BAHIRDAR UNIVERSITY

Operating System and System Programming

Name: Abraham Abunu Tafere
ID: BDU1600595
section: A
Department: Software Engineering
Year: 2nd Year, 2017 E.C

Operating System: Elive OS

SUBMITTED TO: lec Wendimu Baye

DEADLINE: MAY, 16, 2017 E.C.

Contents

1.Installation of Operating System in Virtual Environment 3

1.1 Introduction (Background & Motivation) 3

1.2 Objectives and Requirements 3

1.3. Steps of Elive os Installation in VMware Workstation Player 5

1.4 Problems and Solutions 12

1.5 Filesystem Support in Elive OS 13

1.6 Disadvantages of Elive 6 Advantages and OS 13

1.7 Future Outlook & Recommendations 15

2. Virtualization in Modern Operating Systems 16

2.1 What is Virtualization? 16

2.2 Why Use Virtualization? 16

2.3 How Virtualization Works 16

3. Implementing System Calls Using Shell Script 17

3.1 Practical Examples in Elive OS 17

1. Installation of Operating System in Virtual Environment

1.1 Introduction (Background & Motivation)

When I started working on this project, I was new to virtualization and had never installed an operating system inside a virtul machine before. I chose Elive OS because it's lightweight and runs well on older systems perfect for testing in a virtual setup.
During the project, I learned how virtualization lets me run multiple systems on the same computer without affecting the host machine. This was especially helpful while experimenting and testing features, as I could mess around with settings without any real risk.
Overall, this experience helped me understand how virtualization works in practice, not just theory. It also gave me the confidence to explore more Linux distributions in virtual environments.
This project centers the insta llation and evaluation of **Elive OS**, a Debian-basedLinux distribution, within a virtualized environment. Elive I s recognized for its lightweight footprint**,** low hardware requirements**,** and visually striking Enlightenment desktop environment, making it a compelling option for both older machines and users seeking a streamlined, aesthetically pleasing Linux experience.
The project not only guid users through the installation of Elive OS in tools like VirtualBox or VMware but also investigates its performance usability and system behvior under virtualization.

I was motivated to work on this project because I wanted to improve my skills in using Linux and understand how operating systems work behind the scenes. Virtualization was something I had heard about, but I had never tried it myself. The idea of running a full operating system inside another one was exciting to me it felt like having a computer inside my computer.
I also chose this topic because I know that in fields like software development and system administration work with different operating systems is important. Being able to set up and test systems without needing multiple computers seemed like a valuable skill to learn. Plus, it gave me a chanc to try out Elive OS, which I had never used before

1.2 Objectives and Requirements

a. Project Goals (Objectives)

My main goal wih this project was to get handson experience installing and using a Linux distribution in a virtual environment. I decided to use Elive OS because it’s lightweight and visually unique, and I wanted to see how well it would run on limited system resources using VMware Workstation Player

Here’s what I focused on during the project:

- **Successfully installing Elive OS** on a virtual machine and noting the steps clearly.
- **Learning what kind of hardware and software setup is needed** to run Elive smoothly in a virtual environmet.

- **Solving real problems I encountered** like screen resolution issues and system lag during the setup process.
- **Testing ho well different filesystems like ext4 or FAT32 work** especially when used in a virtual machine
- **Listing what I liked and didn't like about Elive OS** especially from the perspective of a student or someone new to Linux.
- **Sharing ideas for improvements** both for developers and anyone trying Elive in an educational seting.

b. What I Needed (Requirements)

To run Elive OS, here's what I used:

- **Processor**: Dual-core or better. I had a basic Intel i3, and it worked fine.
- **RAM**: I gave the virtual machine 2 GB, which kept it responsive.
- **Disk space**: 20 GB was enough for installation and testing.
- **Display**: A basic monitor with 1024x768 resolution worked well.

Software used:

- **VMware Workstation Player** (free version)
- **Elive OS ISO** file (downloaded from the official site)
- **Host system**: I ran everything on my Windows laptop

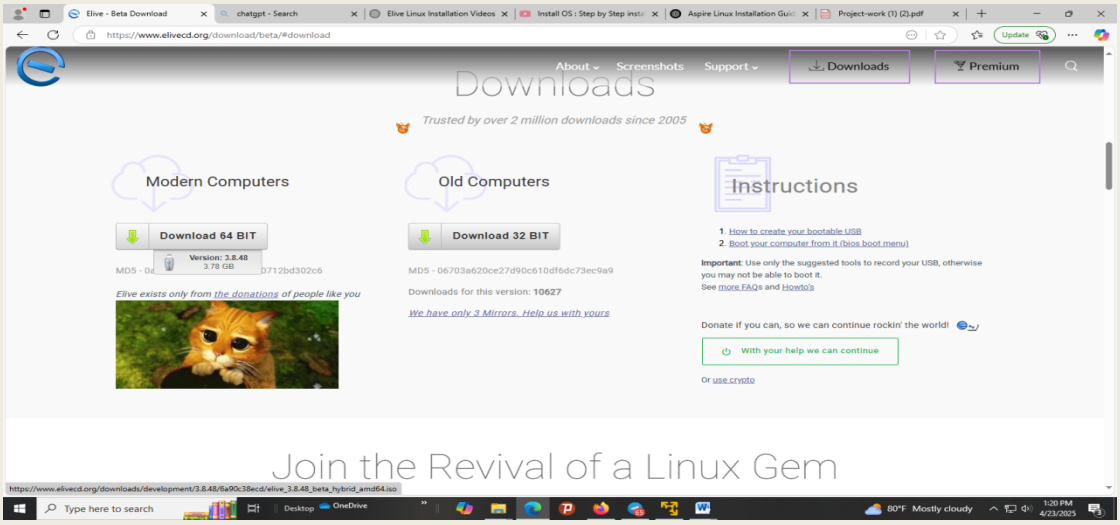
1.3. Steps of Elive os Installation in VMware Workstation Player

When I starte this part of the project, I isn't sure if it would be complicated or straightforward turns out it was most smooth with a few small issues along the way. Here's how I installed Elive OS in a virtual machine using VMware Workstation Player

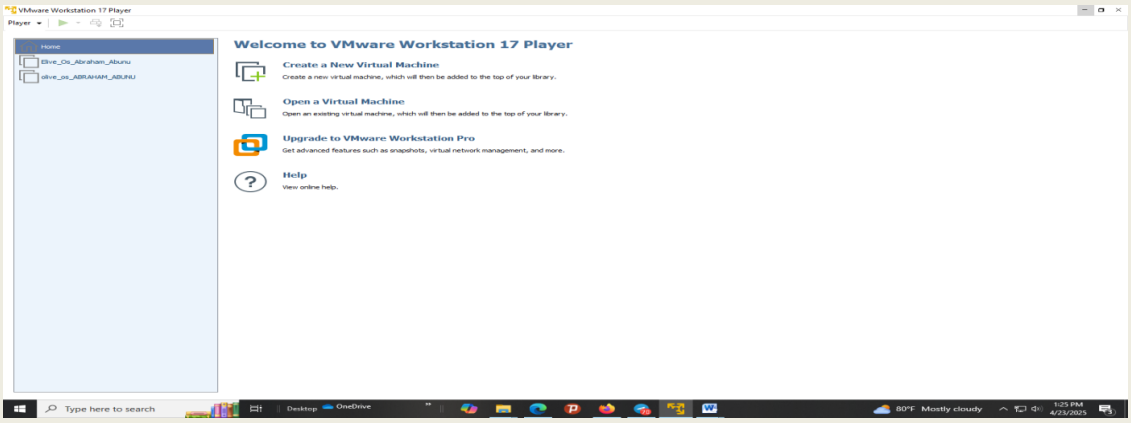
1.3.1 VMware workstation Set-up process for Elive os

The installation process is done using VMware workstation workstation free and open-source virtalization tool.

Step 1: I went to the official Elive OS webste and downloading the latest .iso file. The download was around 3.5 GB and took about 15-45 minutes with my internet connection.

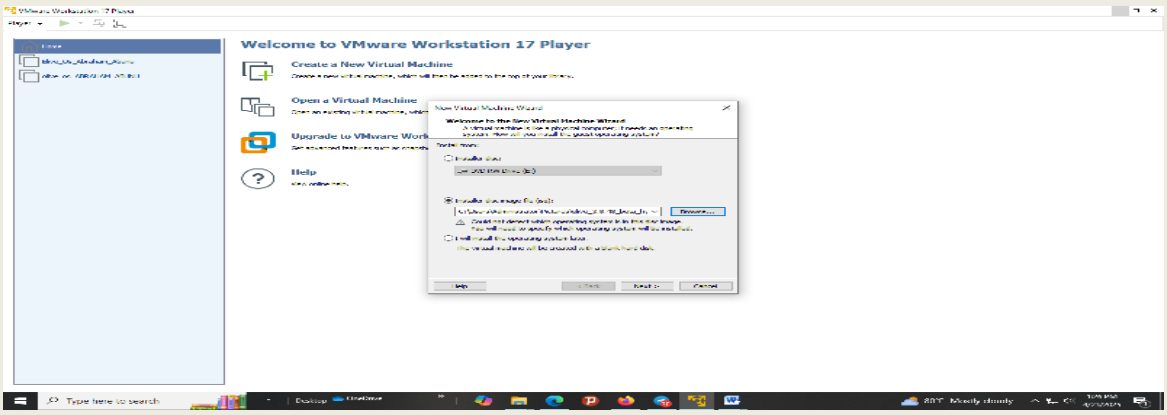


Step 2: After opened **VMware Workstation Player**, I clicked on **“Create a New Virtual Machine.”**



For the installer option, I selecte:

“Installer disc image file (iso)”, then browsed and selecte the Elive .iso I had downloaded.



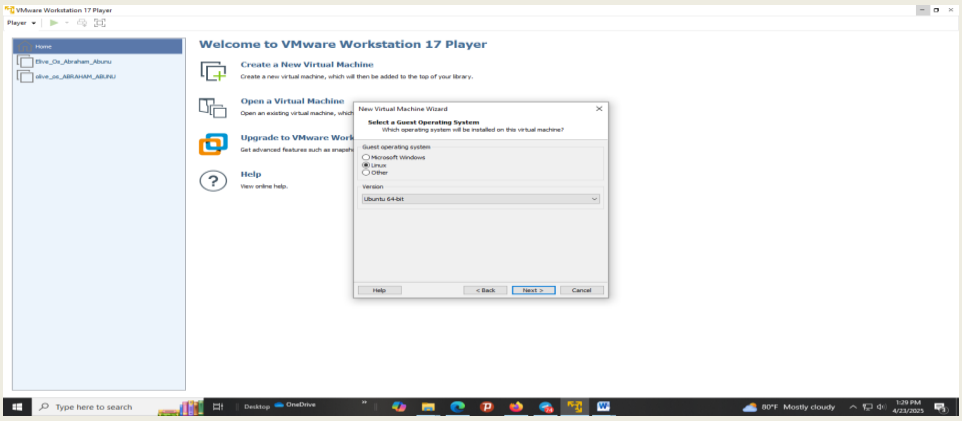
Click Next.

Note: If VMware says “Cannot detect operating system,” no worries—just continue manually selecting Linux. Because you can setetting up latter

Step 3: VMware didn’t automatically detect the OS, so I manually selected:

Guest OS: Linux

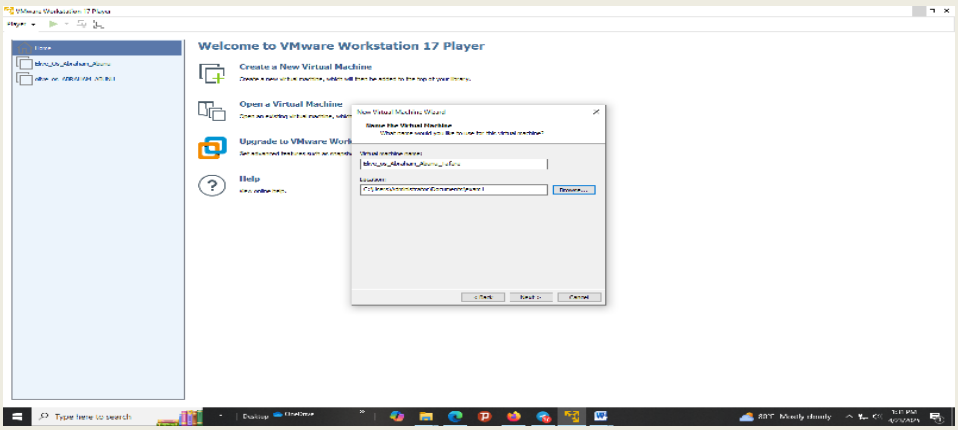
Version: Ubuntu 64-bit.



Click Next.

Step 4: Name Your VM & Set Location for your new virtual environment

I named the VM:
Elive_OS_Abraham
And saved it to a folder I created on my desktop for the project.

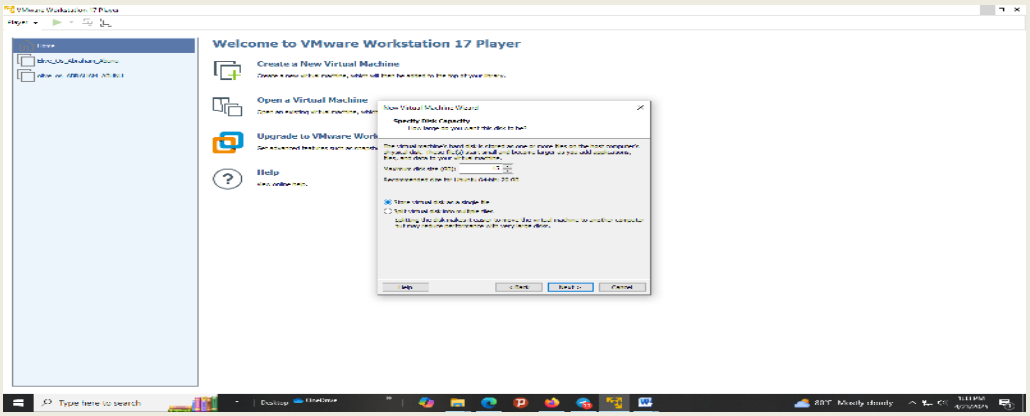


Click Next.

Step 5: I set the virtual disk to:

Disk size: 20 GB or more

Store as a **single file** for better performance and display capacity



Click Next.

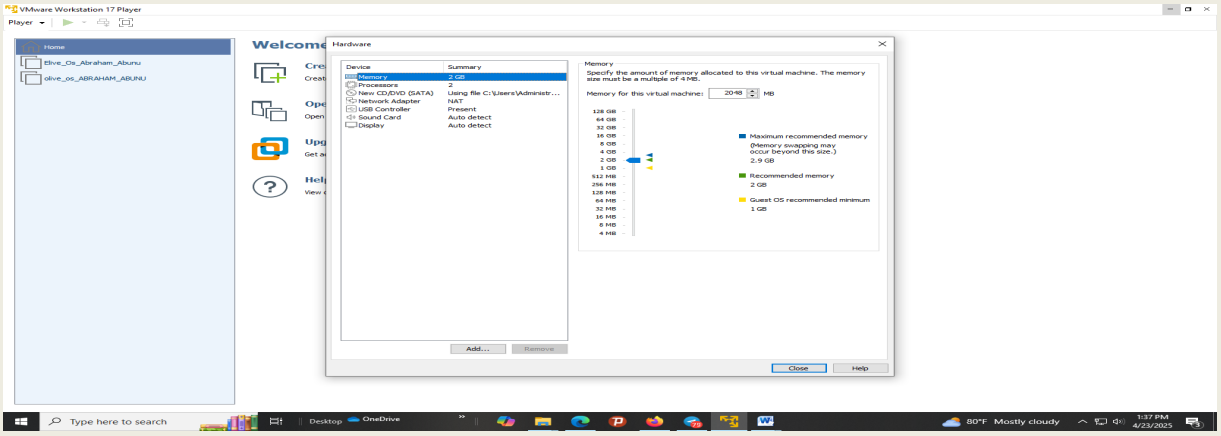
Step 6: Customize Hardware (Optional but Recommended)
Before finishing, I clicked on **"Customize Hardware"** and made a few changes:

Memory: 2 GB

Processors: 2 cores

Network Adapter: NAT (default) is fine for nework configuration

Display: Enable 3D acceleration if desired

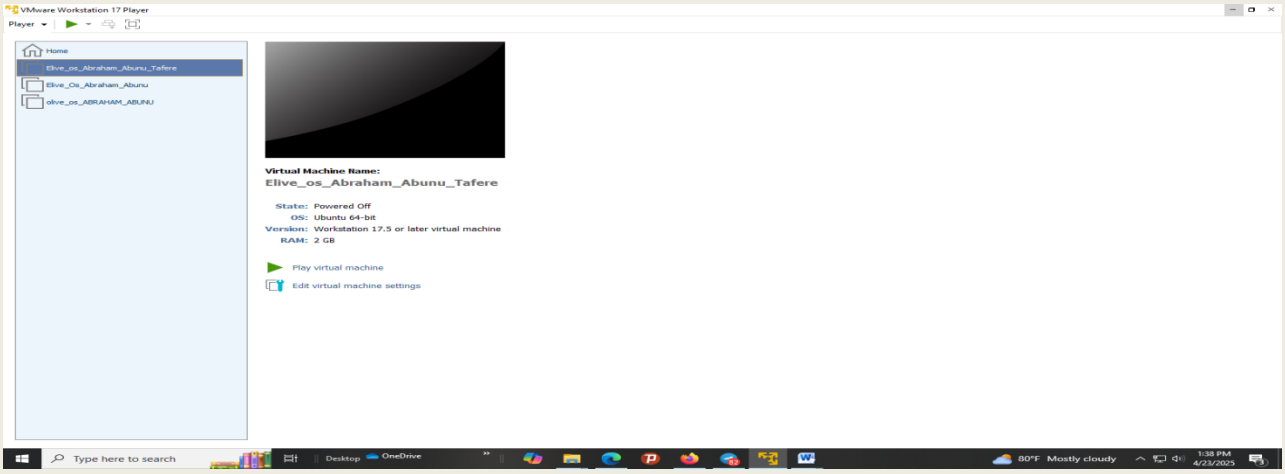


Click **Close**, then **Finish.**

Step 7: Start the VM

I clicked **“Play virtual machine”**, and after a few seconds, the Elive OS live session loaded.

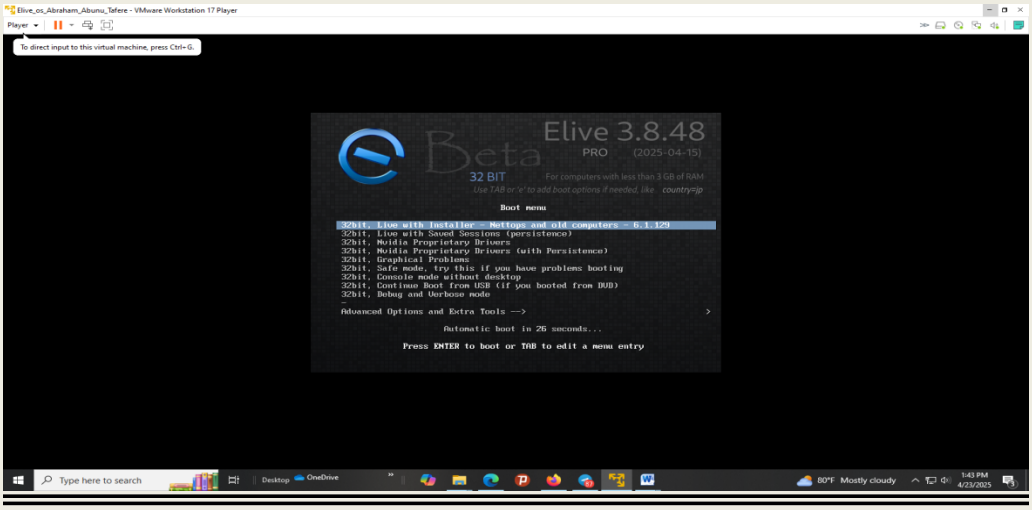
I used the live environment to explore a bit, then clicked on the **Elive Installer** icon on the desktop to begin the actual installation.



It should boot into Elive’s Live environment or installer.

1.3.2 Elive Linux installation process

Once you're on the Elive desktop installer:

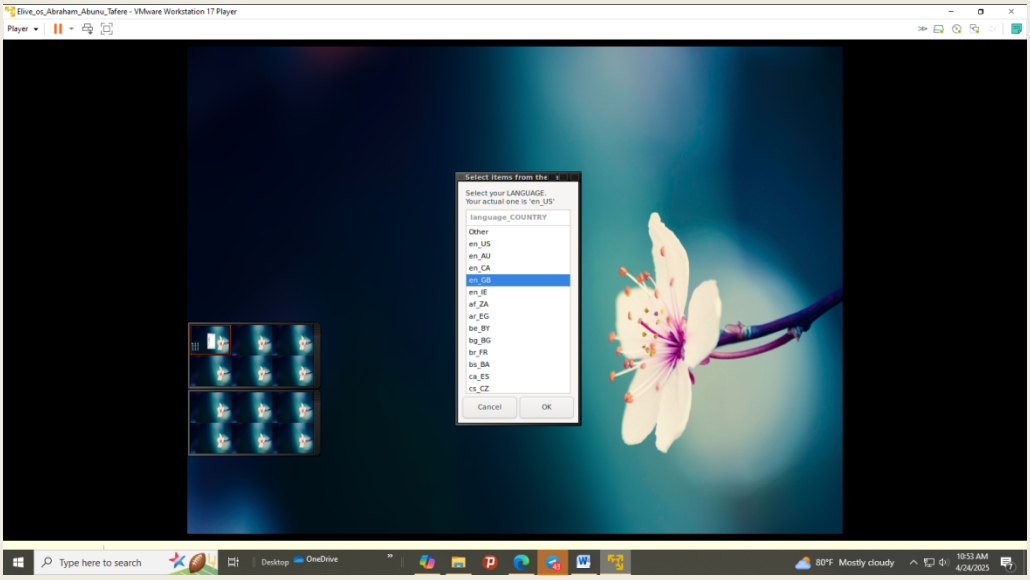


Step 1: Start the Installer

Click on the **Elive Installer** icon as shown in the above image

Step 2: Language and Keyboard

Choose your **language**



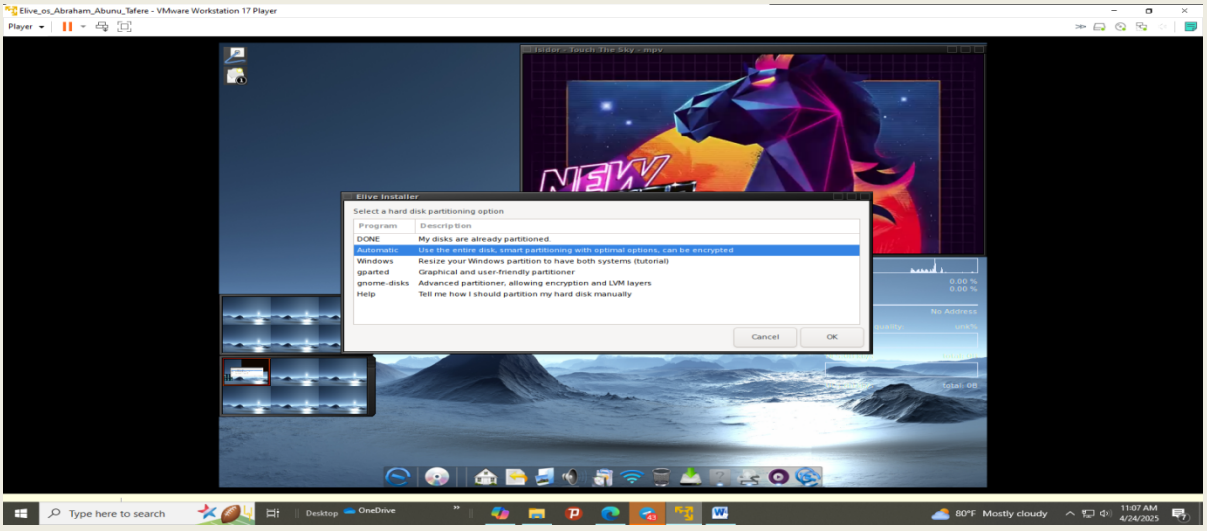
Select your **keyboard layout**

Click **Next**

Step 3: Partitioning

Choose **Automatic Partitioning** (recommended for most users)

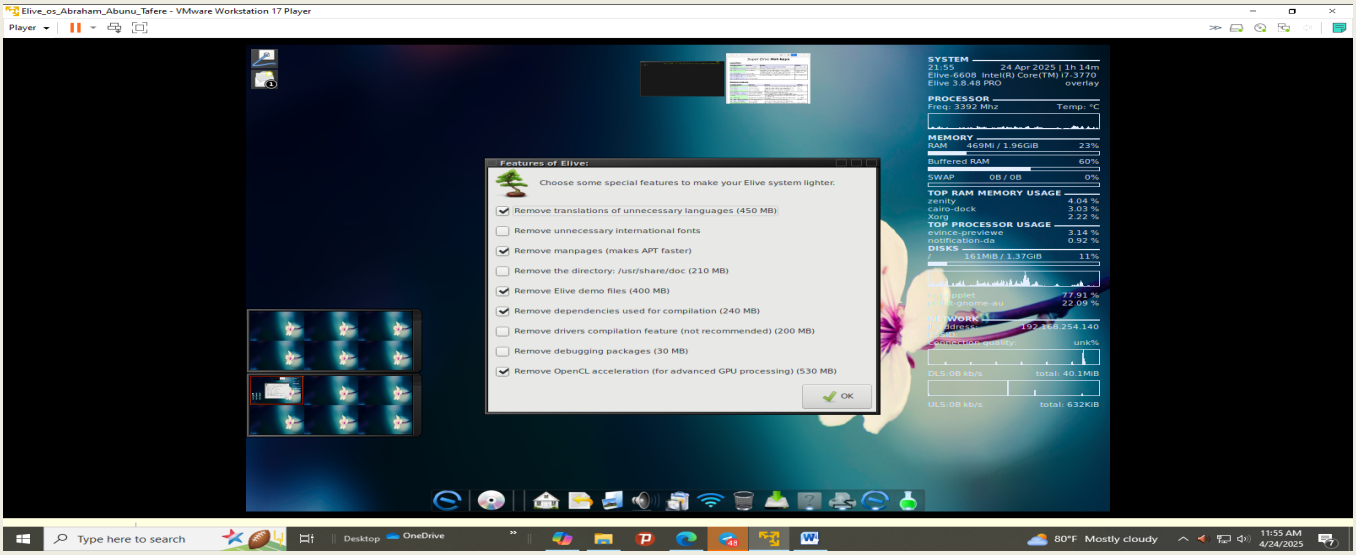
It will automatically set up root and swap partitions



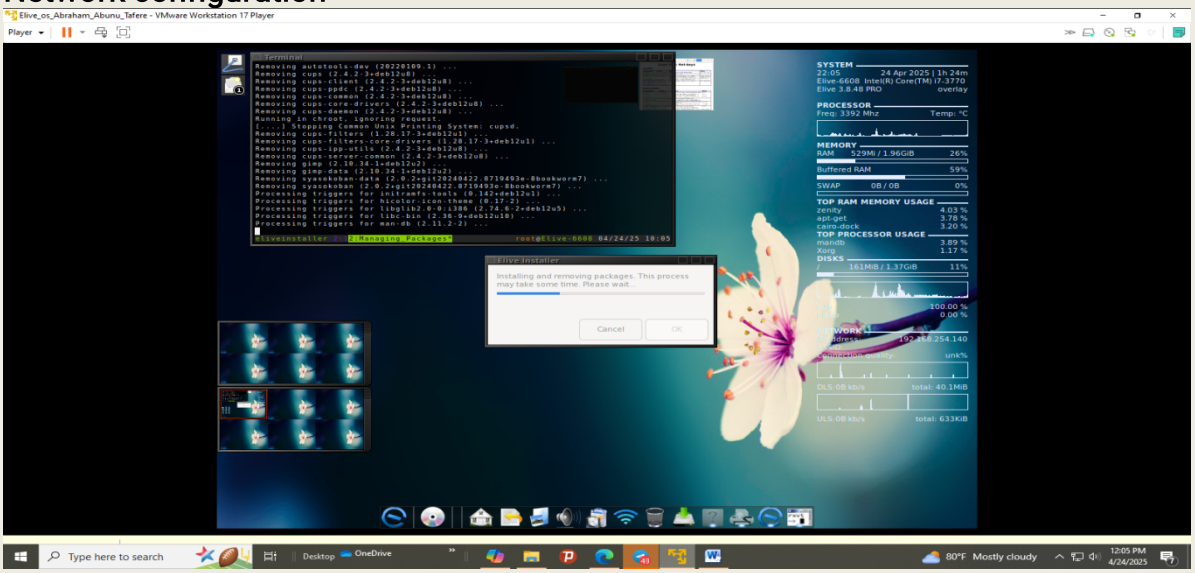
Alternatively If you're experienced you can choose **manual partitioning**

Confirm to continue and wait untile the disk is installed the OS

Step 4: configuire the installed Elive os
follow some steps to complete configuration

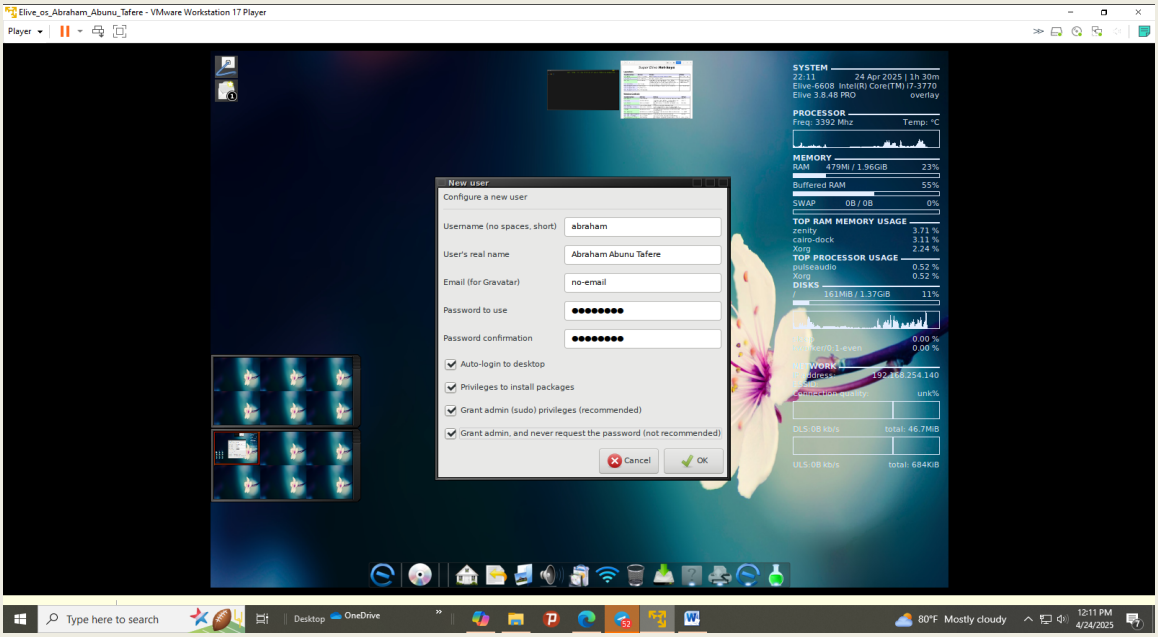


Network configuration



Step 5: User and System Settings

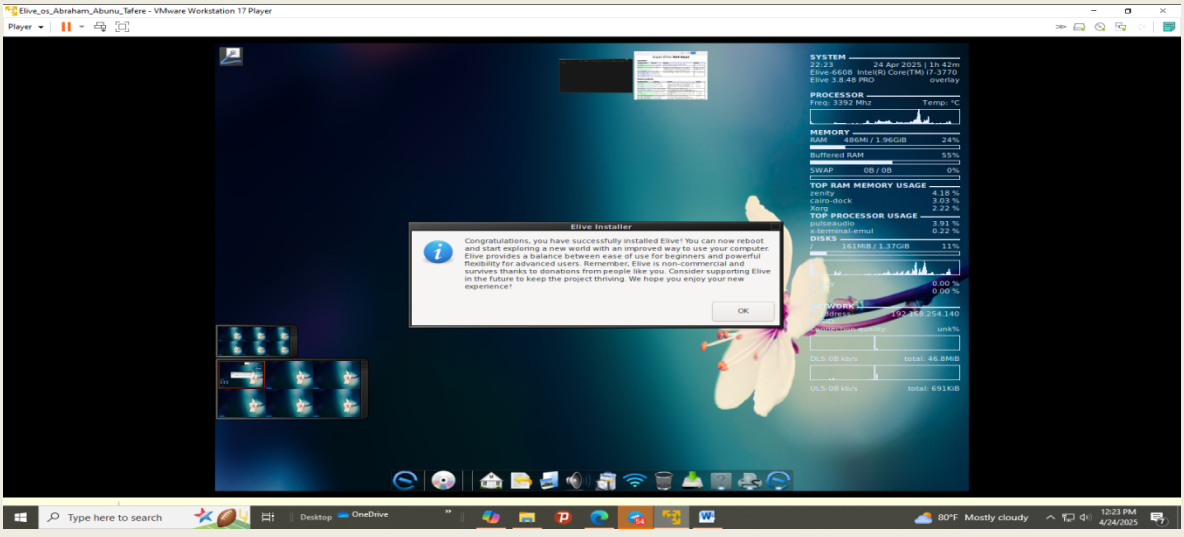
Set your **username**, **password**, and **hostname** (computer name)



Confirm the password and continue

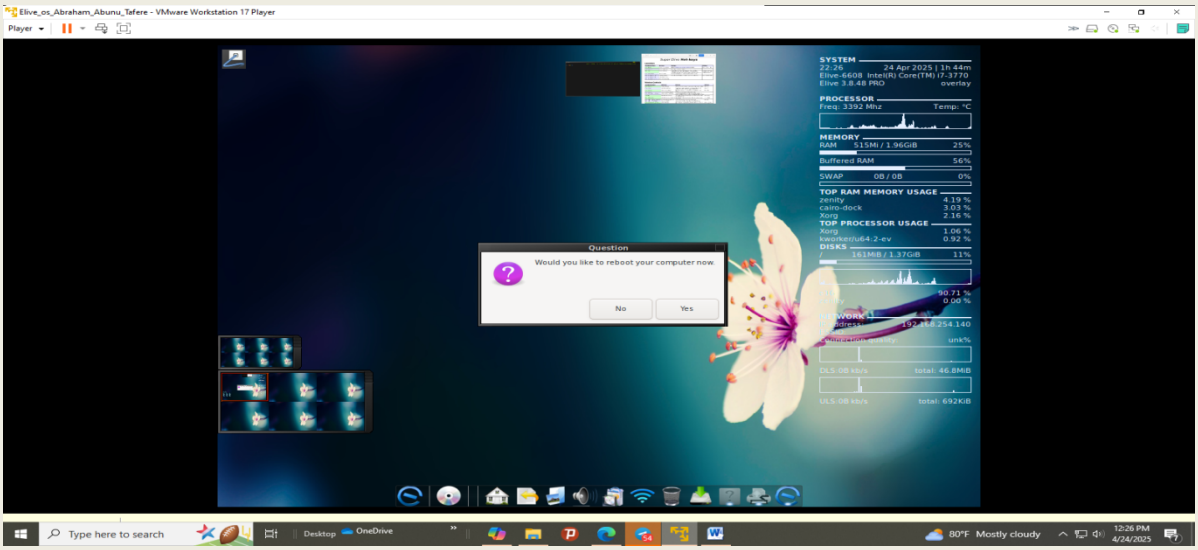
Step 6: complete Installation

Confirm all your choices



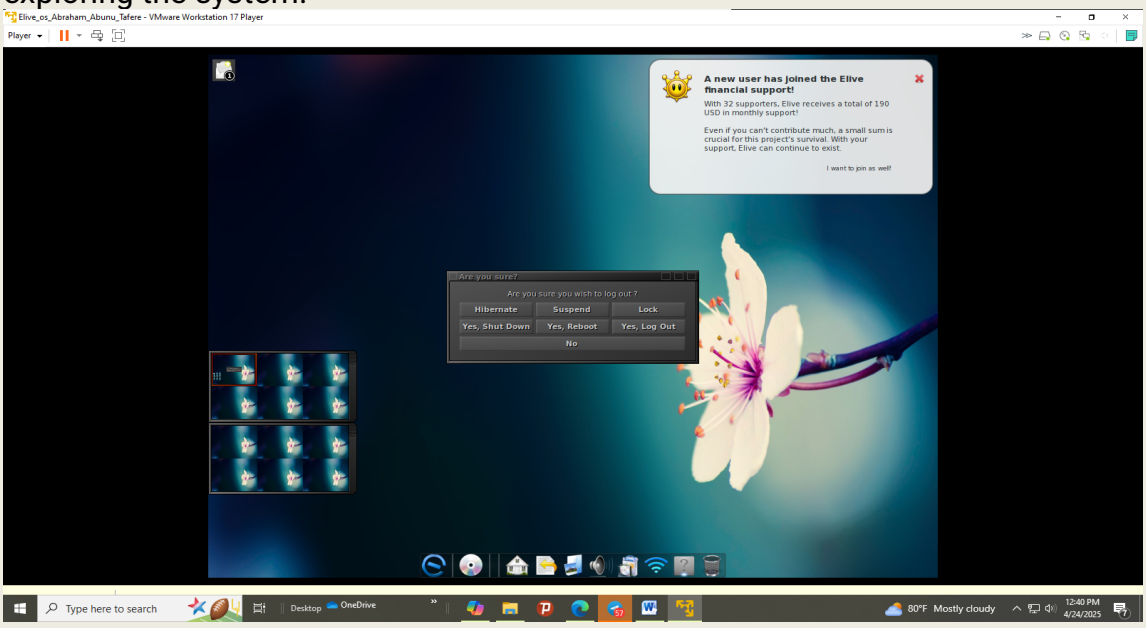
After Installation

After that, I was asked to reboot the system. Before restarting, I made sure to **disconnect the ISO file** from the virtual CD drive (in VMware settings), so it wouldn't boot into the installer again.

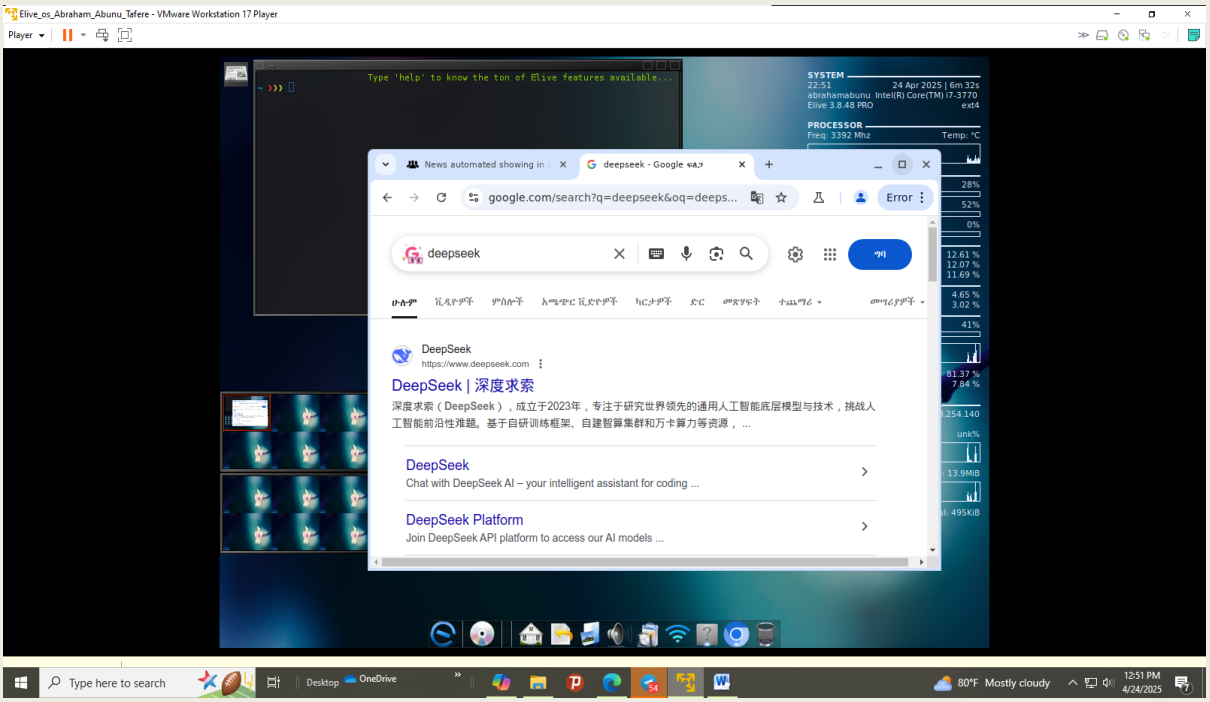


First Boot into Installed Elive

Once it restarted, Elive loaded properly! I logged in with my user credentials and began exploring the system.



Now we can you use the new operating system like these **example chrome.**



1.4 Problems and Solutions(Problems I Faced (And How I Solved Them))

Even though the installation process went fairly smoothly, I ran into a few issues along the way. Here's what happened and how I fixed it:

1. Confusing Partitioning Step

At first, *I tried manual partitioning, thinking I could customize the setup. But the UI wasn't clear, and I ended up getting confused about which partition was for root or swap. So I went back and chose automatic partitioning instead — much easier and worked without issues."*

Lesson learned: If you're not sure about partitions, stick with the **automatic option**. It saves a lot of hassle.

2. Slow Performance During Installation

"The installer lagged a bit the first time I tried it. I checked the VM settings and realized I had only assigned 1 GB of RAM. I increased it to 2 GB and enabled 3D acceleration — after that, everything felt smoother."

Tip: Always check your VM's RAM and hardware settings before starting installation.

3. Display Resolution Was Too Small

After the OS was installe, the screen was stuck at low resolution, and it don't adjust automatical. I looks into it and found that I needed to install VMware Tools or use the Display Settings in Elive to set it manually.

Fix: Installing VMware Tools or adjusting resolution manually from within the Elive settings helps fix this.

Summary

These small issues were part of the learning process, and solving them gave me a better understanding of how virtual machines work — especially how hardware resources like RAM, graphics, and storage affect performance.

1.5 Filesystem Support in Elive OS

What I Learned About Filesystems in Elive OS

Before this project, I didn't think much about filesystems — I just clicked "Next" during installations. But while working with Elive OS, I got curious about what each filesystem option actually does and how it affects performance.

1. ext4 The Default and Best Choice

I used **ext4** for my Elive OS installation, and it worked great. Its the default for most Linux systems, and now I understand why. It's fast, reliable, and supports journaling — which basically means it keeps track of changes in case something goes wrong

Why I liked it:

- Quick boot times
- No problems with file storage or corruption
- Recommended by most Linux guides

2. FAT32 – Good for Sharing with Windows

I tested a FAT32 partition just out of curiosity. It's super compatible — you can read/write to it from both Linux and Windows. But I noticed some big downsides: it can't store files larger than 4 GB, and it doesn't support Linux-style file permissions.

When it's useful: For USB drives or external storage that you want to use on multiple system

3. NTFS – Works, But Not Ideal

Elive could read NTFS partitions, but writing to them was limited. I had to install an extra package (ntfs-3g) to get full access. It worked, but it felt slower and didn't really match well with Linux tools.

Verdict: Good for dual-boot setups, but not recommended for a main Linux installation.

What I prefer:

Choosing **ext4** was the right move for this project. It gave me the best performance inside the virtual machine and felt stable. Now I understand why filesystems matter — they actually affect how fast your system feels and how safe your data is.

1.6 Disadvantages of Elive 6 Advantages and OS

What I Liked (Advantages)

1. Lightweight and Fast

Elive OS ran surprisingl well with just 2 GB of RAM in virtual machine. Even on limited resource it felt smooth and responsive This makes it a great choice for older computers or low-spec devices.

2. Unique and Beautiful Desktop

I had never used the **Enlightenment** desktop environment before, and it really stood out. It looks clean, has cool animations, and doesn't use much system power. It gave the system a "futuristic" feel, which I enjoyed.

3. Stable Because It's Debian-Based

Since it's based on Debian, I could use the **apt** package managers and install most Linux software easy. I had no majorer system crashe or bugs while using it.

4. Great for Learning

Elive doesn't hide everything behind GUIs. I had to do some things in the terminal, which at first was challenging, but it helped me understand Linux better — especially when setting up things manually.

As Disadvantages

1. Small Community and Fewer Tutorials

When I got stuck, it was harder to find help online. Compared to Ubuntu or Fedora, there aren't as many YouTube tutorials or forums specifically about Elive.

2. Not Everything "Just Works"

For example, getting full screen resolution took some tweaking. Also, things like Wi-Fi or sound drivers might not work perfectly out of the box on real hardware (I only tested in a VM though).

3. Not the Easiest for Beginners

If someone is completely new to Linux, the Enlightenment desktop might feel confusing at first. There aren't many "welcome" popups or walkthroughs like you get in Ubuntu or Linux Mint.

4. Some Apps Aren't Preinstalled

Tools like Firefox or Zoom weren't there by default. I had to install them manually using the terminal, which is fine for me, but may be annoying for others.

Therefore

Elive OS is a **great project for learning Linux**, especially if you want to go beyond the more mainstream distributions. It's fast, different, and educational — but it does require patience and a bit of Linux curiosity.

Conclusion:

What I Took Away from This Project

Working on this project gave me real, hands-on experience with Linux and virtualization. Before this, I had never installed an OS inside a virtual machine, and now I feel confident doing it — not just with Elive, but with any Linux distro.

Elive OS surprised me with how lightweight and visually different it was. I ran it on just 2 GB of RAM, and it still felt fast. The Enlightenment desktop was totally new to me, and it made the system feel

more dynamic than I expected.

I also ran into some issues along the way — like resolution problems and partitioning confusion — but solving those helped me understand how Linux works under the hood. This project wasn't just about installing an OS; it was about learning to fix things myself, which I think is a big part of becoming a better tech student or future engineer.

1.7 Future Outlook & Recommendations

For Elive Developers:

- **Improve First-Time User Guidance**
  A small setup wizard or "Getting Started" tour would make the system more beginner-friendly.
- **Better Virtual Machine Integration**
  Adding pre-installed VMware Tools or VirtualBox Additions would help with full-screen support and smoother performance in VMs.
- **Expand the Software Repository**
  Making it easier to install common tools like Firefox, Zoom, or LibreOffice would help users who don't want to rely on the terminal right away.

For Students Like Me:

- **Try Elive on Old Hardware**
  If you have an old laptop sitting around, Elive is a great way to bring it back to life.
- **Use Projects Like This to Practice Real Skills**
  I learned more from installing Elive than I did from just reading about Linux. The best way to understand systems is by breaking and fixing them.

This project helped me go beyond theory and actually build something — even if it was inside a virtual machine. I'm glad I chose Elive OS for this because it challenged me and taught me new things at the same time.

2. Virtualization in Modern Operating Systems

2.1 What I Learned About Virtualization

Before this project, I didn't fully understand what virtualization actually meant. I had heard the term, but I thought it was only used by IT experts in data centers.

After setting up Elive OS in a virtual machine, I realized that virtualization is just the idea of running one computer **inside** another — like opening a second operating system in a window, without needing two physical devices.

It's kind of like running a game in a simulator. You're not installing it directly on your real system — instead, it's happening in a safe, separate environment. That's what VMware or VirtualBox does.

2.2 Why It's Useful (In My Experience)

While working on this project, I saw how useful virtualization can be, especially for learning or testing:

- **I didn't have to format my laptop** to try a new OS — I could just use a virtual machine.
- **If I messed something up**, I could easily reset or delete the VM and start over.
- **No risk to my real system**, since everything happened inside a safe, isolated virtual space.
- It also saves time — I didn't need extra hardware or a USB installer.

For anyone learning Linux, this is one of the best ways to experiment without fear.

2.3 How Virtualization Actually Works

Once I started digging deeper, I learned that virtualization relies on something called a **hypervisor** — this is the tool (like VMware or VirtualBox) that lets your real computer share its resources (CPU, memory, disk) with virtual ones.

There are two types of hypervisors:

- **Type 1** runs directly on the hardware (like in servers)
- **Type 2**, which I used, runs on top of an existing OS (Windows in my case)

The hypervisor makes it feel like each virtual machine is a real computer, even though it's just using a portion of your actual machine's hardware.

One thing I found interesting was how each virtual machine has its own "fake hardware" — it gets virtual RAM, a virtual disk, and even a fake network card. Yet everything inside works just like a real computer. That really helped me understand how flexible and powerful virtualization is.

3. Implementing System Calls Using Shell Script

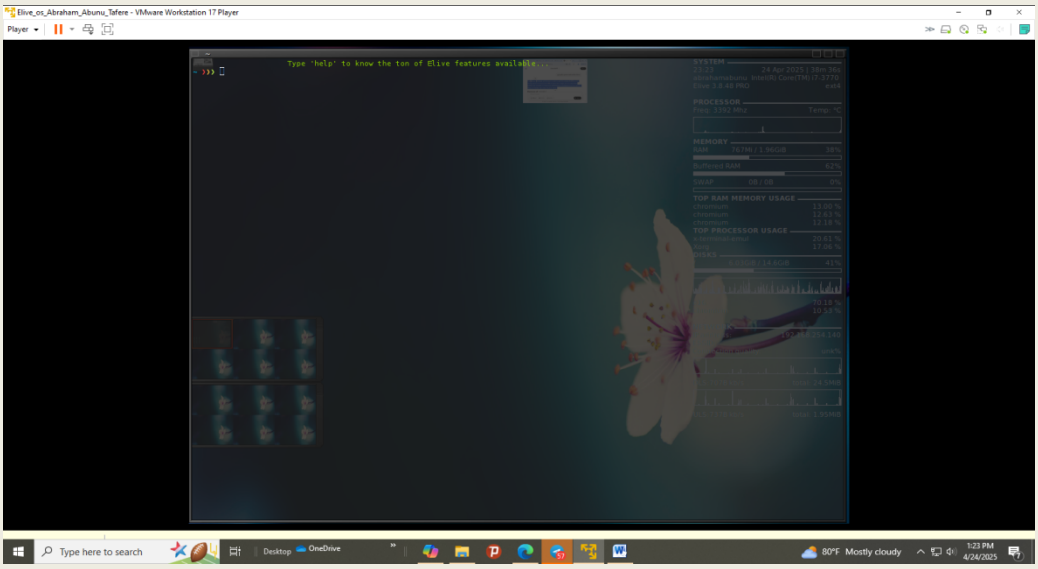
3.1 What is getpid()?

During the project, I wanted to try something simple that involved a real Linux system call. I picked `getpid()` because it felt like a good place to start — it's a basic call that returns the process ID (PID) of whatever program is running.

3.2 Practical implementing of getpid() in Elive OS

Step 1: Open the Terminal

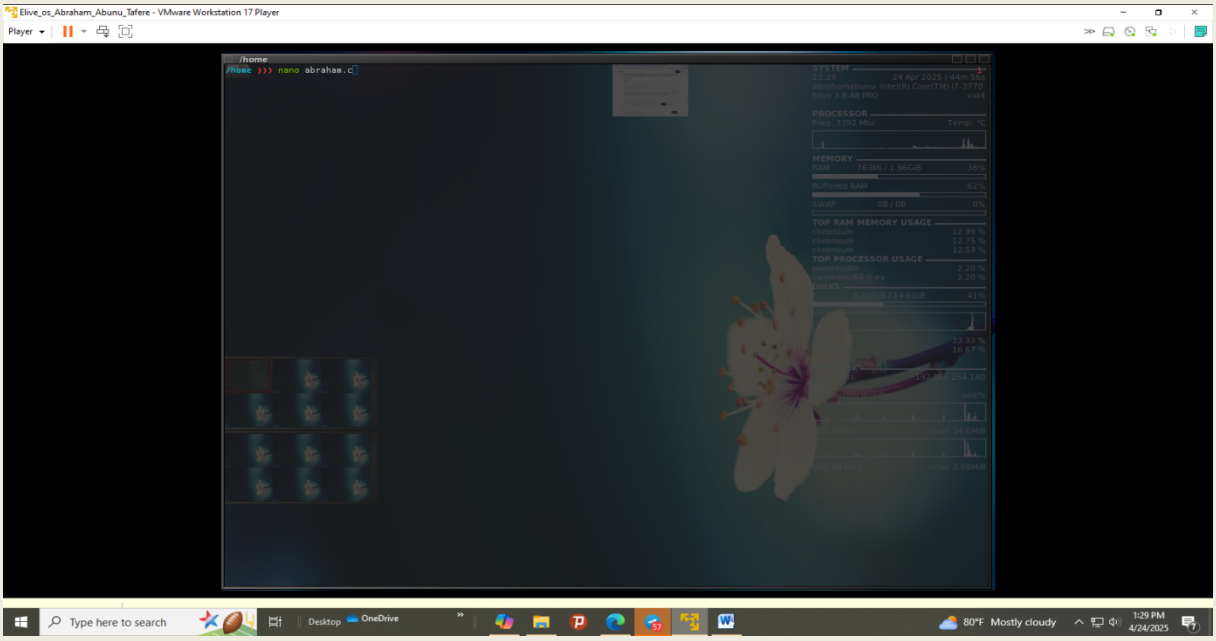
I used the terminal built into Elive OS. Since it's a Debian-based system, all the usual Linux commands worked.First open the terminal from your applications menu or by pressing



Step 2: Write the Code

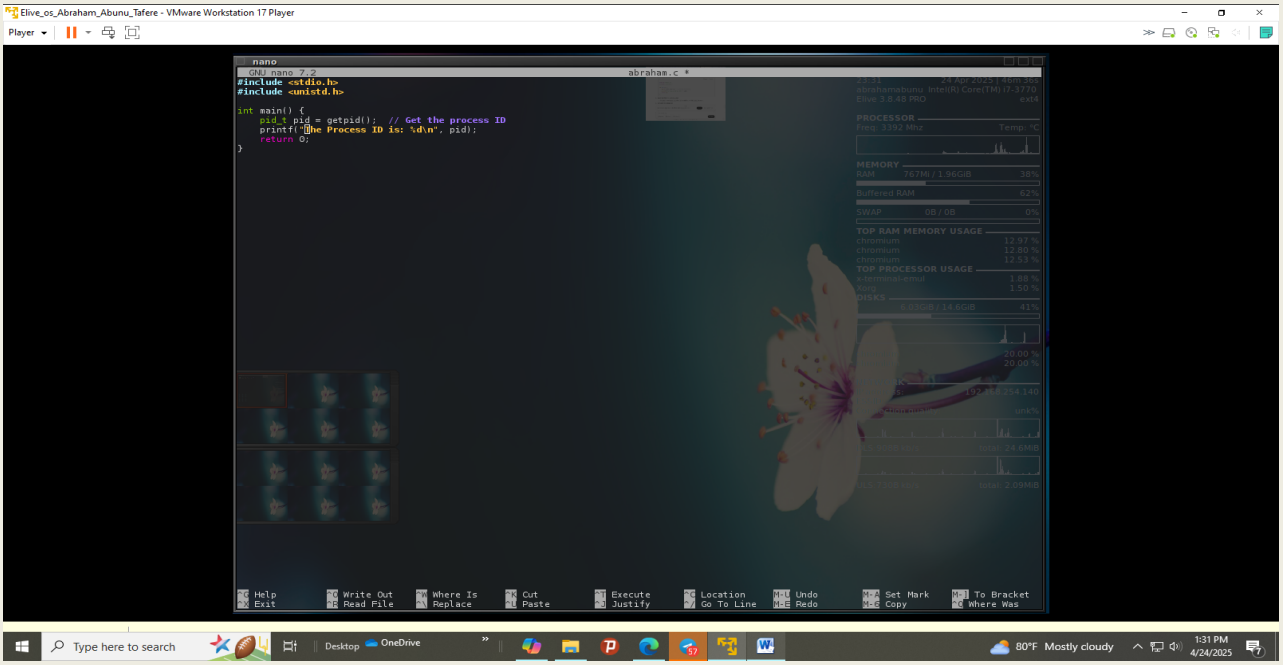
Create a new C file using the text editor nano (I used nano to create a simple C file):

nano abraham.c



Inside the file, I wrote this:

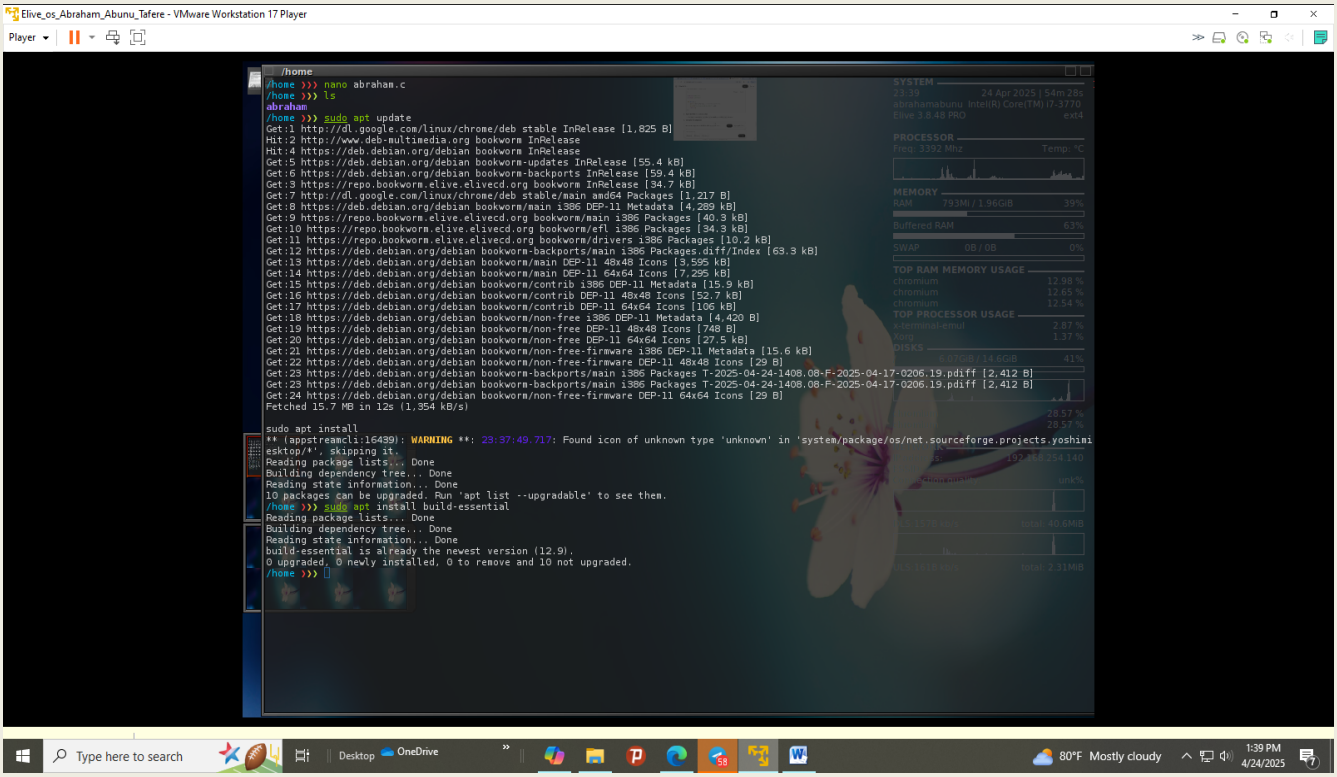
```
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t my_pid = getpid(); // my process id
    printf("My Process ID is: %d\n", my_pid);
    return 0;
}
```



Step 3: Compile the Program

To compile the code, you need gcc, which is available by default or can be installed using:

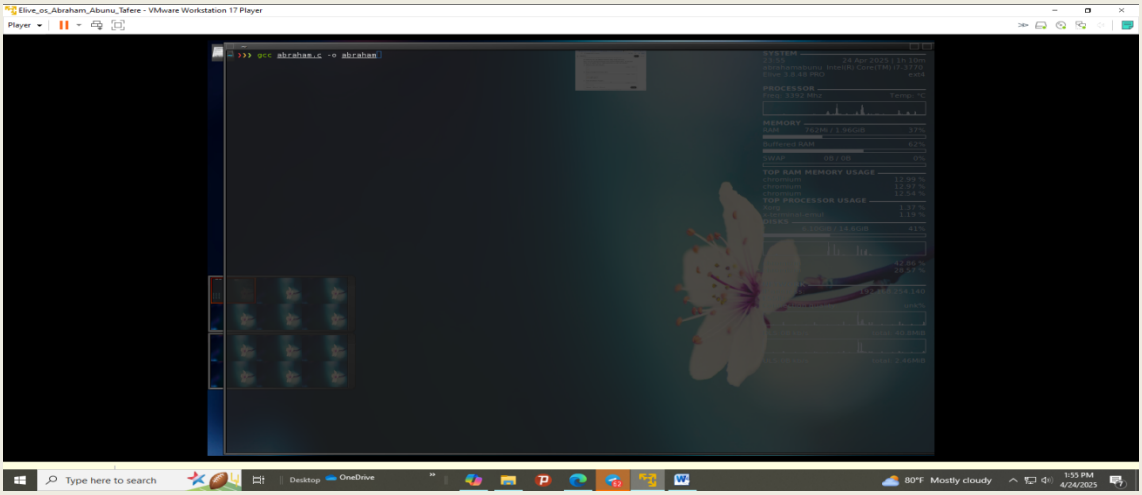
```
sudo apt update
sudo apt install build-essential
```



Then **compile** the program:

After saving the file and gcc installed, I compiled it like this:

gcc abraham.c -o Abraham

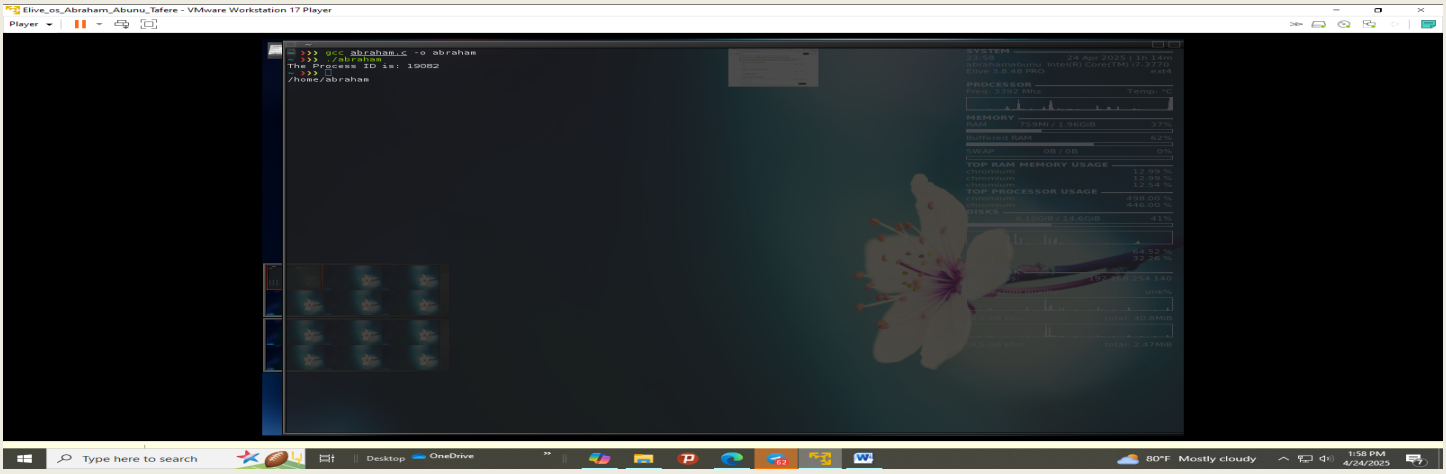


Step 4: Run the Program

Now you can run your compiled program:

./abraham

You should see something like:



Why Use getpid()?

Here's a practical use case: Imagine you are building a program that creates logs for monitoring. Each log entry can be tagged with the process's PID so that if something goes wrong, you know exactly which process was involved.

conclusion

I thought it was something only advanced programmers use. But getpid() showed me that even basic tasks — like asking the OS for your process ID — are system calls under the hood.

When I first tried using getpid() in Elive OS, I was curious how Linux tracks processes under the hood. It turns out that even tiny scripts and apps get their own PID, and this system is what allows multitasking to work smoothly. Once I ran the code and saw my PID printed on the screen, it felt like opening a window into how the OS "sees" my program.

Thank You!