Bahir Dar institute of Technology

Faculty of Computing

Software Engineering

Project-work

Operating system and system programming

Name: Elias Zeleke

Id: BDU1601353

Submitted to: Wendimu Baye

Submitted date: 16/8/2017

Table of contents

Introduction

Operating systems are essential for managing computer hardware and enabling efficient execution of software applications. One of their core responsibilities is facilitating communication between user programs and the system's kernel through system calls. This assignment explores the functionality and significance of the execve system call within the context of Tiny Core Linux.

The execve system call plays a crucial role in process management by replacing the current process image with a new executable, thus enabling program execution. By examining how execve operates in Tiny Core Linux, this study aims to provide a deeper understanding of process creation, execution, and the lightweight design principles of modern operating systems.

- *Tiny Core Linux: Background and Technical Overview*

Tiny Core Linux (TCL) is a minimalist, open-source Linux distribution designed for speed, modularity, and flexibility. It emerged in 2009 as a spiritual successor to Damn Small Linux (DSL), a project led by Robert Shingledecker, who later co-created Tiny Core.

Shingledecker parted ways with DSL later in 2008 due to differences in development aims and philosophies.

He desired a more modular and flexible system, and thus was born Tiny Core Linux.

Architecture and Design

TCL is built around a minimal Linux kernel (from the stable series) and leverages BusyBox for core Unix utilities, ensuring a tiny footprint. Its graphical interface uses FLTK (Fast Light Toolkit) and a lightweight window manager like FLWM.

Key design principles include

- Modularity: Only essential components (kernel, init system, and basic tools) are included in the base.

- On-Demand Extensions: Users install packages (e.g., GUI tools, drivers, apps) via a repository, loaded dynamically.

- Persistence: Changes are preserved through manual backups or persistent storage (e.g., tce directory), allowing customization without bloating the core.

Versions

TCL offers three editions to suit diverse needs:

1. Tiny Core (16 MB): Base system with CLI and basic hardware support.

2. Core (11 MB): Pure text-mode version for servers/embedded use.

3. Core Plus (200+ MB): Installer-friendly ISO with Wi-Fi support, non-US keyboard layouts, and remastering tools.

Extension System

The TCE (Tiny Core Extension) repository hosts over 1,000 precompiled extensions, managed via the tce-ab tool. Users can dynamically load/unload software (e.g., browsers, development tools) without rebooting. Extensions are SquashFS-compressed for efficiency, and dependencies are resolved automatically. This system enables a "just enough OS" approach, ideal for tailored deployments.

Installation and Deployment

TCL supports multiple installation modes:

- Live CD/USB: Runs entirely in RAM, leaving the host system untouched.

- Frugal Install: Stores OS files on a disk partition, allowing persistence.

- Cloud/Embedded: Minimal footprint suits virtualization (e.g Docker) or IOT devices.

Community and Development

TCL is developed collaboratively, with Shingledecker and a global community maintaining extensions and core updates. The project adheres to a "do-it-yourself" ethos, encouraging users to contribute packages or modify the system. Documentation is sparse but supplemented by active forums, reflecting its niche, expert-oriented audience.

Use Cases

- Legacy Hardware Revival: Runs smoothly on PCs with as little as 128 MB RAM.
- Embedded Systems: Serves as a base for kiosks, routers, or IoT controllers.
- Education/Experimentation: Ideal for learning Linux internals.
- Disposable Environments: Secure, ephemeral sessions for testing or recovery.

Technical Innovations

- Mountable Application Files: Extensions are mounted as virtual file systems, reducing disk writes.
- Boot Options: Customizable boot codes (e.g., persistent, norestore) control system behavior.
- Security: Minimal attack surface and optional firewall/encryption tools.

• *Objectives*

Objectives of Tiny Core Linux

Tiny Core Linux is a minimalistic Linux distribution with specific goals that distinguish it from standard Linux distros. Its core objectives are:

1. Minimalism and Lightweight Design

To provide a fully functional Linux system with the smallest possible footprint (the Core edition is only around 11MB).

To operate efficiently on resource-constrained environments such as old computers or embedded systems.

2. Modularity and Customization

To enable users to build and customize their own system from a minimal base. Users can load additional features and applications as extensions, making it highly flexible.

3. Fast Boot and Performance

To boot extremely quickly by loading the core system into RAM.

Running from RAM also allows for fast and responsive system behavior, especially useful in testing and kiosk environments.

4. Educational Use and Experimentation

To serve as a platform for users who want to learn how Linux works internally without the complexity of full-featured distros.

Perfect for operating system research, custom system builds, and kernel-level exploration.

5. Minimal Resource Consumption

Requires as little as 46MB of RAM and can run on very low-powered CPUs.

Ideal for virtual machines, embedded devices, and lightweight desktop usage.

Objectives of the execve System Call

The execve system call is one of the most fundamental components of Unix-like operating systems. Its objectives include:

1. Program Execution

To execute a new program by replacing the current process image with a new one.

Unlike fork(), which creates a new process, execve transforms the calling process into a new program.

2. Process Control and Management

To give processes the ability to launch other programs directly.

This is essential in shells, system utilities, and application launchers where dynamic program execution is needed.

3. Resource Reuse

To avoid the overhead of creating a new process when reusing the existing one is more efficient.

It discards the old program's memory and loads the new one, preserving the PID.

4. Passing Arguments and Environment Variables

To provide a mechanism for passing argv[] and envp[] (arguments and environment) to the new program.

This is crucial for the configuration and runtime behavior of programs.

5. Security and Isolation

Ensures that the new program runs with its own execution context, memory layout, and privilege levels.

Enables the safe switching between user programs without contaminating the previous process's state.

6. Kernel-Level Interaction

Serves as an interface between user space and the kernel for executing binaries.

Involves subsystems such as the file system (to locate the binary), memory manager (to map the new image), and scheduler.

additional to this the overall objectives of this individual assignment:

1. Hands-On Experience with System Programming

To provide experience in compiling or navigating the Linux kernel in a lightweight system.

To develop and test small programs that utilize the execve system call, using tools like strace or gdb for analysis.

2. Enhancing Conceptual and Practical Knowledge

To reinforce theoretical knowledge of process creation, execution, and memory management in operating systems.

To build foundational skills for advanced topics such as custom OS development, kernel module programming, or embedded systems.

3. Research and Technical Writing

To practice documenting technical findings, analysis, and source code explanations clearly and professionally.

To encourage independent exploration of open-source operating system components.

• Requirements

A. Software Requirements

1. Operating System

Primary: Tiny Core Linux (for system-level experimentation)

Host OS: Ubuntu / Windows with WSL / or another Linux distro (for development and VM management)

2. Virtualization / Emulation Tools

Virtual environment tools (oracle vm virtual box , VMware Workstation Player…) to run Tiny Core Linux

3. Development Tools

GCC (GNU Compiler Collection – for compiling C code)

GNU Make (for building and managing code compilation)

GDB (GNU Debugger – to trace and debug system-level code)

strace and ltrace (for tracking system and library calls)

4. Linux Utilities

Core Unix/Linux commands (ps, top, kill, ls, chmod, etc.)

System calls reference documentation (man pages, TLDP, or Linux kernel docs)

B. Hardware Requirements

1. Processor (CPU)

At least Intel Core i5 or Ryzen 5 (quad-core recommended)

2. Memory (RAM)

8 GB minimum, 16 GB recommended (especially if using VMs)

3. Storage

256 GB SSD (Tiny Core is lightweight, but VMs and development tools need space)

4. Other Essentials

USB Flash Drive (4–8 GB) – for creating bootable Tiny Core Linux if needed

Reliable Internet – for downloading packages, tools, and accessing documentations

❖ Installation steps

Step 1: Download Required Files

Download Tiny Core Linux (CorePlus ISO) from:

https://tinycorelinux.net its official website

Download and install VMware Workstation Player from:

https://www.vmware.com its official website

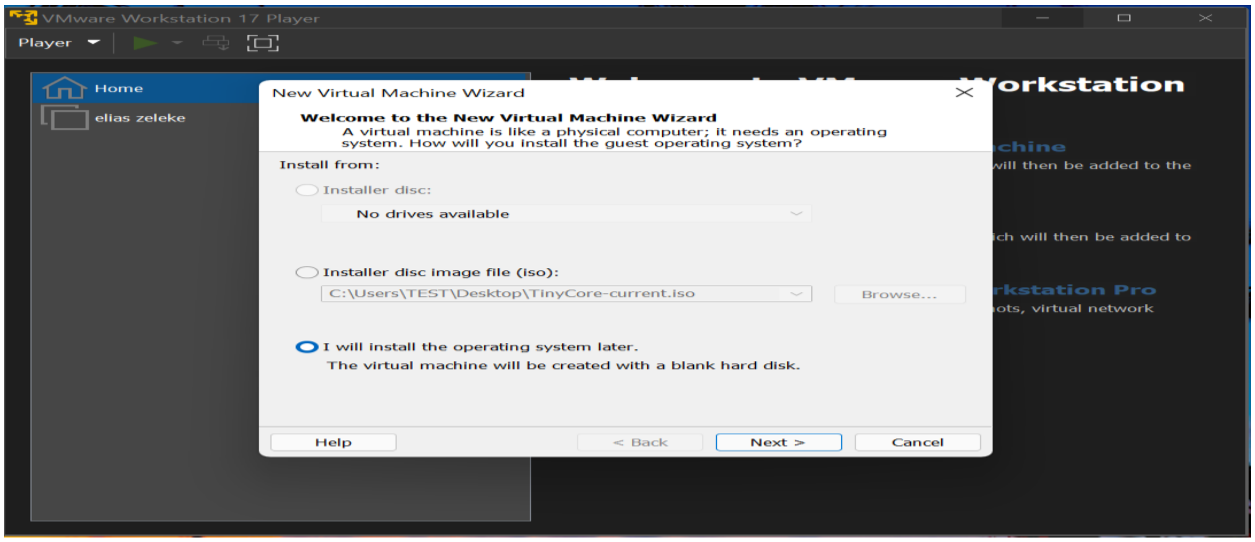
after complete download and installation process:

Step 2: Create a New Virtual Machine

1. Open VMware Workstation Player.

2. Click on "Create a New Virtual Machine".

3. Choose "Installer disc image file (ISO)" and browse the downloaded Tiny Core ISO. The file that used in this step is the file which is downloaded from step1.



New Virtual Machine Wizard

Welcome to the New Virtual Machine Wizard

A virtual machine is like a physical computer; it needs an operating system. How will you install the guest operating system?

Install from:

Installer disc:

No drives available

Installer disc image file (iso):

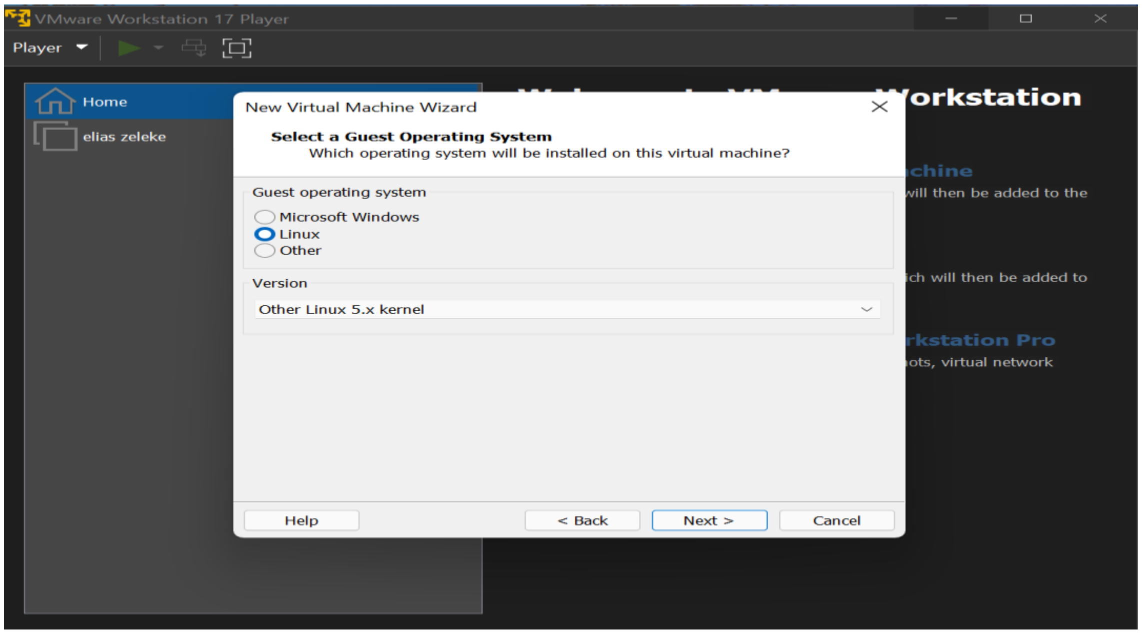
C:\Users\TEST\Desktop\TinyCore-current.iso

[✓ I will install the operating system later.]

The virtual machine will be created with a blank hard disk.

4. Select: *Guest OS: Linux*

Version: other linux 5.x kernel

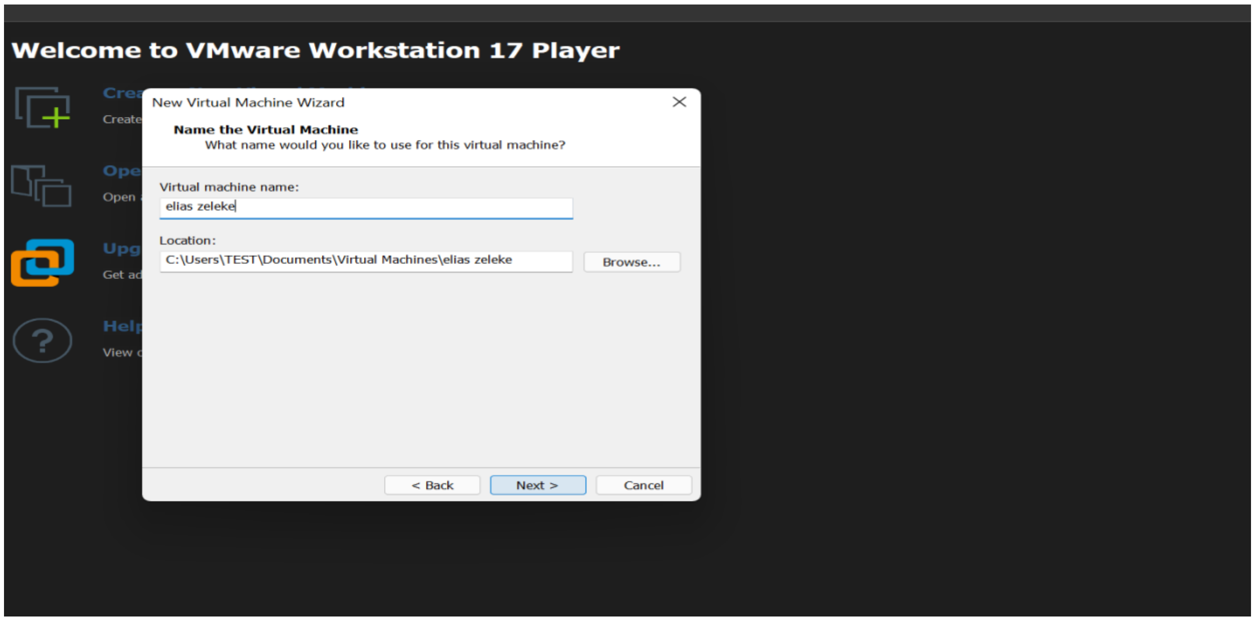


Select a Guest Operating System
Which operating system will be installed on this virtual machine?

Guest operating system
Microsoft Windows [✓ Linux] Other

Version
Other Linux 5.x kernel

5. fill the virtual machine name by your name


Welcome to VMware Workstation 17 Player
New Virtual Machine Wizard

Name the Virtual Machine
What name would you like to use for this virtual machine?

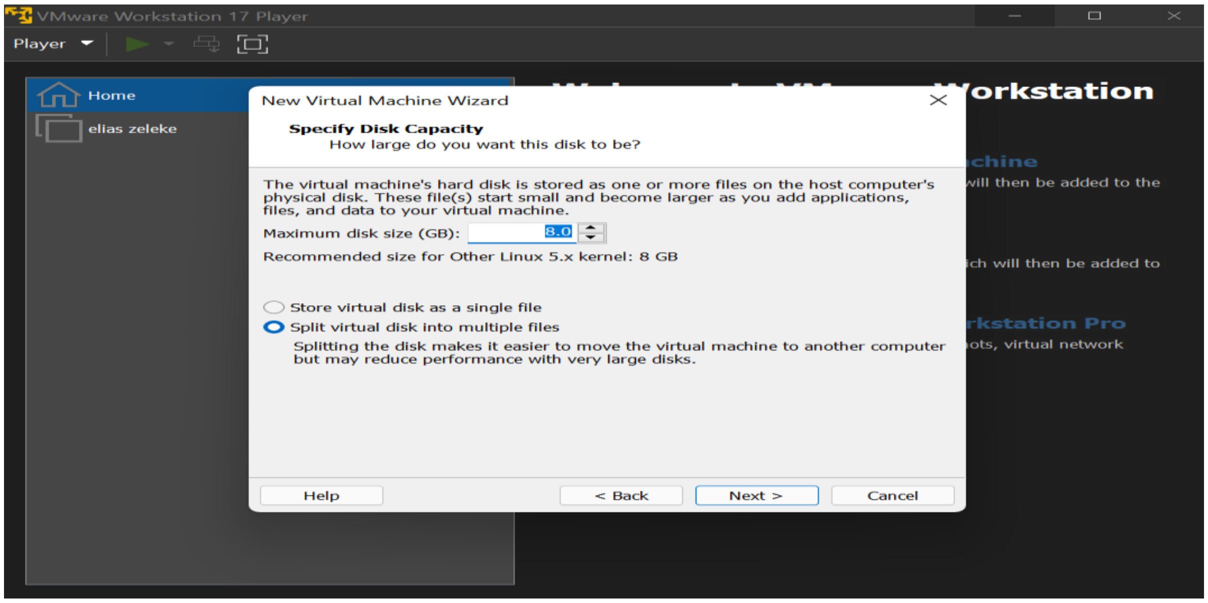
Virtual machine name:
elias zeleke

Location:
C:\Users\TEST\Documents\Virtual Machines\elias zeleke

Browse...
< Back
Next >
Cancel

6. select the maximum disk size

Recommended size of *other linux 5.x kernel* 8GB



Specify Disk Capacity
How large do you want this disk to be?

The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.

Maximum disk size (GB): 8.0

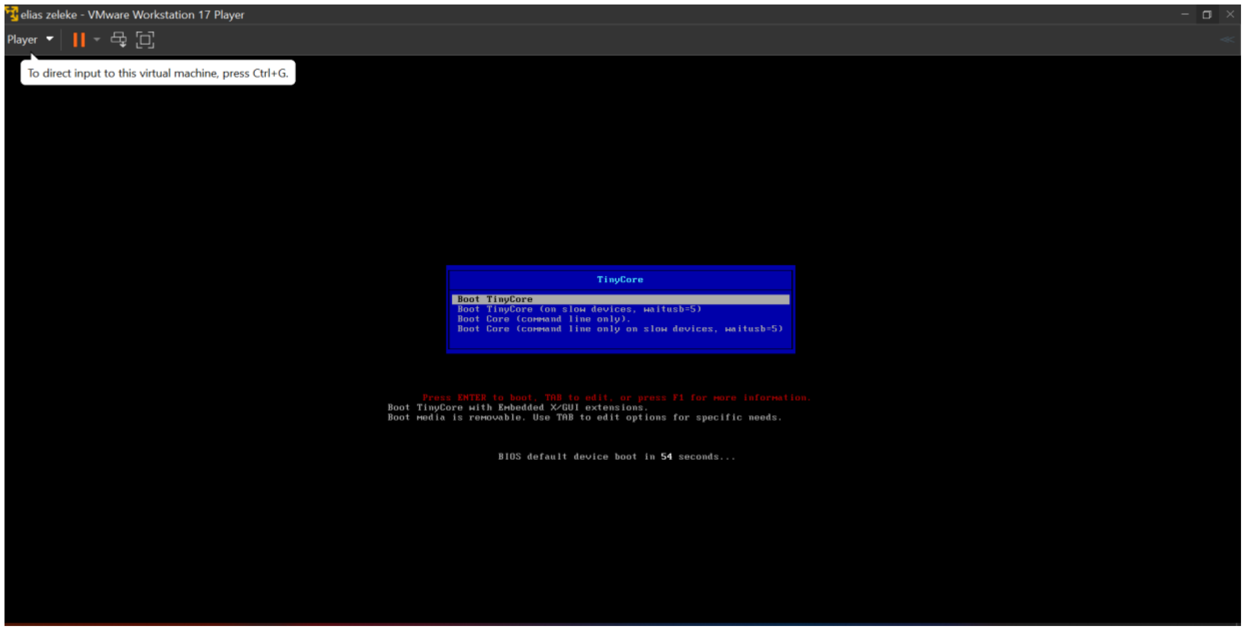
Recommended size for Other Linux 5.x kernel: 8 GB

( ) Store virtual disk as a single file
[✓ Split virtual disk into multiple files]
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

7. finally select finish button.

Step 3: open the created virtual machine

1. tap enter to boot it or bios default device boot it after 1min.

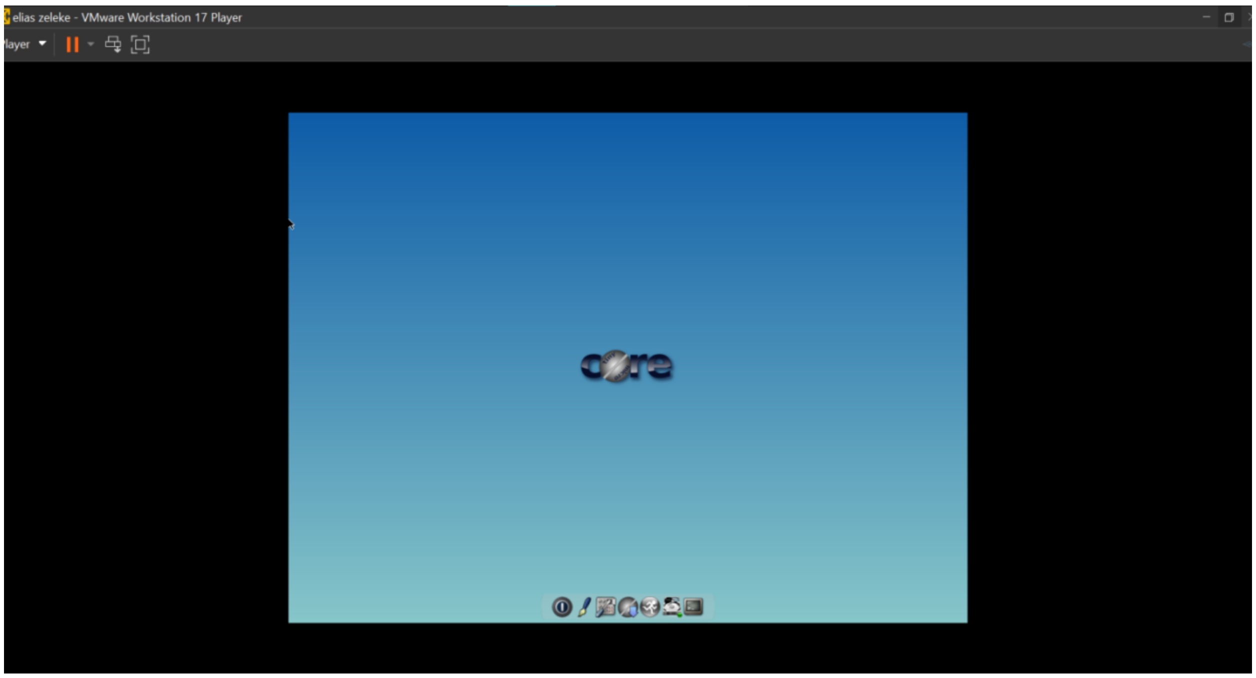

elias zeleke - VMware Workstation 17 Player
Player
To direct input to this virtual machine, press Ctrl+G.

```
                    TinyCore
 Boot TinyCore
 Boot TinyCore (on slow devices, waitusb=5)
 Boot Core (command line only).
 Boot Core (command line only on slow devices, waitusb=5)


     Press ENTER to boot, TAB to edit, or press F1 for more information.
 Boot TinyCore with Embedded X/GUI extensions.
 Boot media is removable. Use TAB to edit options for specific needs.


              BIOS default device boot in 54 seconds...
```

2. finally you will see this interface



❖ Issues faced

I did not face any problems during the installation. Before starting the installation and I used the Google search engine to access the official website of Tiny Core Linux. I read the available documentation and researched all the necessary steps required for this assignment.

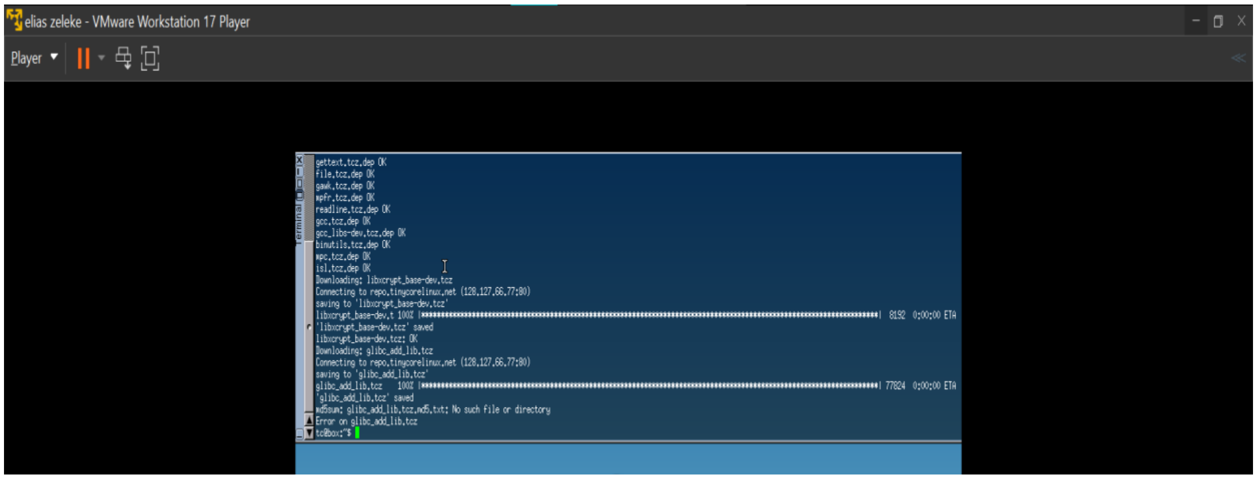
But on the execution of the system call

Even if I gathered all the needed information step by step and followed the procedures strictly as instructed I ***got a server crush or error*** on installing compilers by using commands multiple times. Even I try it on dead line's day but I got the same result.

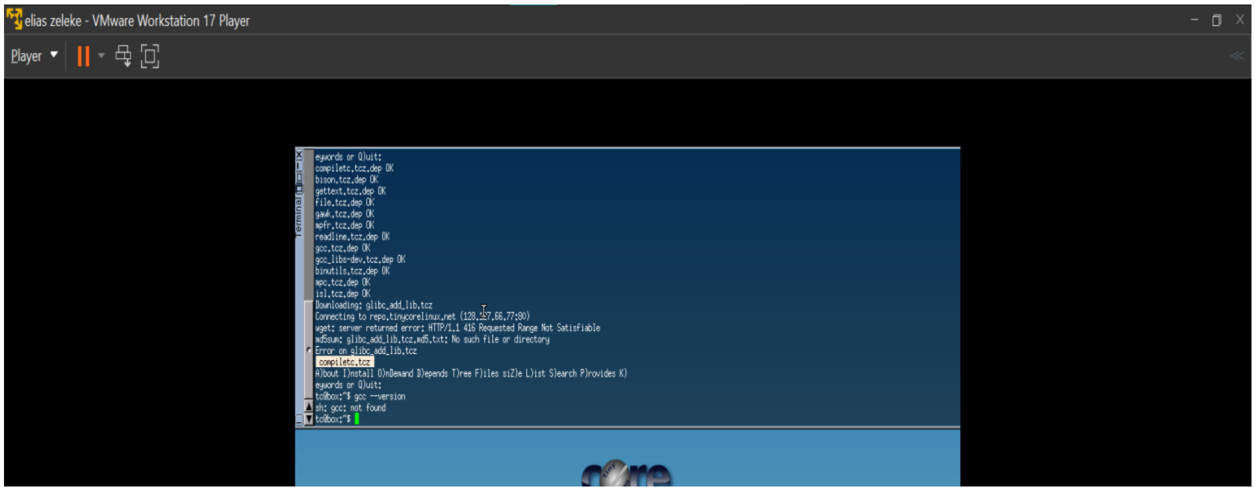
Some of unsuccessful outputs during installations are;

No file directory…

Erron on…



```
gettext.tcz.dep OK
file.tcz.dep OK
gawk.tcz.dep OK
mpfr.tcz.dep OK
readline.tcz.dep OK
gcc.tcz.dep OK
gcc_libs-dev.tcz.dep OK
binutils.tcz.dep OK
mpc.tcz.dep OK
isl.tcz.dep OK
Downloading: libxcrypt_base-dev.tcz
Connecting to repo.tinycorelinux.net (128.127.66.77:80)
saving to 'libxcrypt_base-dev.tcz'
libxcrypt_base-dev.t 100% |********************************************************************************|  8192  0:00:00 ETA
'libxcrypt_base-dev.tcz' saved
libxcrypt_base-dev.tcz: OK
Downloading: glibc_add_lib.tcz
Connecting to repo.tinycorelinux.net (128.127.66.77:80)
saving to 'glibc_add_lib.tcz'
glibc_add_lib.tcz    100% |********************************************************************************| 77824  0:00:00 ETA
'glibc_add_lib.tcz' saved
md5sum: glibc_add_lib.tcz.md5.txt: No such file or directory
Error on glibc_add_lib.tcz
tc@box:~$
```



```
eywords or Q)uit:
compiletc.tcz.dep OK
bison.tcz.dep OK
gettext.tcz.dep OK
file.tcz.dep OK
gawk.tcz.dep OK
mpfr.tcz.dep OK
readline.tcz.dep OK
gcc.tcz.dep OK
gcc_libs-dev.tcz.dep OK
binutils.tcz.dep OK
mpc.tcz.dep OK
isl.tcz.dep OK
Downloading: glibc_add_lib.tcz
Connecting to repo.tinycorelinux.net (128.127.66.77:80)
wget: server returned error: HTTP/1.1 416 Requested Range Not Satisfiable
md5sum: glibc_add_lib.tcz.md5.txt: No such file or directory
Error on glibc_add_lib.tcz
 compiletc.tcz
A)bout I)nstall O)nDemand D)epends T)ree F)iles siZ)e L)ist S)earch P)rovides K)
eywords or Q)uit:
tc@box:~$ gcc --version
sh: gcc: not found
tc@box:~$
```

Solutions

I was try to install by different commands, I try to go to options and install manually,

I was try to see youtube videos to solve the problem.

Unfortunately I can't solve the problem but manually I studied how to compile and run it on terminal by referencing different documentations. You can see it from the last part in execuv system calls.

❖ File system support

File system Support in Tiny Core Linux

Supported types:

1. NTFS

Supported? **Yes (via extension)**

Reason: NTFS is a Windows file system. Tiny Core doesn't support it natively to stay lightweight, but it can be added using the ntfs-3g extension for read/write support.

2. FAT32

Supported? **Yes**

Reason: Fully supported because it is simple, lightweight, and widely used on USBs. Compatible across operating systems and fits Tiny Core’s minimal design.

3. exFAT

Supported? **Yes (via extension)**

Reason: Microsoft file system not included by default, but support can be added via fuse-exfat and exfat-utils extensions for handling newer USB storage.

4. ext4

Supported? **Yes (recommended)**

Reason: Native Linux file system with full support. It Offers journaling , performance, stability and persistence features. Best fit for Tiny Core Linux.

5. Btrfs

Supported? **Yes (via extension)**

Reason: Supports advanced features like snapshots, but is resource-heavy. Not ideal for Tiny Core's minimal environment unless specifically needed.

Unsupported types:

1. ZFS

Supported? **No**

Reason: Very heavy and complex. Licensing issues prevent it from being included in most Linux distros by default. Not suitable for Tiny Core's design goals.

2. HFS+

Supported? **No**

Reason: A macOS file system with limited Linux support. Tiny Core does not include support due to minimal use and complexity.

3. APFS

Supported? **No**

Reason: Apple's modern proprietary file system. Closed source with no official Linux support. Not available in Tiny Core.

Advantages of Tiny Core Linux

1. **Minimal Size**:
   - Core version is only **11MB**, making it extremely lightweight and ideal for low-resource systems or embedded environments.
2. **Fast Boot Time**:
   - Due to its small footprint, it **boots very quickly**, even on older hardware.
3. **Modular Design**:
   - Allows you to **add only the features you need** (via extensions), making it highly customizable.
4. **Runs in RAM**:
   - The OS can run entirely in memory, making it **very fast and responsive** after boot, and **reducing disk usage**.
5. **Great for Learning**:
   - Helps users understand the **core elements of Linux** and system internals due to its simplicity and transparency.
6. **Active Community and Documentation**:
   - Despite its size, there is an active community and good documentation available.

Disadvantages of Tiny Core Linux

1. **Not Beginner-Friendly**:
   - Requires **command-line knowledge** and manual configuration, which can be difficult for beginners.
2. **Limited Hardware Support**:
   - Because it doesn't include many drivers by default, **some hardware may not work out-of-the-box**.
3. **No GUI by Default (Core Version)**:
   - You must manually install a desktop environment if needed (CorePlus includes GUI options though).
4. **Software Compatibility**:
   - Some mainstream software may not run easily without **manual dependency resolution** or may not be available in the extension repository.
5. **Persistence Requires Setup**:
   - Since it runs in RAM by default, any changes are lost on reboot unless **persistence is configured properly**.

❖ *Conclusion*

This assignment gave a deep and clear look into Tiny Core Linux a small but powerful operating system that shows how simplicity can lead to great efficiency. By exploring its background, file system support, system requirements, and main features, we can see that Tiny Core is more than just a small Linux version. It’s a smart system that gives users full control while keeping things lightweight and fast.

When we looked at the supported and unsupported file systems, we saw how Tiny Core chooses to support only the most useful and lightweight ones. File systems like ext4 work well with it and are fully supported. Others like NTFS and exFAT are available through extra tools, while heavy systems like ZFS and APFS are not supported   which makes sense because Tiny Core wants to stay simple and efficient.

The main goals of this assignment to understand how Tiny Core is built, how it uses system resources, and how it runs processes were successfully met through both study and analysis. The small hardware and software needs of Tiny Core make it perfect for older or limited devices. At the same time, we saw that while it’s fast and flexible, it may not be easy for beginners.

In short, Tiny Core Linux shows us that an operating system doesn’t need to be big to be strong. It’s perfect for students, developers, and anyone who wants to really understand how Linux works from the inside out. Tiny Core is not just another Linux it’s a valuable tool for learning and exploring how operating systems are designed and used.

Future outlook / recommendation

Tiny Core Linux will likely continue to be a useful and important tool for people who need a fast, lightweight, and customizable operating system. As technology grows, more powerful systems become common, but there is still a strong need for small systems like Tiny Core especially in areas like education, testing environments, virtual machines, and older or low-resource hardware.

In the future, Tiny Core could improve by adding easier setup tools, better documentation for beginners, and smoother support for more file systems and devices without losing its small size and speed. If these improvements are made, it can reach a wider audience and become more useful in real-world projects, especially in embedded systems or cloud environments.

Recommendations:

For learners: Use Tiny Core to understand Linux better — it's a great tool to explore how operating systems work under the hood.

For developers: Try it in virtual machines or low-resource projects to save space and speed up performance.

For the Tiny-Core team: Continue improving user experience while keeping the system minimal and fast. Adding guided setup tools and better hardware support could make it even more powerful. Tiny Core shows that simple systems can still be strong, and with the right improvements, it has a bright future as a learning tool and lightweight -OS for special tasks.

2. What is Virtualization?

Virtualization is technology that allows a physical computer to be used to run multiple OS or applications as if on independent separate machines.

How it works?

Virtualization software also possesses a virtual environment in the guise of a virtual machine (VM) that emulates the hardware of an actual computer. The VM can also then run its own OS and applications separate from the other VMs and host OS.

Types:

Full Virtualization: The VM runs an unchanged standard OS using hardware simulation.

Para virtualization: The paravirtualizing guest OS speaks directly to the virtualizer in order to get enhanced performance.

OS-level Virtualization: Multiple isolated OS environments are run on the same kernel, with resources shared.

Why Use Virtualization?

Resource Efficiency:

Virtualization makes it possible to utilize hardware resources better with multiple VMs all sharing one physical machine's CPU, memory space, and storage.

Cost savings

Virtualization reduces hardware costs and operational cost by consolidating servers and applications.

Scalability and Flexibility

Virtualization allows easy scaling of resources and rapid deployment of new applications and VMs.

Easier administration

VMs are easy to manage and migrate, making IT operations simpler and downtime shorter.

Testing and Development:

Virtualization provides an isolated and secure environment to test new applications and software.

Disaster Recovery

VMs are easy to back up and restore in case of hardware failure to achieve business continuity.

How is Virtualization Implemented?

Virtualizing software

These include software such as VMware, Microsoft Hyper-V, and QEMU used to create and manage VMs.

Hyp

Hypervisor is a software layer responsible for facilitating communication between the host OS and the VMs through the provision of resource allocation and isolation.

Virtual Hardware:

Virtual

Virtualization software emulates multiple hardware components such as CPUs, memory space, storage devices, and network interfaces in each VM

Implement system calls

From the given table, my system call is: execve – it executes a program, passing an array of arguments.

What is execve?

execve is a system call in Linux that is used to run a new program from your code. It replaces the current running program with the new one you want to run.

How it works:

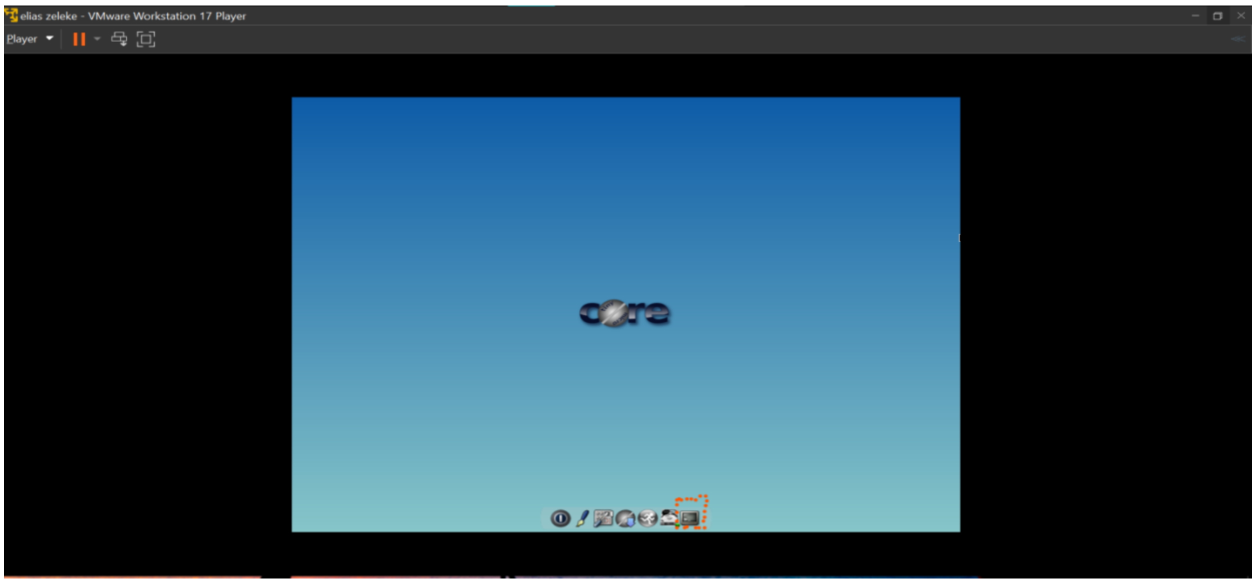
You tell it the path of the program you want to run (like /bin/ls).

You give it a list of arguments (like -l, /).

It starts that program, and your current program does not continue after that unless something goes wrong.

I finished the installation step and viwed its interface, now we continue from that and do the following:

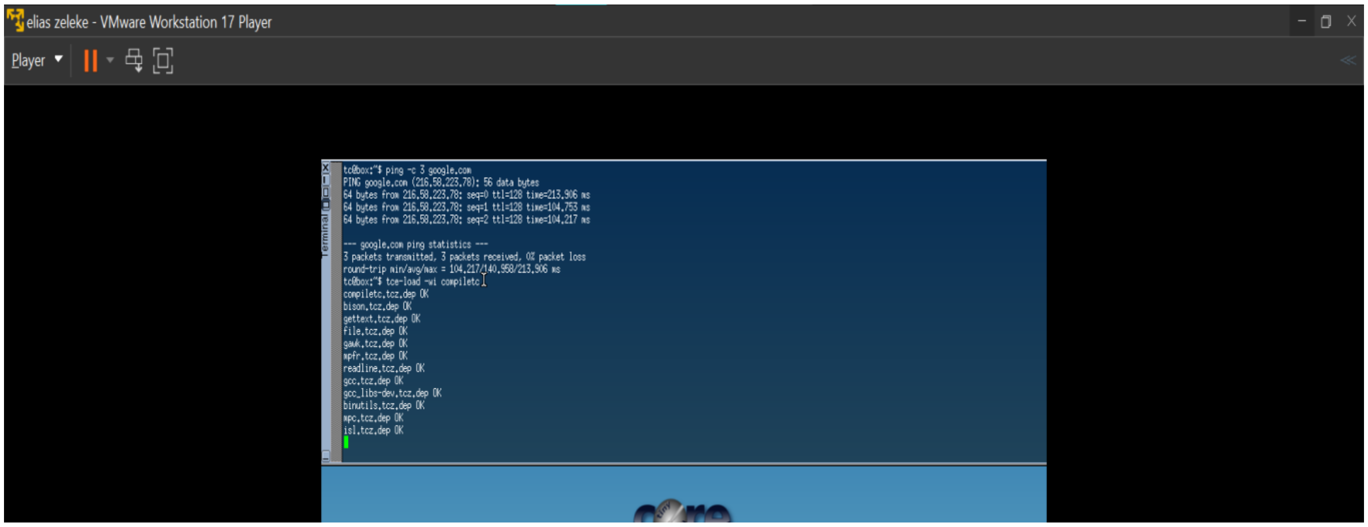
Open the terminal: indicated by red points surrounding it.



Step 1: Install Development Tools

Before everything check the connection then Write **tce-load -wi compilec** to install.

Tiny Core Linux doesn't include development tools by default, so the first step is to install them using the built-in package manager.



```
tc@box:~$ ping -c 3 google.com
PING google.com (216.58.223.78): 56 data bytes
64 bytes from 216.58.223.78: seq=0 ttl=128 time=213.906 ms
64 bytes from 216.58.223.78: seq=1 ttl=128 time=104.753 ms
64 bytes from 216.58.223.78: seq=2 ttl=128 time=104.217 ms

--- google.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 104.217/140.958/213.906 ms
tc@box:~$ tce-load -wi compiletc
compiletc.tcz.dep OK
bison.tcz.dep OK
gettext.tcz.dep OK
file.tcz.dep OK
gawk.tcz.dep OK
mpfr.tcz.dep OK
readline.tcz.dep OK
gcc.tcz.dep OK
gcc_libs-dev.tcz.dep OK
binutils.tcz.dep OK
mpc.tcz.dep OK
isl.tcz.dep OK
```

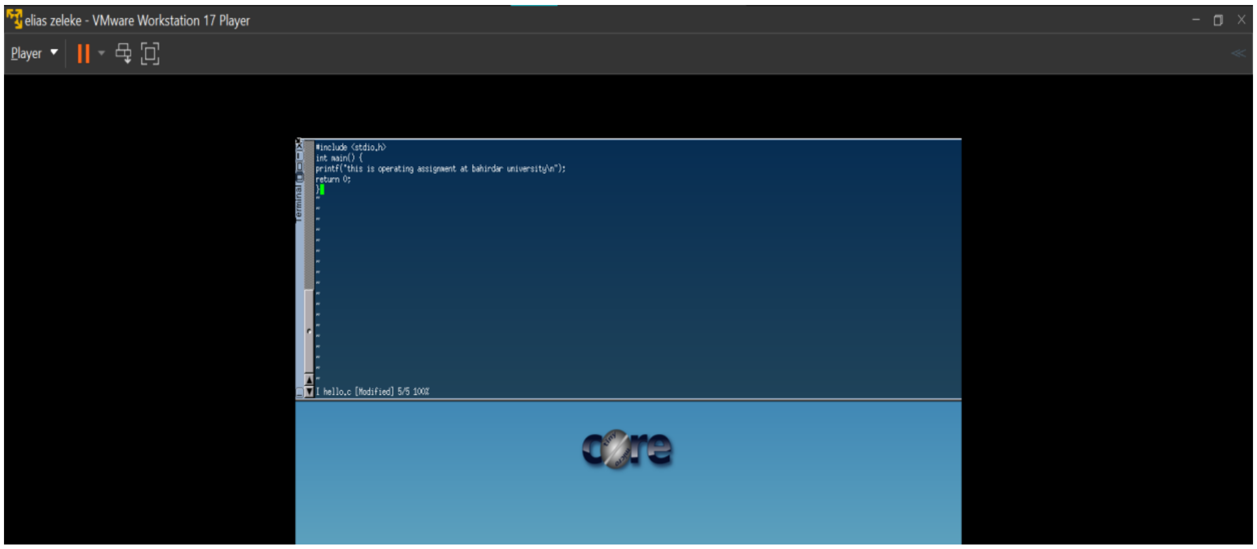
Step 2: Create and Compile Source Files

❖ *Program execution: Program to display a message*

to Create the file on vi write: **vi hello.c**

Write this code: **#include <stdio.h>**

```
int main() {
printf ("This is Operating assignment at Bahir Dar University/n");
return 0;
}
```



```
#include <stdio.h>
int main() {
printf("this is operating assignment at bahirdar university\n");
return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
I hello.c [Modified] 5/5 100%
```

After type your code press ***esc.***

Then type : ***wq*** and press ***enter*** which mean save and quite.

If the compiler is working, here's how to run and compile the c program :

1. save the code as a .c file

eg. os_assignment.c

2. compile the program using for instance gcc compiler

eg. gcc os_assignment.c –o os_assignment

3. now run the program

eg. .os/_assignment

Expected Output: **this is Operating assignment at bahir Dar University**

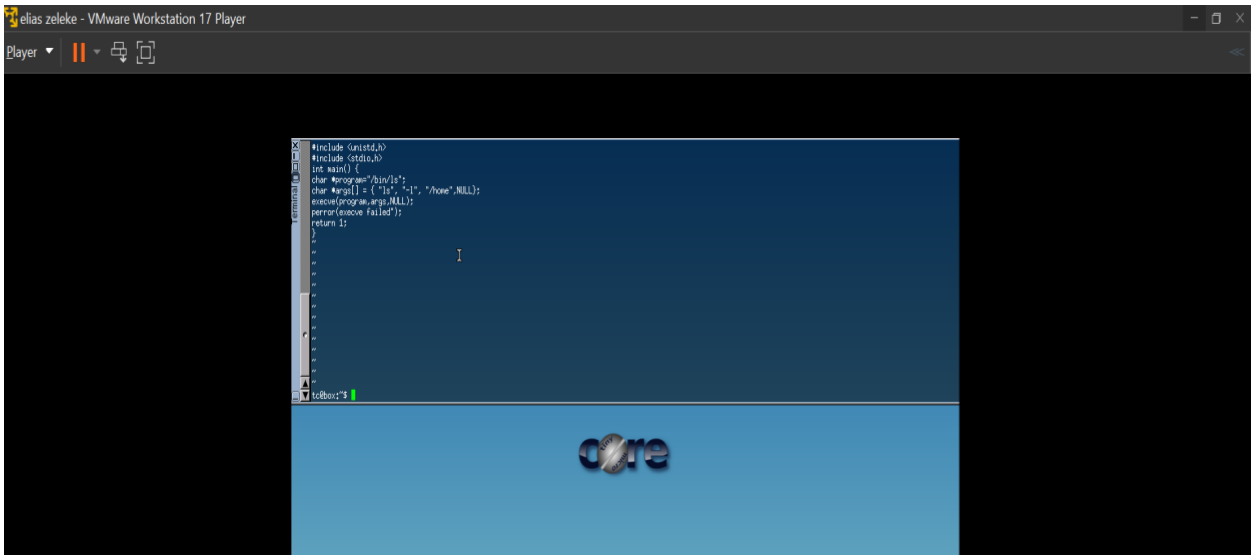
❖ *passing an array of argument using execve()*

to Create the file write this: **vi execdemo.c**

press i to write a program code

write this code : **#include <unistd.h>**

```
#include <stdio.h>
int main() {
char *program = "/bin/ls";
char *args[] = { "ls", "-l", "/home", NULL };
execve(program, args, NULL);
perror("execve failed");
return 1;
}
```



```
#include <unistd.h>
#include <stdio.h>
int main() {
char *program="/bin/ls";
char *args[] = { "ls", "-l", "/home",NULL};
execve(program,args,NULL);
perror(execve failed");
return 1;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
tc@box:~$
```

Then, press Esc

Type this **:wq** and hit Enter to save and exit

If the compiler is working, here's how to run and compile the c program :

1. save the code as a .c file

eg. execuv_assignment.c

2. compile the program using for instance gcc compiler

eg. gcc execuv_assignment.c

3. now run the program

eg. ./a.out

Expected Output: the actual output will depend on what is in your /home directory but the format will look like this:

eg. If /home contains 2 items then output will be:

```
      /home:

     total 8

 drwxr-xr-x 2 root root 4096 Apr 24 09:59 user1

-rw-r--r-- 1 root root   10 Apr 24 10:00 notes.txt
```

----Thank you----