**PROJECT REPORT ON**

**BUILDING OWN LINUX TERMINAL IN C**

***Submitted By***

## **DEBIPRASAD SAHOO– 160301120174**

## **SUJIT PRADHAN– 160301120151**

## **JAYADEV DAS-160301120179**

## **ASISH KUMAR PRADHAN-160301120180**

**Under the supervision of Prof. Mr. Suvendu Nayak**

**Department of Computer Science & Engineering**

**In partial fulfilment for the award of the degree of**

**Bachelor of Technology**

**In**

**Computer Science & Engineering**

**Centurion University of Technology & Management**

**Jatni Campus, Pin Code: 752050, Bhubaneswar Odisha**

CERTIFICATE

This is to certify that the project “Own Terminal Using C” carried out by “Debiprasad Sahoo,Sujit Pradhan,Jayadev Das,Asish Kumar Pradhan” under my supervision at Department of Computer Science & Engineering, CUTM, Bhubaneswar. The work is original, as it has not been submitted earlier either in part or full for any purpose before.

(Prof. Mr. Suvendu Nayak)

(Dept. Of Computer Science & Eng.,)

Certified that the above mentioned project has been duly carried out as per the norms of the college and statutes of the university.

Dr. Prasanta Ku. Mohanty

Dean Academics

Head of the Department

Computer Science Engineering.

DECLARATION

We, hereby declare that the work presented in this dissertation entitled **“LINUX SHELL USING C”** has done by us, and this dissertation embodies our own work under the guidance of **“Mr. Suvendu Nayak”** of **“Operating System Project”** subject of Computer Science and Engineering at **“Centurion Institute of Technology”**.

Place: (Mr. Suvendu Nayak)

Date:

## 

## **DEBIPRASAD SAHOO(160301120174)**

## **SUJIT PRADHAN(160301120151)**

## **JAYADEV DAS(160301120179)**

## **ASISH KUMAR PRADHAN(160301120180)**

ACKNOWLEGEMENT

We thank Prof. Suvendu Nayak who have been the great inspiration and who have provided sufficient background knowledge and understanding of this project. Our humble prostration goes to our Dr. P K Mohanty Head of the Department of Computer Science & Engineering, Dean Academics, CUTM, for providing all the necessary resources and environment, which have aided us to complete this project successfully.

We would like to thank all our friends and specially our seniors for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious.

## **DEBIPRASAD SAHOO(160301120174)**

## **SUJIT PRADHAN(160301120151)**

## **JAYADEV DAS(160301120179)**

**ASISH KUMAR PRADHAN(160301120180)**

PREFACE

This report is an introduction to the OWN LINUX TERMINAL in “C” anybody who doesn’t know even the basics of about mentioned project in “c”, will be certainly able to understand and gain knowledge from this report. The core theme of this project report focuses on the development of Linux Terminal Using “C”.

The report also contains the strategy of making different module of this project which serves good idea to make the project and learn “C” language to the programmer. The most of the idea of making this project and report is taken from **“Operating System Concepts (9th Edition)** by Silberschatz, Galvin and Gange”.

CONETNTS

Title Page No

**Introduction …….……………………………………………………………. 7**

**Virtual Environment …….…………………………………………….…………….. 8-9**

**What is shell? …….……………………………………………………………. 10**

**How Does Shell Works? ..……….………………………………………………… 11**

**C Language ………………………………………………………………………. 12**

**Writing The C code For Linux Terminal ……………………………………………. 13-15**

**PART I (Startup The Shell)…………………………………………………………….. 16-17**

**PART II (Wait For User Input) ………………………………………………………… 18-19**

**PART III (Prase User Input)……………………………………………………………. 20-21**

**PART IV (Execute The command And Return The Result) ……………………… 22-23**

**Conclusion ……………….…………………………………………………………… 24**

**INTRODUCTION**

The goal of this project is to explain Linux basics to new users , covers getting started with the terminal, the Linux command line, and executing commands.  If you are new to Linux, you will want to familiarize yourself with the terminal, as it is the standard way to interact with a Linux server. Using the command line may seem like a daunting task but it is actually very easy if you start with the basics, and build your skills from there.

So, basically, a shell is a program that receives commands from the user and gives it to the OS to process, and it shows the output. Linux's shell is its main part. Its distros come in GUI (graphical user interface), but basically, Linux has a CLI (command line interface). In this report, we are going to cover the basic commands that we use in the shell of Linux.

The whole project is written in “C” language,Which is run through the virtual Environment to Protect our OS from System Crash.

VIRTUAL ENVIRONMENT SETUP

A Virtual Environment is a software based system that can fully mirror the functionality of a target System-on-Chip or board. These virtual platforms combine high-speed processor simulators and high-level, fully functional models of the hardware building blocks, to provide an abstract, executable representation of the hardware to software developers and to system architects.

***Notes:***

### Advantages of Virtualization

**#1 - Consolidation**  
Multiple operating systems can run in the same server, eliminating the need to dedicate a single machine to one application.Old and new applications can run simultaneously with their respective operating systems in multicore servers with manythreads of execution, saving space and power consumption in the datacenter. New versions of an OS and new applicationscan be deployed without purchasing new hardware.  
  
**#2 - Stability and Security**  
Conflicts can arise between supposedly stable applications, and troubleshooting can be daunting. As a result, cautioussystem administrators often host each type of application in a separate server even if the server is grossly underutilized.Multiple virtual machines running bread and butter applications are kept safely separated from each other. In addition, sinceeach VM is isolated from the rest, a security breach in one does not affect the others. The fault tolerance and securitybrought about by the isolation of each virtual machine is a major benefit of virtualization.  
  
**#3 - Development Flexibility**  
A virtualized computer can host numerous versions of an operating system, allowing developers to test their programs indifferent OS environments on the same machine. In addition, with each application running in its own virtual partition,crashing in one virtual machine will not bring down the system.  
  
**#4 - Migration and Cloning**  
Virtual machines, each with their own OS and applications, function like self-contained packages that are said to be"decoupled from the hardware." It is relatively easy to move a VM from one server to another to balance the workload, tomigrate to faster hardware, as well as to recover from hardware failure. VMs can be quickly cloned and deployed.

How To Install Virtual Box

**On Windows**

**1**

**Open the VirtualBox website.** Go to <https://www.virtualbox.org/> in your computer's Internet browser. This is the website from which you'll download the VirtualBox setup file.

**2**

**Click Download VirtualBox.** It's a blue button in the middle of the page. Doing so will open the downloads page.

**3**

**Click Windows hosts.** You'll see this link below the "VirtualBox 5.2.8 platform packages" heading. The VirtualBox EXE file will begin downloading onto your computer.

**4**

**Open the VirtualBox EXE file.** Go to the location to which the EXE file downloaded and double-click the file. Doing so will open the VirtualBox installation window.

**5**

**Navigate through the installation prompts.** Do the following:

Click **Next** on the first three pages.

Click **Yes** when prompted.

Click **Install**

Click **Yes** when prompted.

**6**

**Click Install when prompted.** Doing so will allow VirtualBox to begin installing on your computer.

**7**

**Click Finish when prompted.** It's in the lower-right side of the window. Doing so will close the installation window and open VirtualBox. Now that you've installed and opened VirtualBox, you can [create a virtual machine](https://www.wikihow.com/Install-VirtualBox#step_4_1) in order to run any operating system on your PC.

**Make sure that you don't uncheck the "Start" box before doing this.**

What Is Linux Shell (Terminal):

An Operating is made of many components, but its two prime components are -

* Kernel
* Shell

In a Linux system, the shell is a command-line interface that interprets a user's commands and script files, and tells the server's operating system what to do with them. There are several shells that are widely used, such as Bourne shell (sh) and C shell (csh). Each shell has its own feature set and intricacies, regarding how commands are interpreted, but they all feature input and output redirection, variables, and condition-testing, among other things.

When you run the terminal, the Shell issues **a command prompt ,**where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**

How does Shell Works?

The program does take control of the screen. When you launch a program via a command to bash and you don't put it into the background, bash goes into hibernation and your program takes over both screen and keyboard. When your program exits, bash wakes up again.

Here's the sequence of events:

1) bash calls fork() to create a new process.

2) The parent process calls wait() and goes to sleep.

3) More or less simultaneously the child process calls exec() to switch to the specified program.

4) The program runs and outputs to the screen.

5) The program exits. The kernel notifies bash.

6) bash wakes up and prints a new prompt.

The C Language:

C is a high-level and general-purpose programming language that is ideal for developing firmware or portable applications. Originally intended for writing system software, C was developed at Bell Labs by Dennis Ritchie for the Unix Operating System in the early 1970s.

C belongs to the structured, procedural paradigms of languages. It is proven, flexible and powerful and may be used for a variety of different applications. Although high level, C and assembly language share many of the same attributes.

Some of C's most important features include:

* Fixed number of keywords, including a set of control primitives, such as if, for, while, switch and do while
* Multiple logical and mathematical operators, including bit manipulators
* Multiple assignments may be applied in a single statement.
* Function return values are not always required and may be ignored if unneeded.
* Typing is static. All data has type but may be implicitly converted.
* Basic form of modularity, as files may be separately compiled and linked
* Control of function and object visibility to other files via extern and static attributes

CODING PART :

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <errno.h>

void parseCmd(char\* cmd, char\*\* params);

int executeCmd(char\*\* params);

#define MAX\_COMMAND\_LENGTH 100

#define MAX\_NUMBER\_OF\_PARAMS 10

int main()

{

char cmd[MAX\_COMMAND\_LENGTH + 1];

char\* params[MAX\_NUMBER\_OF\_PARAMS + 1];

int cmdCount = 0;

while(1) {

// Print command prompt

char\* username = getenv("USER");

printf("%s@shell %d> ", username, ++cmdCount);

// Read command from standard input, exit on Ctrl+D

if(fgets(cmd, sizeof(cmd), stdin) == NULL) break;

// Remove trailing newline character, if any

if(cmd[strlen(cmd)-1] == '\n') {

cmd[strlen(cmd)-1] = '\0';

}

// Split cmd into array of parameters

parseCmd(cmd, params);

// Exit?

if(strcmp(params[0], "exit") == 0) break;

// Execute command

if(executeCmd(params) == 0) break;

}

return 0;

}

// Split cmd into array of parameters

void parseCmd(char\* cmd, char\*\* params)

{

for(int i = 0; i < MAX\_NUMBER\_OF\_PARAMS; i++) {

params[i] = strsep(&cmd, " ");

if(params[i] == NULL) break;

}

}

int executeCmd(char\*\* params)

{

// Fork process

pid\_t pid = fork();

// Error

if (pid == -1) {

char\* error = strerror(errno);

printf("fork: %s\n", error);

return 1;

}

// Child process

else if (pid == 0) {

// Execute command

execvp(params[0], params);

// Error occurred

char\* error = strerror(errno);

printf("shell: %s: %s\n", params[0], error);

return 0;

}

// Parent process

else {

// Wait for child process to finish

int childStatus;

waitpid(pid, &childStatus, 0);

return 1;

}

}

Part –I (Start Up Shell)

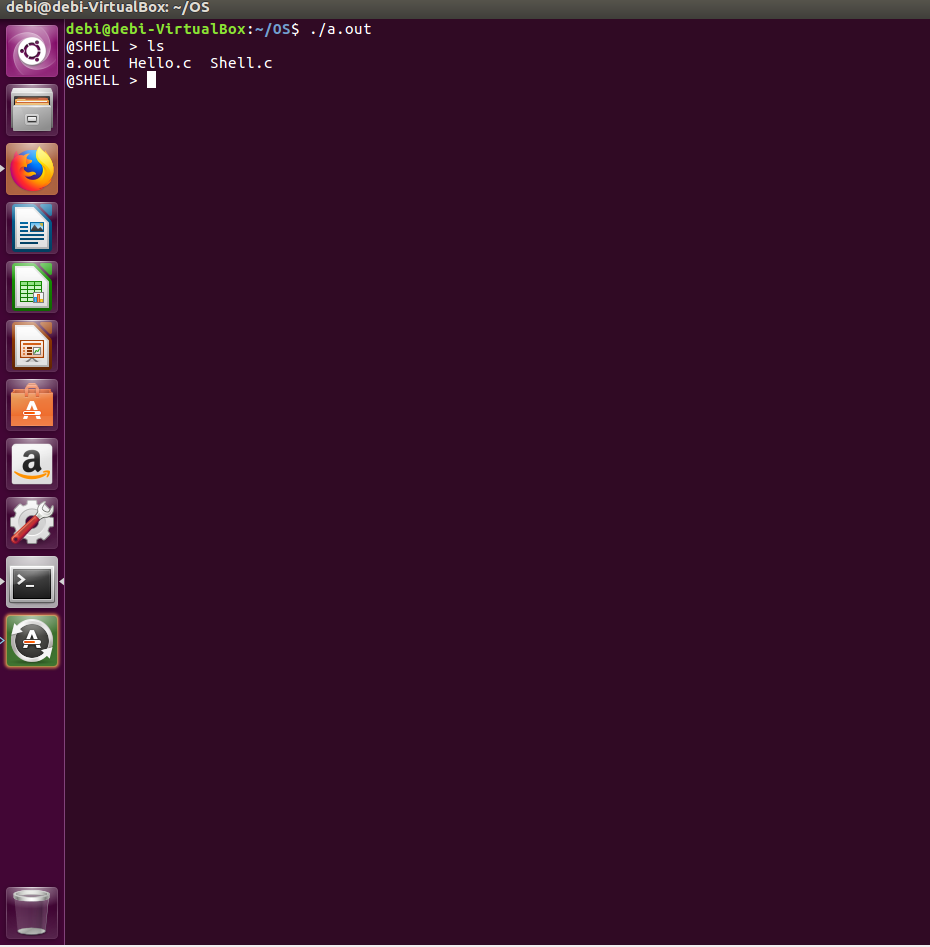
Now, we’re understand about what a shell does. Starting processes is the main function of shells. So writing a shell means that you need to know exactly what’s going on with processes and how they start.

There are only two ways of starting processes on Unix. The first one (which almost doesn’t count) is by being Init. when a Unix computer boots, its kernel is loaded. Once it is loaded and initialized, the kernel starts only one process, which is called Init. This process runs for the entire length of time that the computer is on, and it manages loading up the rest of the processes that you need for your computer to be useful.

Since most programs aren’t Init, that leaves only one practical way for processes to get started: the fork() system call. When this function is called, the operating system makes a duplicate of the process and starts them both running. The original process is called the “parent”, and the new one is called the “child”. fork() returns 0 to the child process, and it returns to the parent the process ID number (PID) of its child. In essence, this means that the only way for new processes is to start is by an existing one duplicating itself.

This might sound like a problem. Typically, when you want to run a new process, you don’t just want another copy of the same program – you want to run a different program. That’s what the exec() system call is all about. It replaces the current running program with an entirely new one. This means that when you call exec, the operating system stops your process, loads up the new program, and starts that one in its place. A process never returns from an exec() call (unless there’s an error).

With these two system calls, we have the building blocks for how most programs are run on Unix. 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 the system call wait().

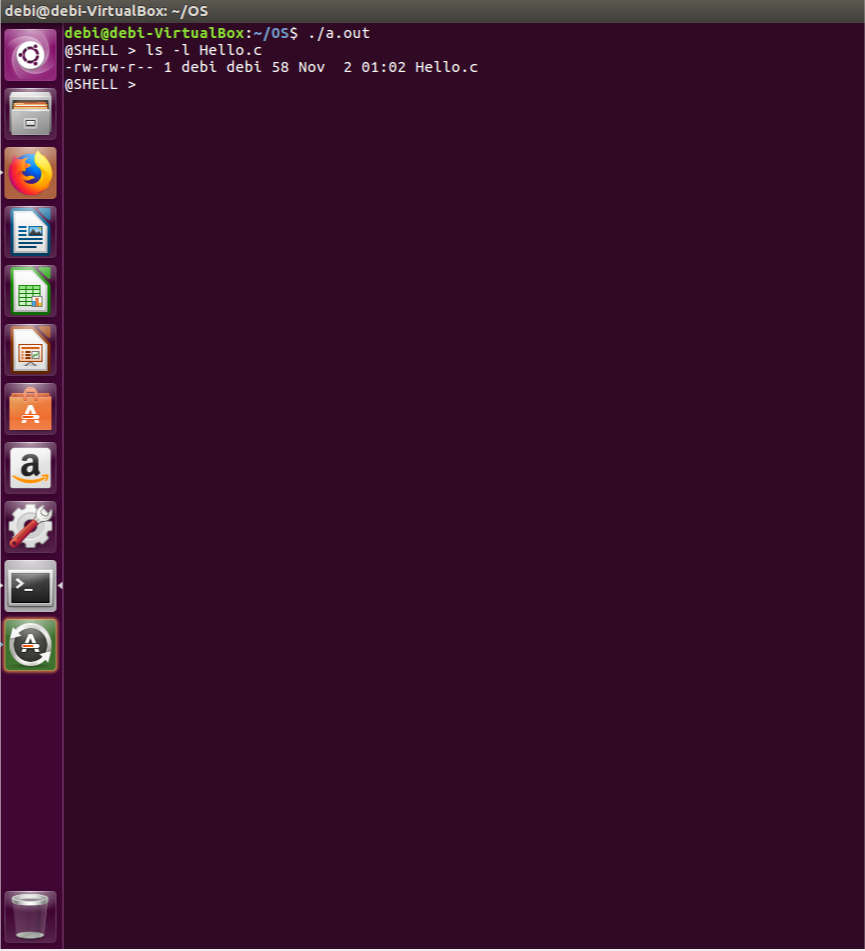


PART-II (Wait For User Input)

Reading a line from stdin sounds so simple, but in C it can be a hassle. The sad thing is that you don’t know ahead of time how much text a user will enter into their shell. You can’t simply allocate a block and hope they don’t exceed it. Instead, you need to start with a block, and if they do exceed it, reallocate with more space. This is a common strategy in C, and we’ll use it to implement lsh\_read\_line

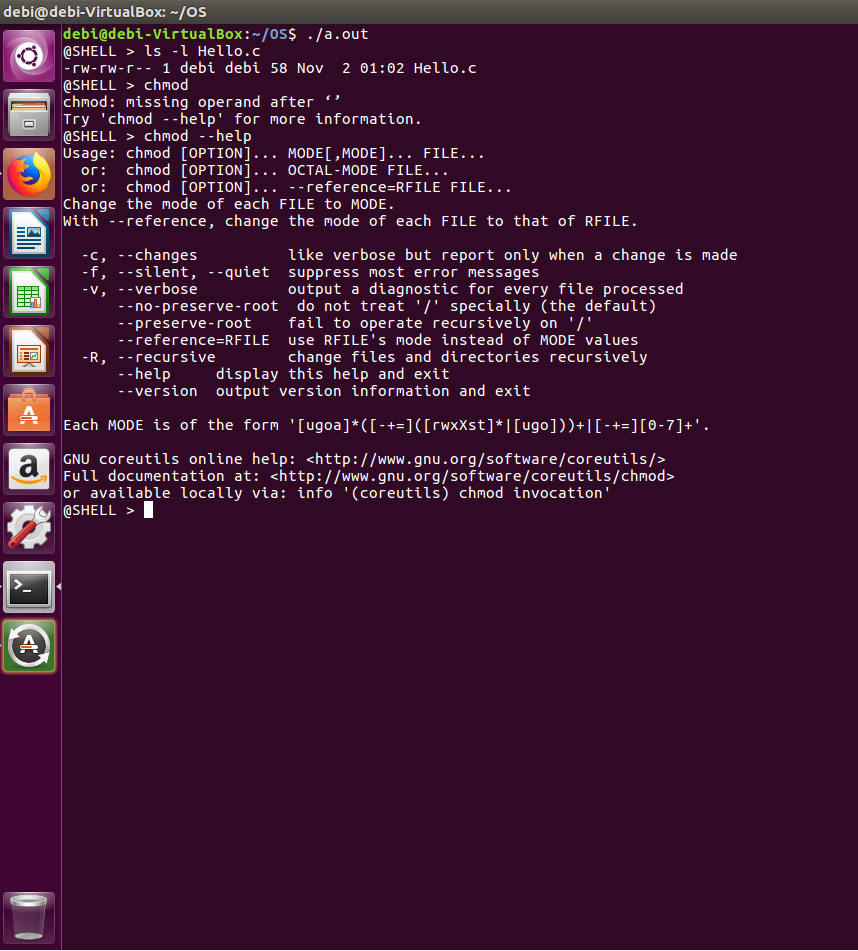
Those who are intimately familiar with newer versions of the C library may note that there is a getline() function in stdio.h that does most of the work

For user input section. This function was a GNU extension to the C library until 2008, when it was added to the specification, so most modern Unixes should have it now.



PART III (Prase User Input)

So if we look back at the loop, we see that we now have implemented lsh\_read\_line(), and we have the line of input. Now, we need to parse that line into a list of arguments. I’m going to make a glaring simplification here, and say that we won’t allow quoting or backslash escaping in our command line arguments. Instead, we will simply use whitespace to separate arguments from each other. So the command echo "this message" would not call echo with a single argument this message, but rather it would call echo with two arguments: "this and message".With those simplifications, all we need to do is “tokenize” the string using whitespace as delimiters. That means we can break out the classic library function strtok to do some of the dirty work for us.



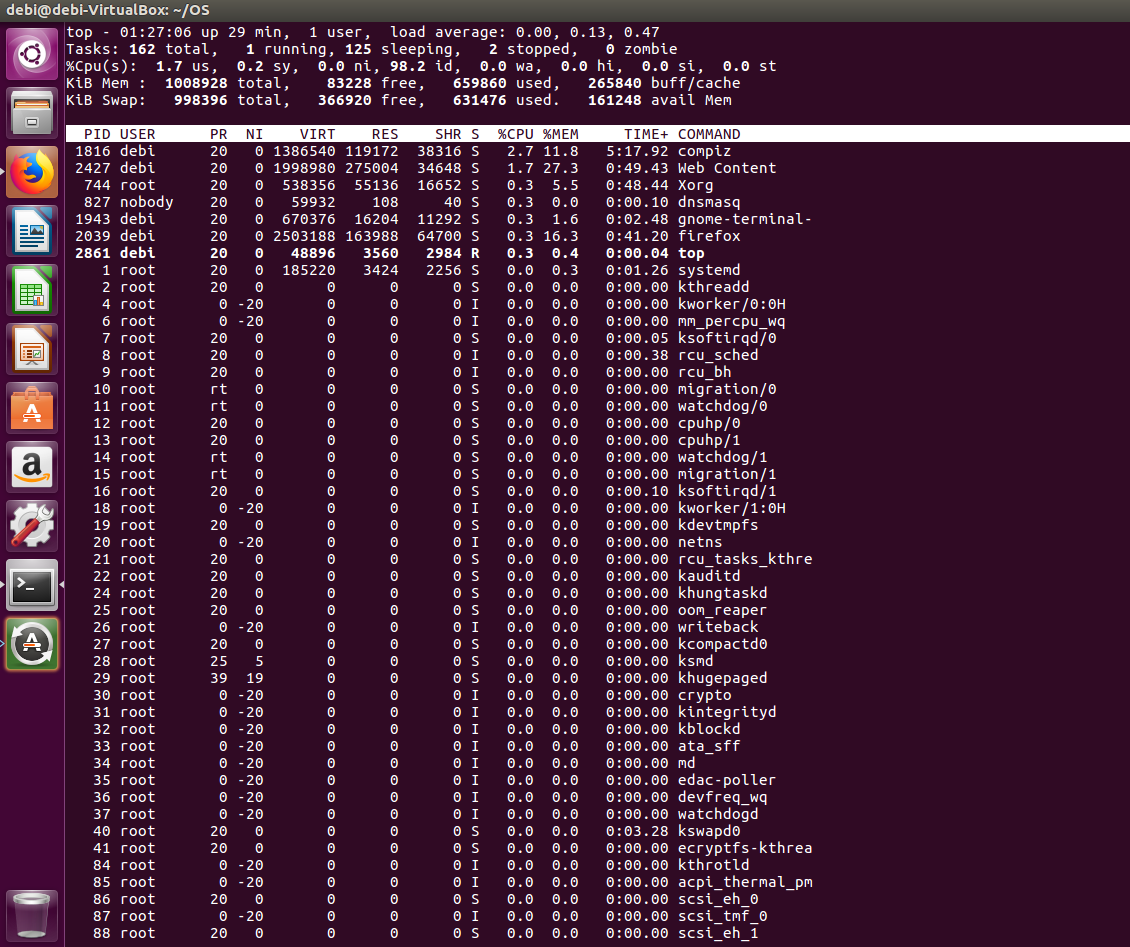
PART IV (Execute The command And Return The Result)

Commands can be issued at the command prompt by specifying the name of an executable file, which can be a binary program or a script. There are many standard Linux commands and utilities that are installed with the OS, that allow you navigate the file system, install and software packages, and configure the system and applications.

An instance of a running command is known as a **process**. When a command is executed in the *foreground*, which is the default way that commands are executed, the user must wait for the process to finish before being returned to the command prompt, at which point they can continue issuing more commands.

It is important to note that almost everything in Linux is case-sensitive, including file and directory names, commands, arguments, and options. If something is not working as expected, double-check the spelling and case of your commands!

Out Put Of The TOP command



**Conclusion**

The title of this project repport is "Own linux shell using C" and that's what we've covered. By now is, you've seen that there are a variety of ways to find out more about your options on the command line.

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](https://en.wikipedia.org/wiki/Kernel_(computer_science)) interface, which is the lowest-level, or "inner-most" component of most operating systems.

so you'll be able to run different linux commands or instructions already given by the OS and understand exactly what's happening. Linux is all about variety and choice, even in such seemingly small matters as commands run in the shell.