: we use the command “firefox &”. The command return firefox process id (in this case it is 19490) and we can also see it by using the “ps” command the checking the PID corresponding to the process name under the CMD column.

To find the parent process of the firefox process: use *“ps -o ppid= -p 19490” .* The parent process pid is “19483” which is the bash in this case (verify in the result of the ps command)

To find the child processes of the firefox process: use *“pgrep -P 19490”.* The child processes are “19490”, “7931” and “19483”. These, respectively, correspond to the process names “Privileged Cont”, “WebExtensions” and “Web Content”.

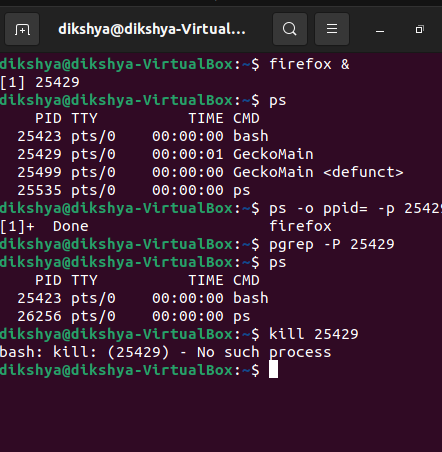


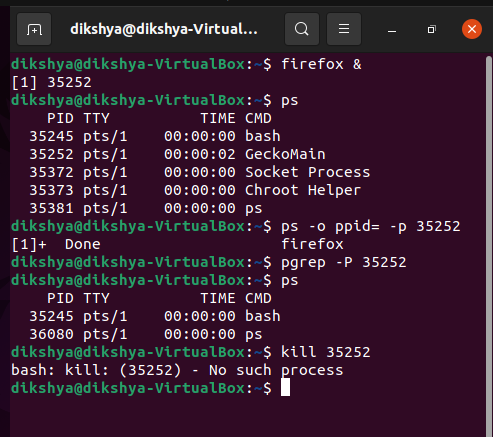


1. Use the command “kill” to kill the processes of Fire-Fox. Give the command you have used. What did you see after killing their processes?

**Answer:**

* To kill the firefox process I used the command “kill 35252”. This has resulted in channel error and I had to click CRTL+C to stop it.





1. **Now you are ready to write a program that reveals its own executing structure. The first.c provides a rather complete skeleton. You will need to modify it to get the**

#include <stdio.h>

#include <stdlib.h>

/\* A statically allocated variable \*/

int var1;

int my\_func(int i){ /\* a recursive function \*/

int j = i; /\* a stack allocated variable within a recursive function \*/

printf("recur call %d: stack@ %lx\n", &i, &j); /\* fix this so it prints the address of j \*/

if (i > 0) {

return my\_func(i-1);

}

return 0;

};

/\* A statically allocated, pre-initialized variable \*/

volatile int stuff = 7;

int main(int argc, char \*argv[]) {

/\* A stack allocated variable \*/

volatile int i = 0;

/\* Dynamically allocate some stuff \*/

volatile char \*buf1 = malloc(10);

/\* ... and some more stuff \*/

volatile char \*buf2 = malloc(10);

printf("\_main @ %lx\n", main); /\* fix to print address of main \*/

printf("recur @ %lx\n", my\_func); /\* fix to print address of recur \*/

printf("\_main stack data i: %lx\n", &i); /\* fix to get address of the stack variable \*/

printf("static data stuff, address: %lx\n", &stuff); /\* fix to get address of a static variable \*/

printf("Heap data buff 1: %lx\n", (long unsigned int) buf1);

printf("Heap data buff 2: %lx\n", (long unsigned int) buf2);

my\_func (3);

printf("%d\n",argc);

for(int j=0;j<argc;j++)

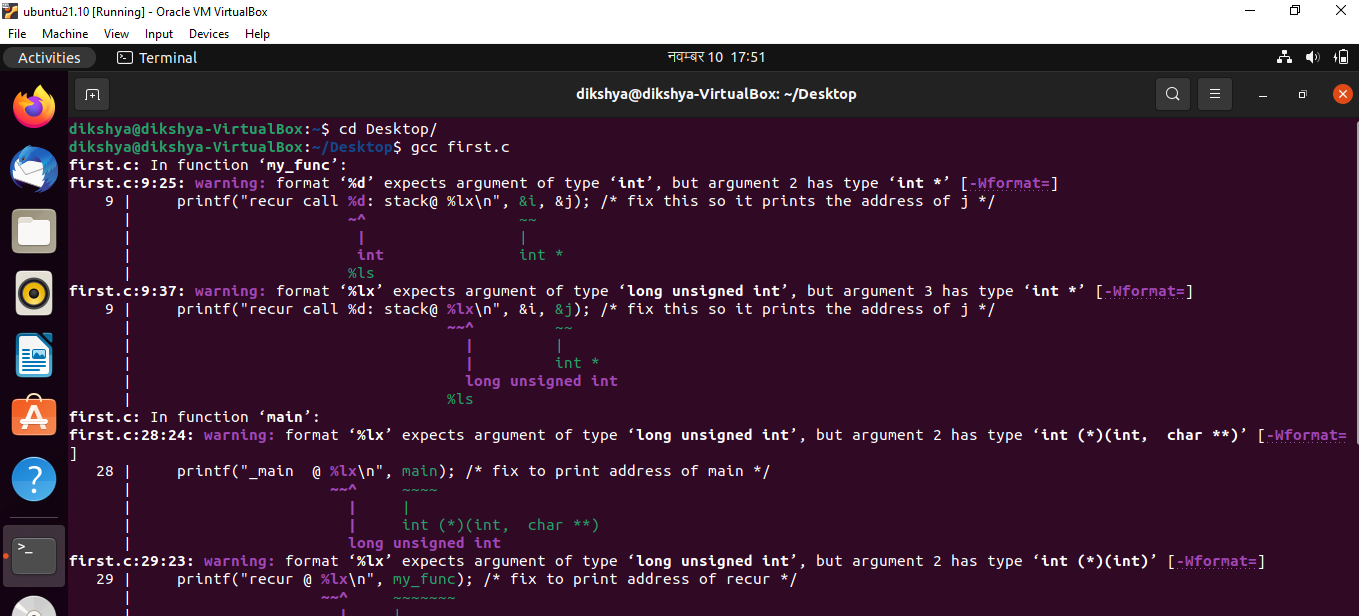
{

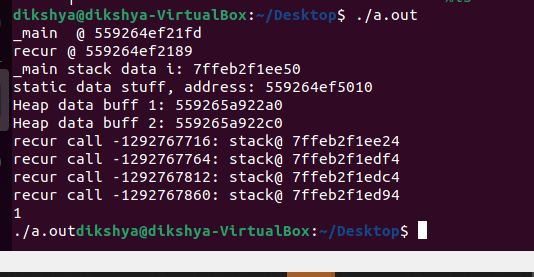
printf("%s",argv[j]);

}

return 0;

}

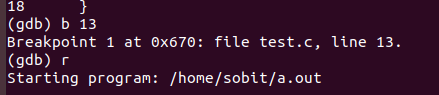




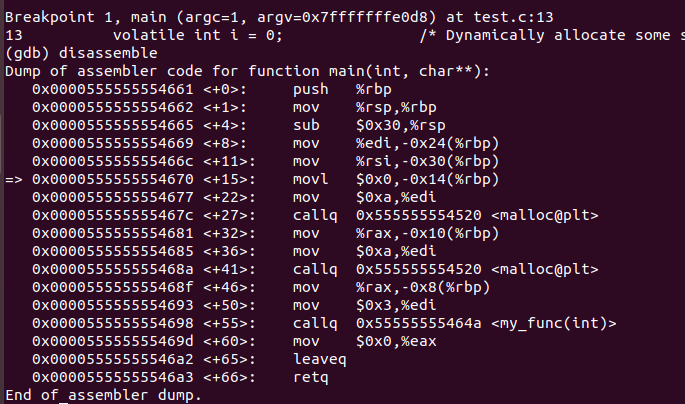
after running the gdb first we list the programme

Using the list command

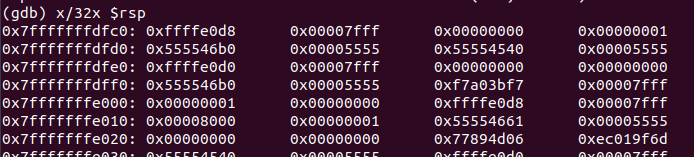
And



Set the breakpoint in the line 13 to example the program

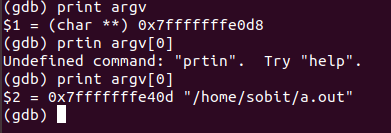


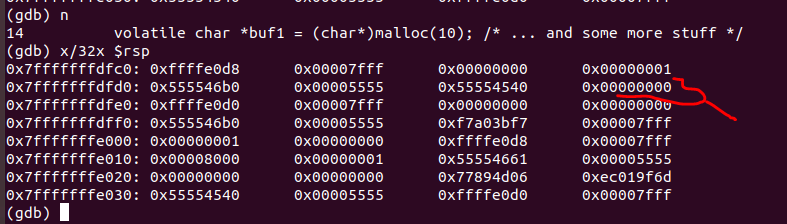
So we show the stack content



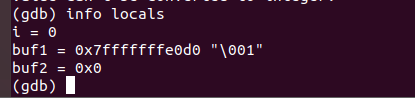
Than increment the instruction by using n command

After that we again print the stack value

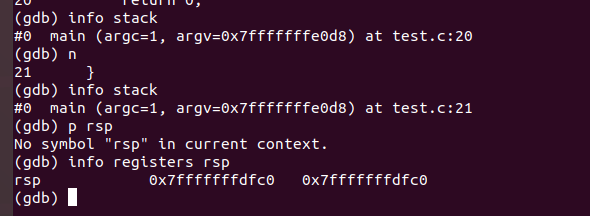




So we can see there that the value of i is store in the stack

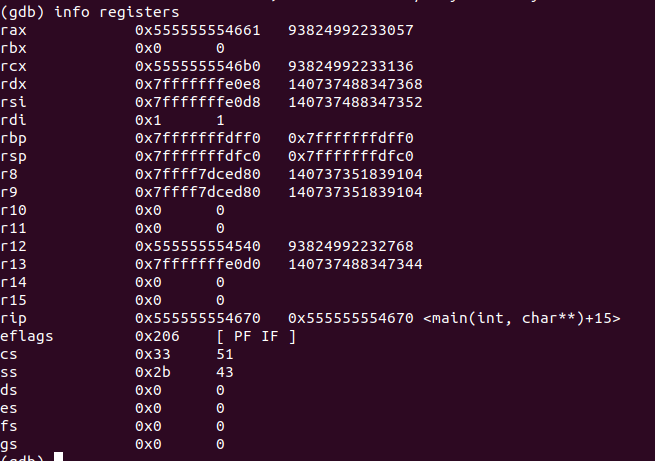






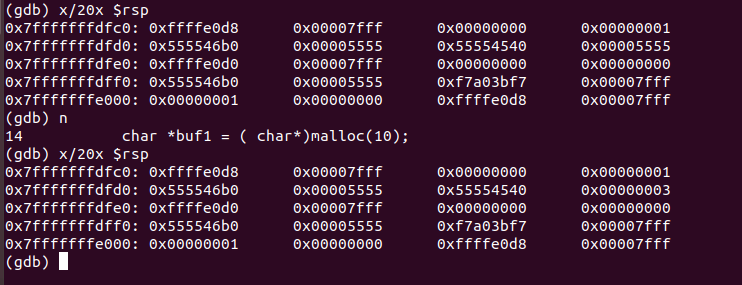
So the main function is in the stack frame 0 which is the first stack frame which is created by the os when programmer is firstly run. The code for the main function reside in the code segment but when the program is called the first stack frame is given to the main function by the OS.

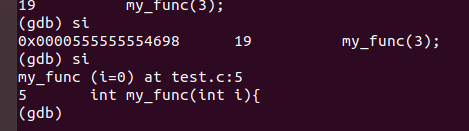
Try info registers. Which registers are holding aspects of the program that you recognize?



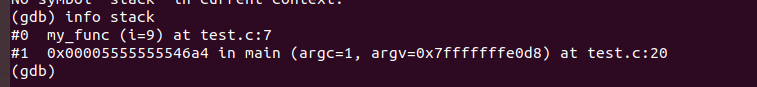
We can see instruction pointer is pointing to the main function with reference to the code segment and rsp is the stack pointer

Then I replace the value of i=3 and run program which we can see in the stack in second row last column





Then we inter in to the my\_func lets check the registers



argv and argc are how command line arguments are passed to main() in C. argc will be the number of strings pointed to by argv. This will (in practice) be 1 plus the number of arguments, as virtually all implementations will prepend the name of the program to the array. argv[0] is the name of the program.The address of main is “0x7fffffffe0d8”. The gdb info stack prints a backtrace of stack frames and prints a verbose description of the selected stack frame.

1. **Write a shell script to capture Process ID by the process name and kill it if it exists**.

The shell script will ask to input a process name, then it tries to capture the process id. If the process is running it kills it otherwise it just outputs that the process is not running and does not perform any action.

