OSSP Individual Assignment

Name: Abel Mitiku

ID: 1601024

Section: B

Installed OS:

Rocky Linux 9.5 in **VMware Workstation**

Rocky Linux is an open-source, free, enterprise-class Linux operating system. It's a bug-for-bug compatible alternative to Red Hat Enterprise Linux (RHEL), intended for use in production servers and data centers.

Advantages Of Rocky Linux Server:
- stable: great for long-term, stable environments.
- Free: the product is free and open-source.
- Great for servers: good for web servers, databases, virtualization, file servers, etc.
- Large community: Active development and support.
- Security focused: regular updates.

Objective

The objective of this project is to install and configure Rocky Linux in a virtual environment, and to implement a system call in C that replaces the current process with a new one using the `exec` system call. This helps in understanding low-level process management and the practical use of system calls in Linux.

Requirements:

Hardware:

- Intel/AMD CPU

- 4 GB RAM

- 20 GB Disk space

- VMware Workstation installed

Software:

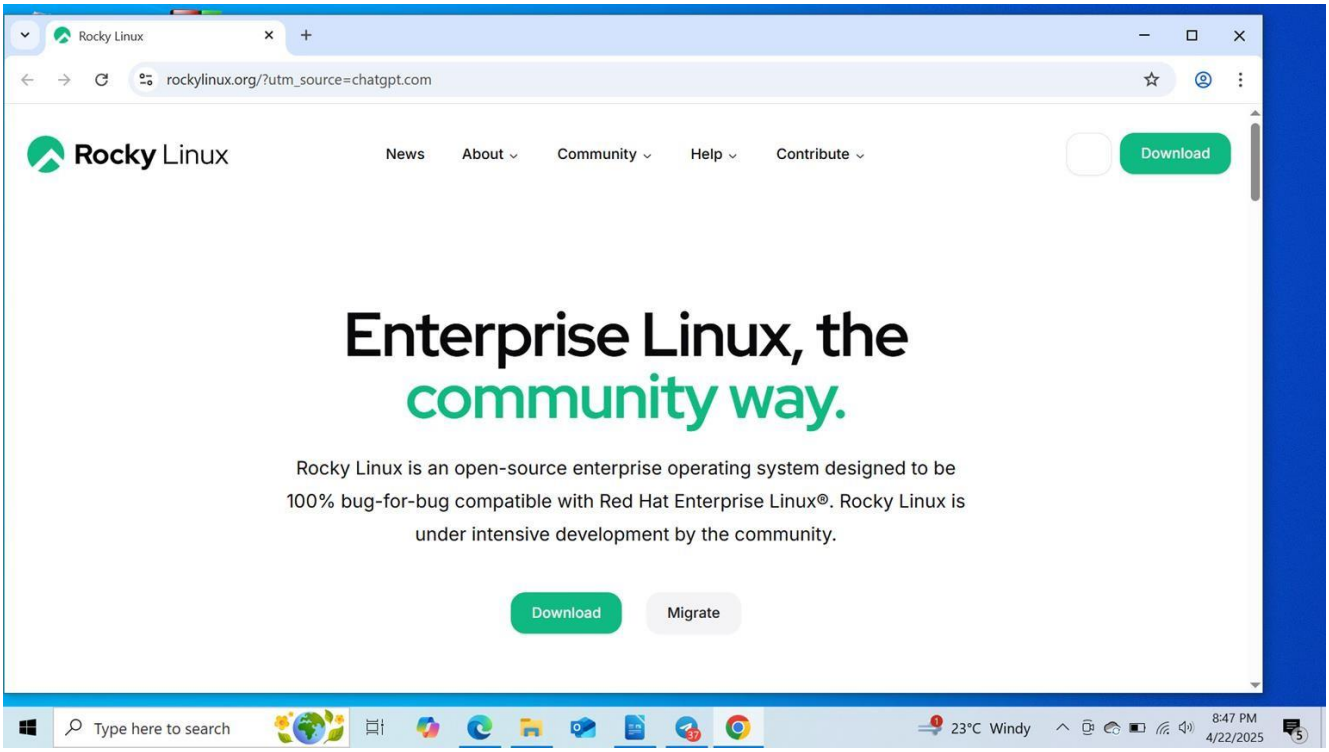
- Rocky Linux 9.5 ISO

- GCC (compiler)

- Basic terminal tools (vim/nano, dnf, etc.)

Installation Steps:

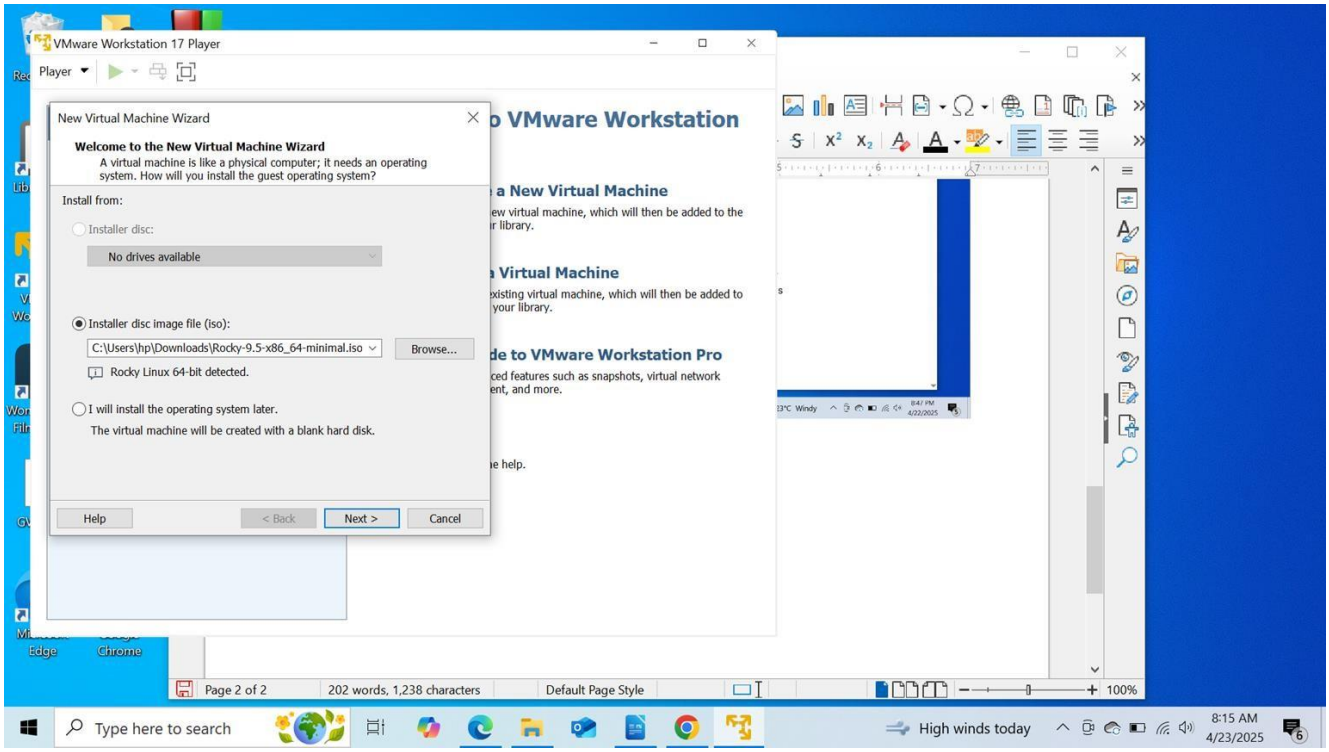
1,Downloaded Rocky Linux ISO from official site. (I downloaded the minimal version)



I chose to install the minimal version of Rocky Linux Server by clicking on the minimal version. I did install the minimal version because the system has all the requirements I needed for my assignment. In addition the system is user friendly and has good graphical user interface (by default) and comes with command line interface. But also a user can install GUI.

2,Created a new virtual machine in VMware:
- OS: Linux, Version: CentOS 8 64-bit

- Disk: 20 GB, RAM: 2 GB

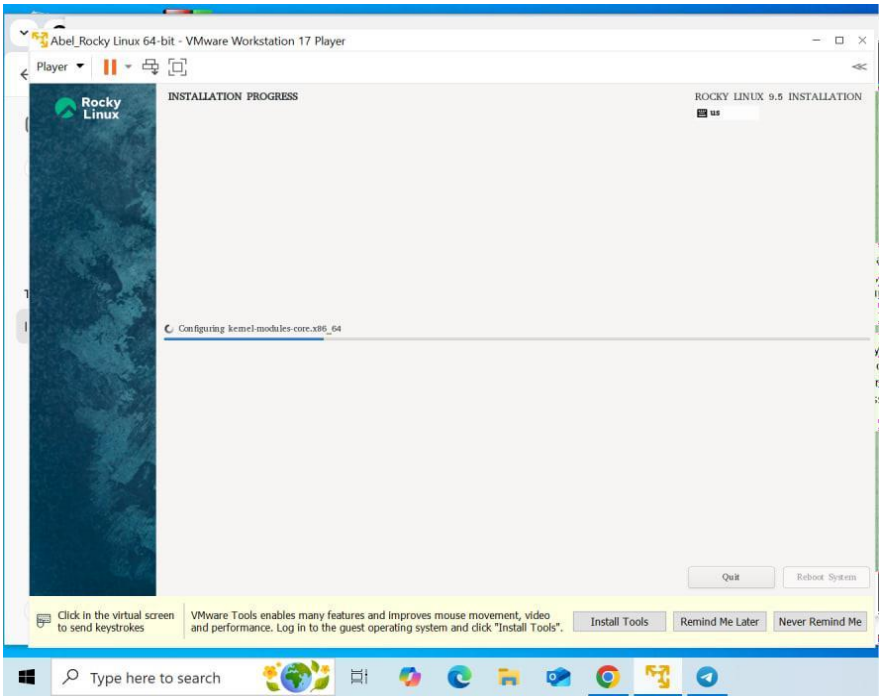


VMware was user friendly virtual machine it did not took me a lot of energy to understand how It actually works. As I double clicked on the VM I clicked on new machine then I chose a disk where I stored the ISO. Then I did set up the VM I scheduled the RAM,CPU,storage device and other resources, to make the VM more efficient and fast loading speed.

4,Followed the GUI installer:

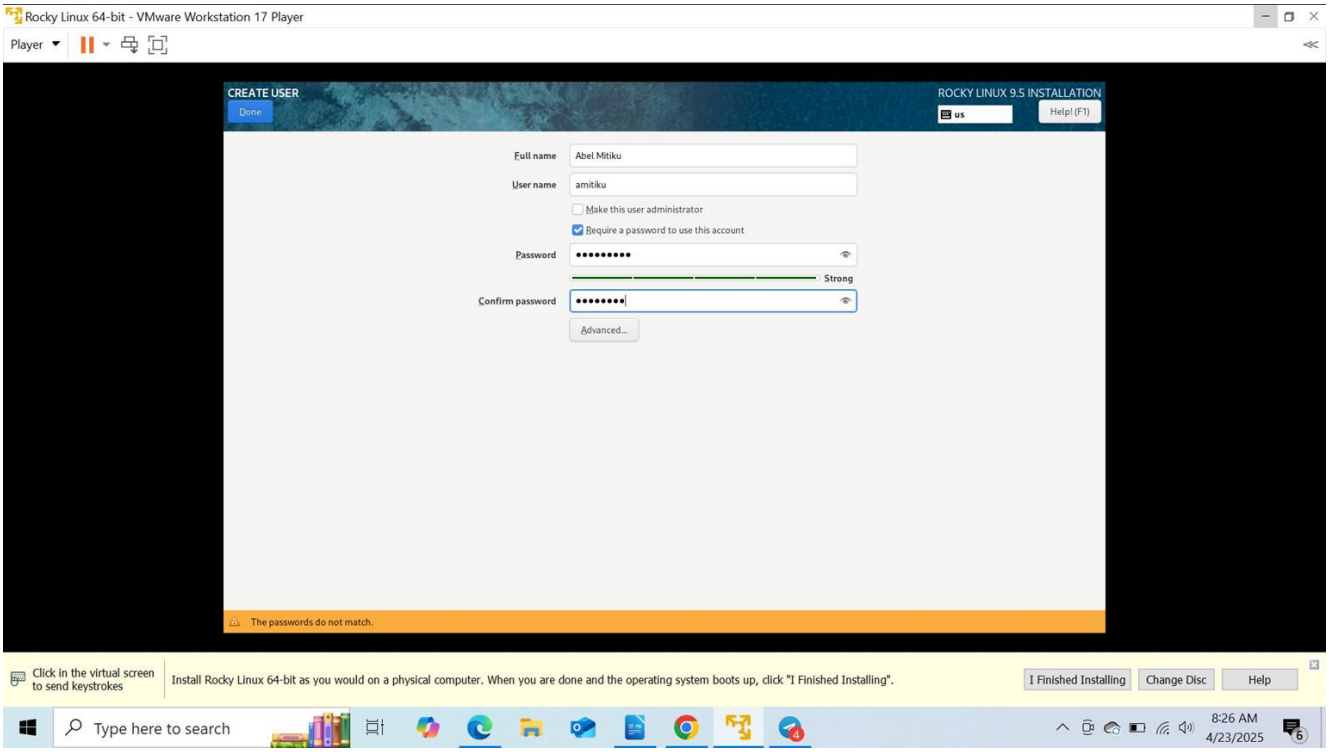
- Chose "Minimal Install"

- Enabled network and set root password



The above snnipet shows the installation process as I expected the installation took a bit long time. As I finished the installation the ISO displayed a page where I can choose a language to preceed the process. The next page was where I can create my VM account. On this page the required information was Full Name,Username and password. I entered the required information and continued my process.

- Here is how I created my VM account I created my virtual machine account.



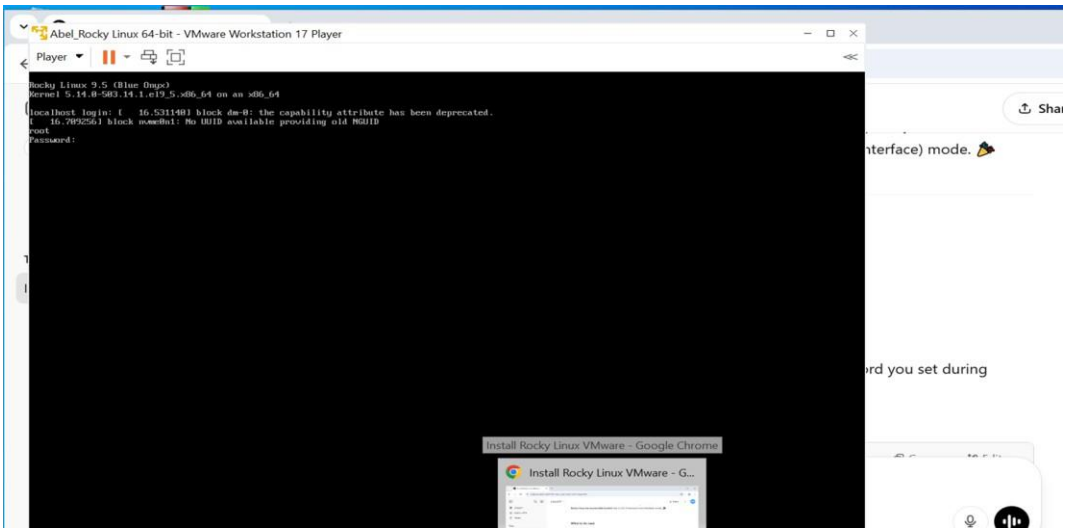
5,Completed installation and rebooted into the OS.

Issues Faced:

1,ISO verification failed.

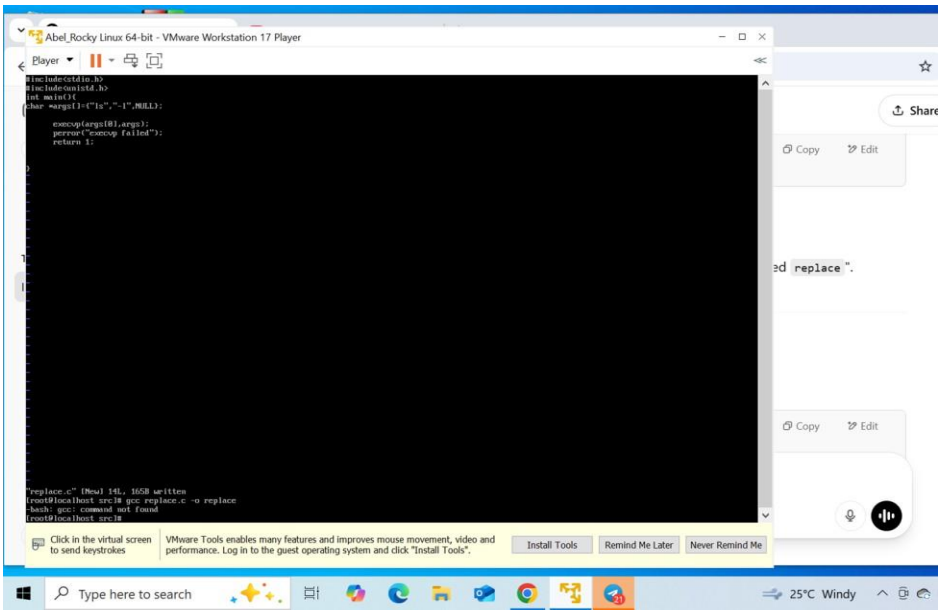
2,Terminal didn’t show password while logging in.

As I entered the required information to log in into my VM I was not able to type my password.



3,Exec program output wasn’t showing.

4,`gcc` not installed by default. (while trying to compile the c file I written to do the exec system call)



5,The file was written or edited using a Windows-based editor (like Notepad), which uses **Windows-style line endings** (`\r\n`). (The c file I used to execute the exec call).

Solutions for the issues I faced:

1,Skipped ISO media check manually.

Started the virtual machine with the Rocky Linux ISO attached.

- Waited for the boot menu to appear (black screen with options).

-On that menu, instead of selecting:

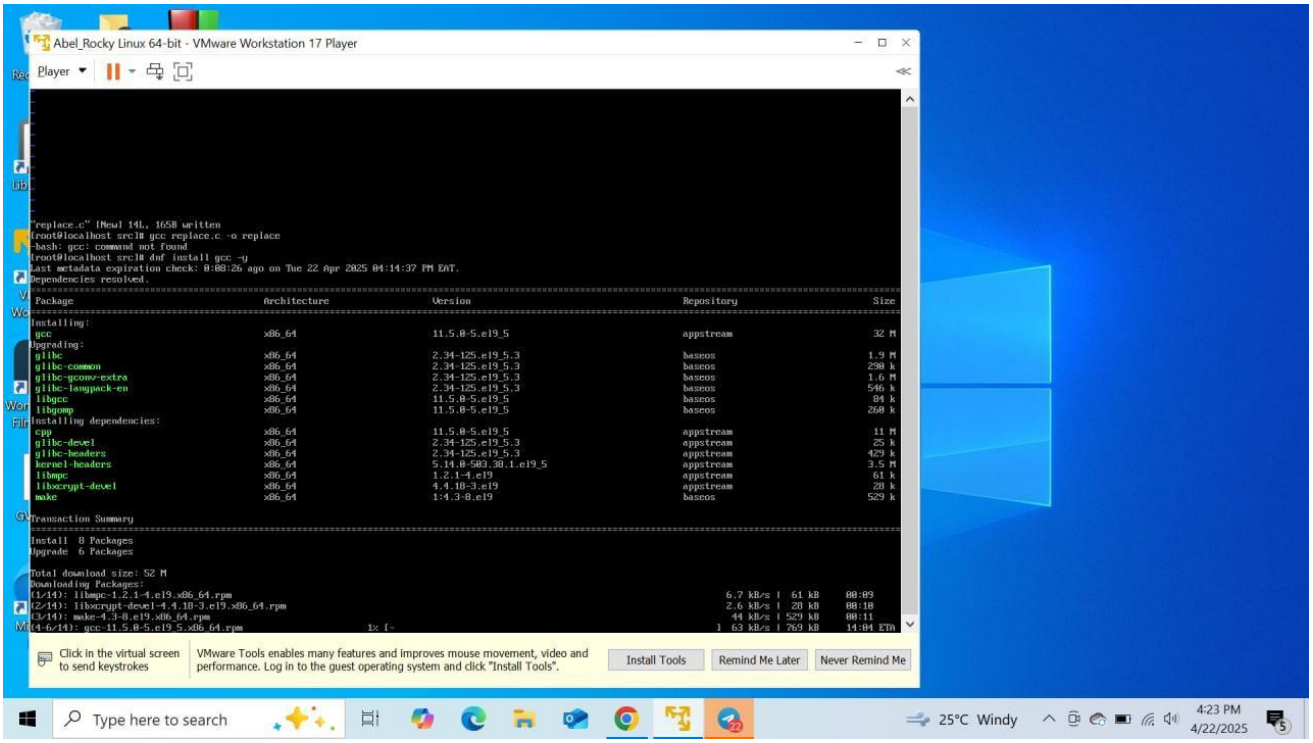
“ Test this media & install Rocky Linux”

I selected:

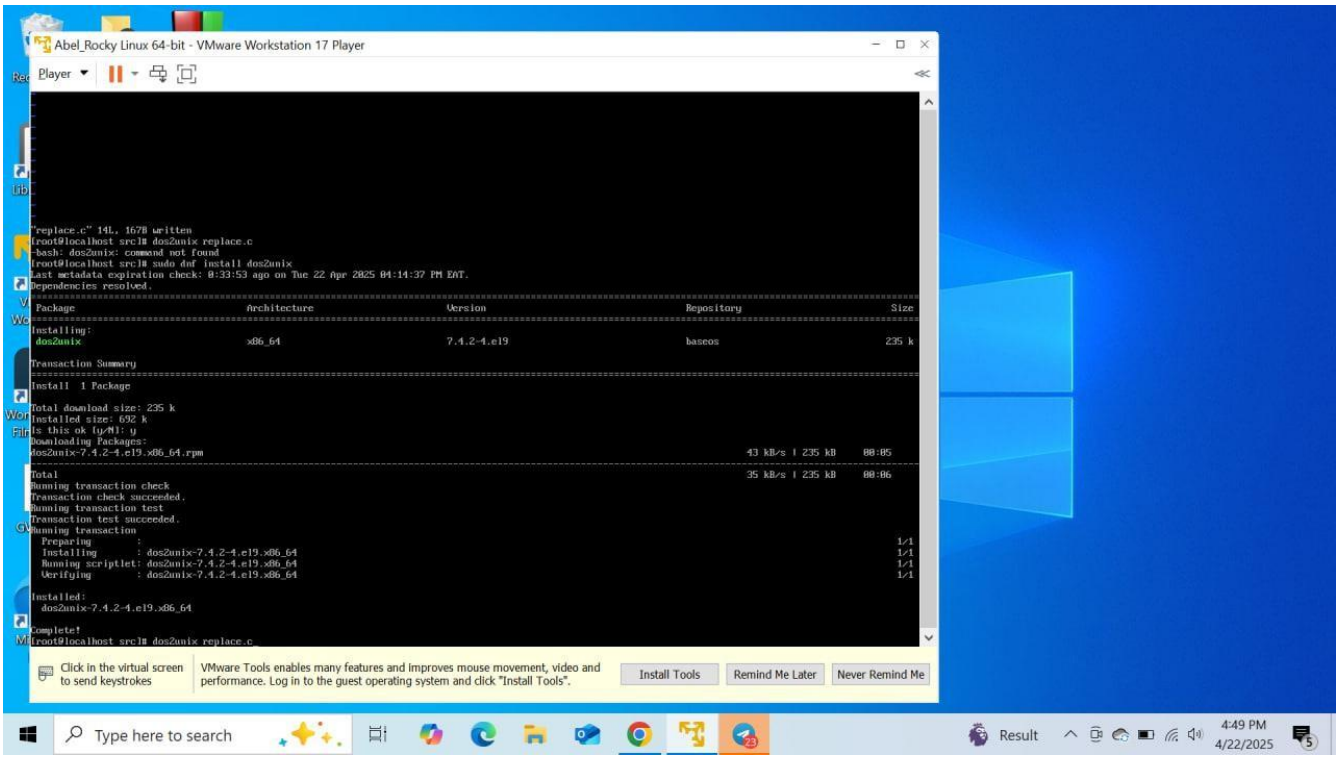
“Install Rocky Linux”

using the arrow key.

2,Installed `gcc` using `dnf install gcc`.



But after installing the gcc I faced another issue which was due to The file was written or edited using a Windows-based editor so I had to download dos2unix.



Then I did the compilation successfully.

3,Understood that Linux hides password input (normal behavior).

4,Used `strace` and `execvp` properly to confirm system call execution.

File support system:

When I installed Rocky Linux 9.5 inside VMware, I went with the default settings in the installer — I didn't manually change partitioning or filesystems. Because of that, the OS automatically used:

ext4 (Fourth Extended Filesystem)

This is the default and most commonly supported filesystem in Linux.

Advantage and Disadvantage of Rocky Linux ISO

- Free and Open Source.
- Enterprise-Grade Stability.
- Long-Term Support (LTS).
- Security-Oriented.
- Large and Growing Community.
- Lightweight (with Minimal Install).

-Not Ideal for Beginners .
- Limited GUI Options by Default.
- Smaller Ecosystem than Ubuntu.
- Some Software Requires Manual Setup.

My conclusion:

In this project, I successfully installed and configured Rocky Linux 9.5 in a virtualized environment using VMware. I explored the use of the `exec` system call in C to replace a running process with a new program, gaining hands-on experience with how system calls work at the process level. Along the way, I learned how to troubleshoot common issues such as line-ending errors (`dos2unix`), media check failures, and output verification using tools like `strace`. Through this process, I also gained a better understanding of Linux filesystems, particularly ext4, and the advantages and limitations of Rocky Linux as a stable, secure, and enterprise-grade operating system. This project has strengthened my foundational knowledge in Linux system programming and prepared me for more advanced work in operating systems and low-level development.

My recommendation and future outlook:

Sustained Growth: Rocky Linux has rapidly become a trusted alternative to CentOS after Red Hat shifted CentOS Stream. Backed by the **Rocky Enterprise Software Foundation (RESF)**, it continues to gain adoption in enterprise environments.

- **Enterprise Adoption:** With its RHEL compatibility and long-term support model, it is likely to remain a **go-to OS for servers, cloud infrastructure, and system-level development**.
- **More Developer Support:** As more developers and system administrators adopt it, **tooling, documentation, and community support** will continue to improve.
- **Stability for Education:** Its reliability makes it an excellent teaching tool for system programming,

especially in academic settings like this project.

What, Why, and How of Virtualization in Modern OS :

Virtualization is a technology that allows the creation of a virtual version of a computing environment, including operating systems, hardware platforms, storage devices, and network resources. In simpler terms, it enables a single physical machine (host) to run multiple isolated virtual machines (guests), each behaving like a full-fledged computer with its own operating system and applications.

The concept is made possible by a software layer known as a hypervisor, which acts as a bridge between the physical hardware and the virtual environments. Each virtual machine (VM) gets its own virtual CPU, memory, disk, and network interface — all simulated by the hypervisor.

The why:

Virtualization plays a critical role in modern computing environments for several key reasons:

1,Efficient Resource Utilization: Instead of dedicating an entire physical machine to one task or system, virtualization allows multiple systems to share the same physical resources, maximizing hardware usage.

2,**Cost Reduction**: Fewer physical servers mean lower hardware and maintenance costs. Organizations can run multiple VMs on a single physical server instead of purchasing multiple machines.

3,Isolation and Security: Each virtual machine operates in its own isolated environment. This means that if one VM crashes or is compromised, others are not affected.

4,Testing and Development: Virtualization provides a safe and controlled environment to test new software, operating systems, or configurations without affecting the host system.

5,Flexibility and Scalability: VMs can be easily created, cloned, paused, or deleted, making it easier to scale environments up or down based on needs.

6,Disaster Recovery and Backup: Virtual machines can be backed up and restored more easily than physical systems, which improves disaster recovery strategies.

The how:

Virtualization is implemented through a key component called a **hypervisor**. There are two main types:

- **Type 1 (Bare-metal Hypervisor)**: Runs directly on physical hardware (e.g., VMware ESXi, Microsoft Hyper-V).
- **Type 2 (Hosted Hypervisor)**: Runs on top of a host operating system (e.g., VMware Workstation, VirtualBox), and is more common for personal or educational use.

In my project, I used **VMware Workstation (Type 2)** to install **Rocky Linux 9.5** on a virtual machine.

Here is how I did the implementation:

Objective:

To write a C program that uses the `exec` system call to replace the current process with a new one. This demonstrates a key concept in Linux system programming: process replacement.

I used `execvp()`, a member of the `exec` family of system calls, which allows replacing the current process image with a new program, such as `ls`, `echo`, or any other executable.

Here is a step by step process of the execution:

1,set up the environment:

I installed **Rocky Linux 9.5** inside a **VMware Workstation** virtual machine. After booting into the minimal installation I logged in as the root user.

2. Creating the C Source File

I used `vi` to create a file named `replace.c (I used nano but didn't work):`

"vi replace.c"

then I wrote this C code:

```
#include <stdio.h>

#include <unistd.h>


int main() {

    char *args[] = {"ls", "-l", NULL};

    execvp(args[0], args);

    perror("execvp failed");

    return 1;

}
```

"This program uses `execvp()` to replace itself with the `ls -l` command. If `execvp()` fails, it prints an error. "

Issues Faced and How I Fixed Them

Issue 1: Compilation Errors on **`#include`** lines

Problem:
When compiling, I got errors like:

```
error: expected identifier or '(' before 'include'
```

Cause:
I mistakenly wrote `include<stdio.h>` without the `#` symbol.

Fix:
I edited the file and corrected the syntax to:

```
#include <stdio.h>
```

Issue 2: compilation still failed after fix

Issue:

Even with the correct syntax, the compiler continued showing the same error.

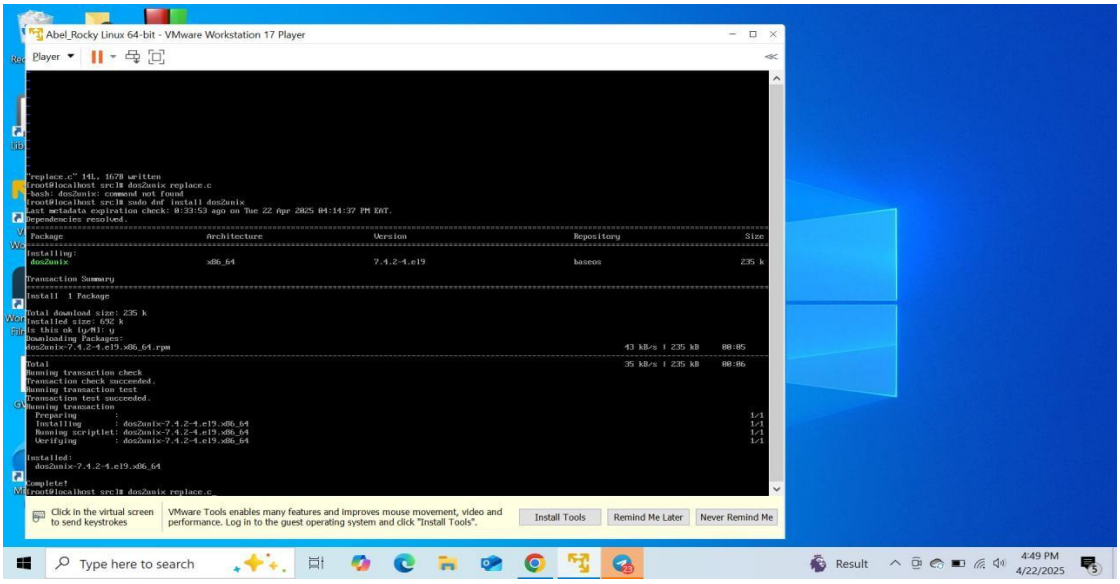
Cause:

`The file had` **Windows-style line endings** `(\r\n) instead of` **Unix-style** `(\n), which confused the compiler.`

Fix:

I did

install dos2unix and converted the file into unix file system for compilation.

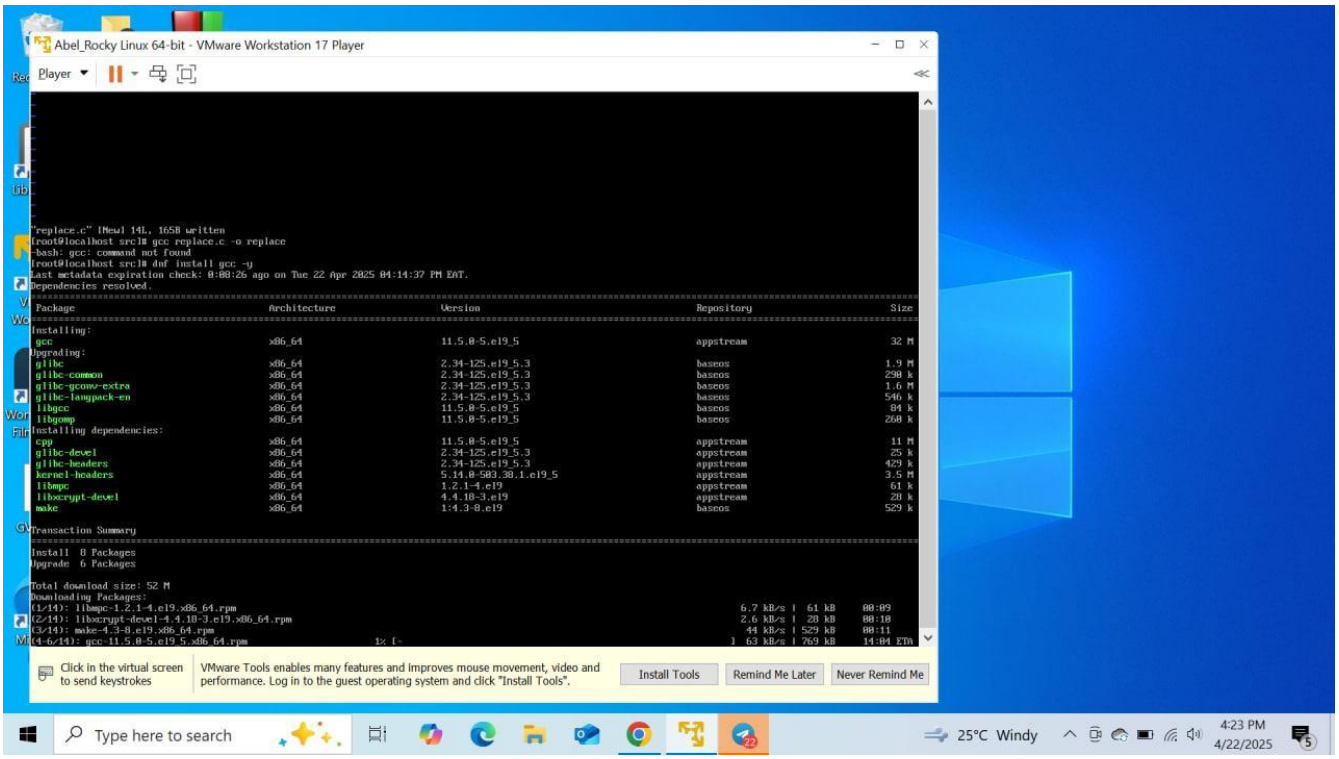


then I did convert the file using “dos2unix replace.c”

Issue: gcc compiler was not installed bt default.

Fix: install the gcc using :

“dnf install gcc -y”



After this, the code compiled successfully using:

`"gcc replace.c -o replace"`

Issue 3: no output when running the code:

Problem:
Running `./replace` produced no visible output, making it seem like the program didn't work.

Cause:
This is actually expected behavior. If `execvp()` is successful, the process is fully replaced, and none of the original code continues to run.

Fix:

I added debug tests by replacing `ls -l` with:

"char *args[] = {"echo", "Hello from exec!", NULL};"
After doing so the output clearly showed:

"Hello from exec!"

I also used:

"strace ./replace"

to confirm that `execvp()` executed correctly and called `execve()` under the hood.

The final working code:

```
#include <stdio.h>
#include <unistd.h>

int main() {
    char *args[] = {"ls", "-l", NULL};
    execvp(args[0], args);
    perror("execvp failed");
    return 1;
}
```