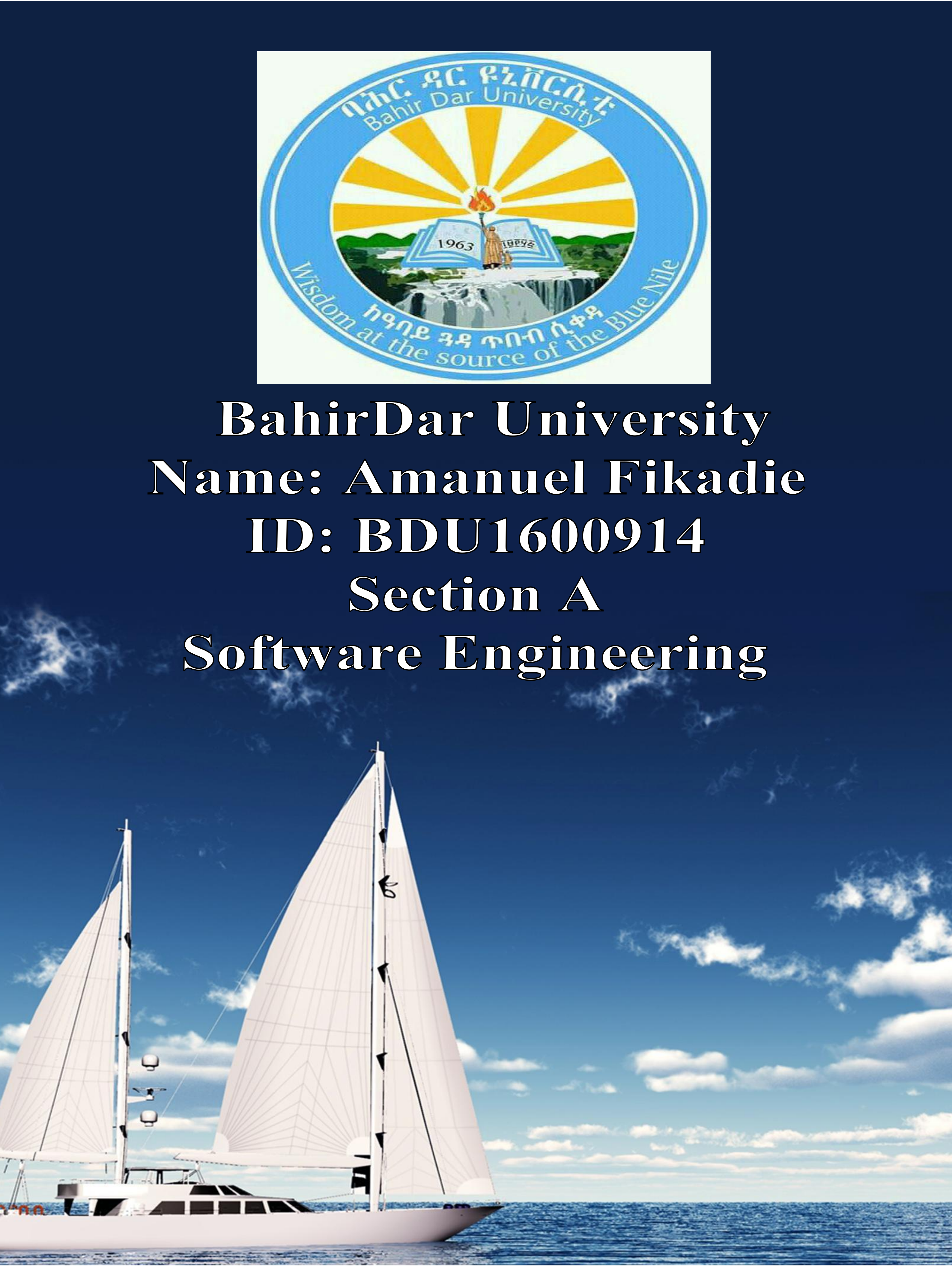


BahirDar University

Name: Amanuel Fikadie
ID: BDU1600914
Section A

Software Engineering

Introduction (Background and Motivation)

Operating systems are the core layer of a computer system that lies between the main hardware and application (the application itself) and provides its basic functions, including process scheduling, memory management, file systems, device control, and security. For future system programmers and engineers, computer OS-level experience in the installation and configuration, as well as system call behavior, are necessary to grasp a large part of what goes on in today's computer systems.

PureOS has been chosen as the operating system of focus for this project. PureOS is a GNU/Linux distribution produced by Purism, built completely using free/libre and open source software (FOSS). It has been officially endorsed by the Free Software Foundation (FSF). As such it is one of the few operating systems in existence that does not contain either proprietary software or binary blobs, thus emulating the ideals of the pure OS – transparency and auditability towards the user as well as control towards the administrator. Its design is oriented toward the general aim of users freedom, security and privacy. Because unlike regular desktop Linux distributions (e. g. Ubuntu or Fedora) which use proprietary firmware and drivers, PureOS strictly follows the free software definition: Offers complete source code transparency. Ensures users' digital rights and freedoms.

- It provides a minimally clean system without telemetry or tracking or any non free components. In this project we install PureOS in Oracle VM VirtualBox using the terminal (CLI) at every possible step. By doing this we are freeing up on graphical installation instruments, and practicing in pure environments (like on headless servers or embedded Linux machines) -- this provides excellent learning opportunities, most notably: Partition management Filesystem formatting User account setup via shell Package installation via APT Service management via systemd Because this use of virtualization in this application is a safe &

isolated way to simulate real world system installations without modifiying the host system itself. Also in this case we chose PureOS because we want to start with something quite secure & minimalist * but * with high freedom levels that is ideal for both developers and users who want complete control over their computing. With this project students learn not only about technical aspects of OS installation and configuration, but also about open source philosophy and the effects of system level decisions (for example filesystem, bootloaders, user permissions) on operating system behavior and performance. Key Concepts Introduced in This Work: Virtualization using Oracle VM VirtualBox. Manual installation of a Linux OS from ISO. Exploring filesystems like ext4 in a pure environment. Just understanding what makes an OS " pure " from the free software / system programming perspective.

Objectives

Purpose The main goal of this project is to practice installing and configuring a pure Linux-based operating system — PureOS — from a command-line perspective in a virtualized environment, thereby allowing students to learn the fundamentals of the operating system, learn how system-level configurations are managed, and learn to interact with the OS at low level without relying on graphical tools.

☑ Understand and Apply the Pure OS Philosophy
Read more about how PureOS strictly adheres to the Free Software Foundation (FSF) requirements & ships with free/libre open source software.

Learn why not using proprietary drivers and firmware increases transparency, user control and auditability of your system.

Get a sense of how PureOS is taking privacy and security aspects into account by default (default search engine via DuckDuckGo, HTTPS Everywhere bundled, etc.).

✅ Practice Virtualized OS Deployment Using a Terminal Approach
Install PureOS with Oracle VM VirtualBox. Most things (creation of VM, ISO mounting, configuration) are done with command line tools [VBoxManage and apt].

Get to know about bare metal configurations — since pureOS install and configuration emphasize the use of terminals instead of GUI.

Create a virtual machine which will run in a real server environment where headless Linux systems are used very often.

✅ Learn the Full Operating System Installation Process
Download PureOS ISO images and check the checksums and signatures - very good habit for safe OS deployments.

Manual installation from the bootloader to desktop/login screen. Understanding of every step of the boot process till creating user account.

Parition disk, choose a filesystem (like ext4), specify hostname and create admin users All the stuff a sysadmin should be doing.

✅ Explore Filesystem Management in Pure Environments
Learn and use ext4 (the default filesystem in PureOS), it 's stable, has journaling and is fast.

Learn about filesystem creation and mounting (mkfs, mount, fstab) as well as the reasons PureOS doesn't support proprietary/less-supported filesystems like NTFS/APFS.

✅ Build Problem-Solving and Debugging Skills
Encounter and resolve real-world system issues such as:

No network connection in the VM.

Display driver problems due to free software restrictions.

Grub bootloader installation issues.

Develop the ability to diagnose and fix problems without help from graphical user interface by using only logs, shell commands and configuration files.
☑ Define PureOS' role as a platform in the Broader OS Ecosystem
Read about the ethical and practical implications of using a privacy-first OS like PureOS in a world where proprietary systems rule.

To compare PureOS with other mainstream Linux distributions (e. g. Ubuntu or Fedora).

Licensing

Security approach

Target audience (privacy-conscious users, developers)

☑ Prepare for System Programming and Kernel Interaction
Orientation / prerequisite for System Calls / Shell Scripting and Low Level Programming in the Following Part of the Class.

Start studying user applications interactions with kernel interfaces - especially in environments where raw OS behavior is available.

Summary of Learning Goals

By undertaking this project, several of the primary learning objectives were met, and each was aiding in the advanced understanding of concepts of operating systems and system-level interaction via PureOS.

To begin with, the project built a good understanding of the PureOS philosophy that is based on user privacy, digital freedom, and adherence to the Free Software Foundation (FSF) principles. Students acquired knowledge of why it is important to use an operating system free from proprietary software and maintains sovereignty of the users.

Secondly, use of virtualization allowed deployment of an operating system within a secure virtual machine, using terminal-based software. It replicated real-time situations in which headless or server-based setups are typically run entirely without any graphical interfaces.

Third, the project involved performing a terminal-based installation of PureOS. By avoiding graphical user interfaces (GUIs), students solidified their ability to use text-based installation tools and command-line utilities — skills that are core in system administration and embedded systems design.

In addition, students engaged in filesystem discovery with a focus on Linux's ext4 filesystem. They learned to format and configure storage partitions, mount filesystems, and understand the role of journaling in data consistency and system reliability.

Problem-solving was a critical component of the project, in which students encountered and resolved common installation issues, such as network misconfiguration, display limitations, and bootloader problems. The issues were addressed using terminal commands and system logs, improving diagnostic and troubleshooting skills.

The project also allowed the building of critical system administration skills, including creating user accounts, managing permissions, installing packages, and configuring the bootloader. These are fundamental tasks that are required in both consumer computing and enterprise IT environments.

Finally, the course provided a firm foundation in system-level preparation. By working directly with the operating system, students are now better equipped to work with system calls, write shell scripts, and understand how user programs interact with the kernel — all of which are essential to subsequent coursework in system programming and operating systems development.

Requirements

Hardware Requirements

Processor: 64-bit CPU that supports virtualization (Intel VT-x or AMD-V).
Memory (RAM): Minimum 4 GB (8 GB recommended).
Storage: Minimum 20 GB of available disk space.
Display: Standard keyboard and monitor (no special graphics requirements).
BIOS/UEFI: Virtualization technology is enabled.

Software Requirements

Host Operating System: Linux, Windows 10/11, or macOS.
Virtualization Software: Oracle VM VirtualBox (version 7.0 or higher).
PureOS ISO: Latest stable release from the official website.
Command-Line Tools: VBoxManage, sha256sum, gpg (for VM management and ISO verification).
Optional: Internet connection for package installation and software update.

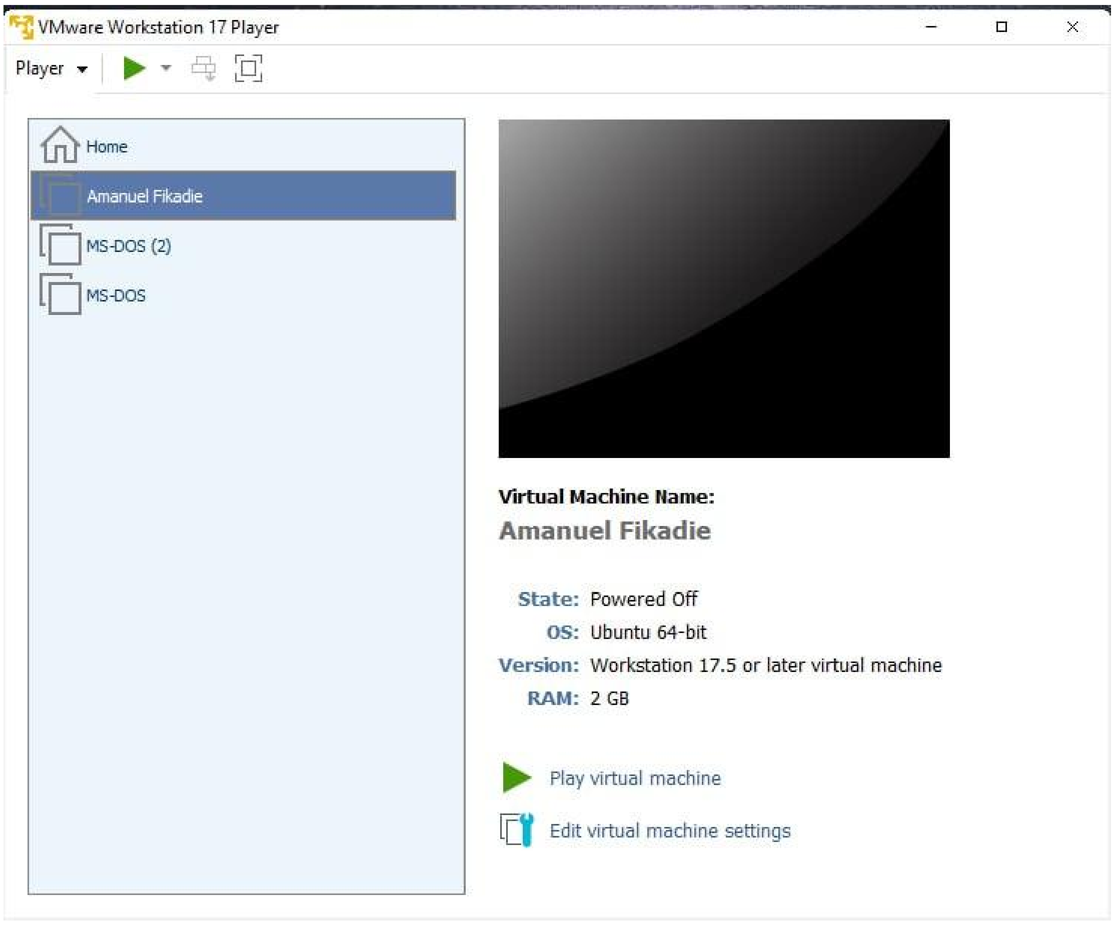
PureOS installation steps in VMware Workstation

Step 1: Open VMware Workstation
Launch VMware Workstation on your machine.
Step 2: Create a New Virtual Machine
- Select "Create a New Virtual Machine"
- Select "Typical (recommended)"
- Click "Next"

Step 3: Select the PureOS ISO File
- Select "Installer disc image file (ISO)"
- Click "Browse" and select the PureOS ISO file you downloaded
- Click "Next"

Step 4: Select Guest Operating System
- Select "Linux"
- In the version dropdown, select "Debian 10.x" or "Other Linux 5.x (64-bit)"
- Click "Next"

Step 5: Name the Virtual Machine
- Give a name to your virtual machine (you may use your full name, e.g., Amanuel Fikadie)
- Give a location to save the virtual machine files
- Click "Next"



Virtual Machine Name:
Amanuel Fikadie

State: Powered Off
OS: Ubuntu 64-bit
Version: Workstation 17.5 or later virtual machine
RAM: 2 GB

Step 6: Set Disk Size
- Assign a disk size of at least 20 GB
- Choose "Store virtual disk as a single file"
- Click "Next"

Step 7: Customize Hardware (Optional)
- Click "Customize Hardware"
- You can increase the memory (RAM) to 2 GB or 4 GB
- You can assign more processor cores if you have any
- Click "Close"

- Then click "Finish"

Step 8: Start the Virtual Machine

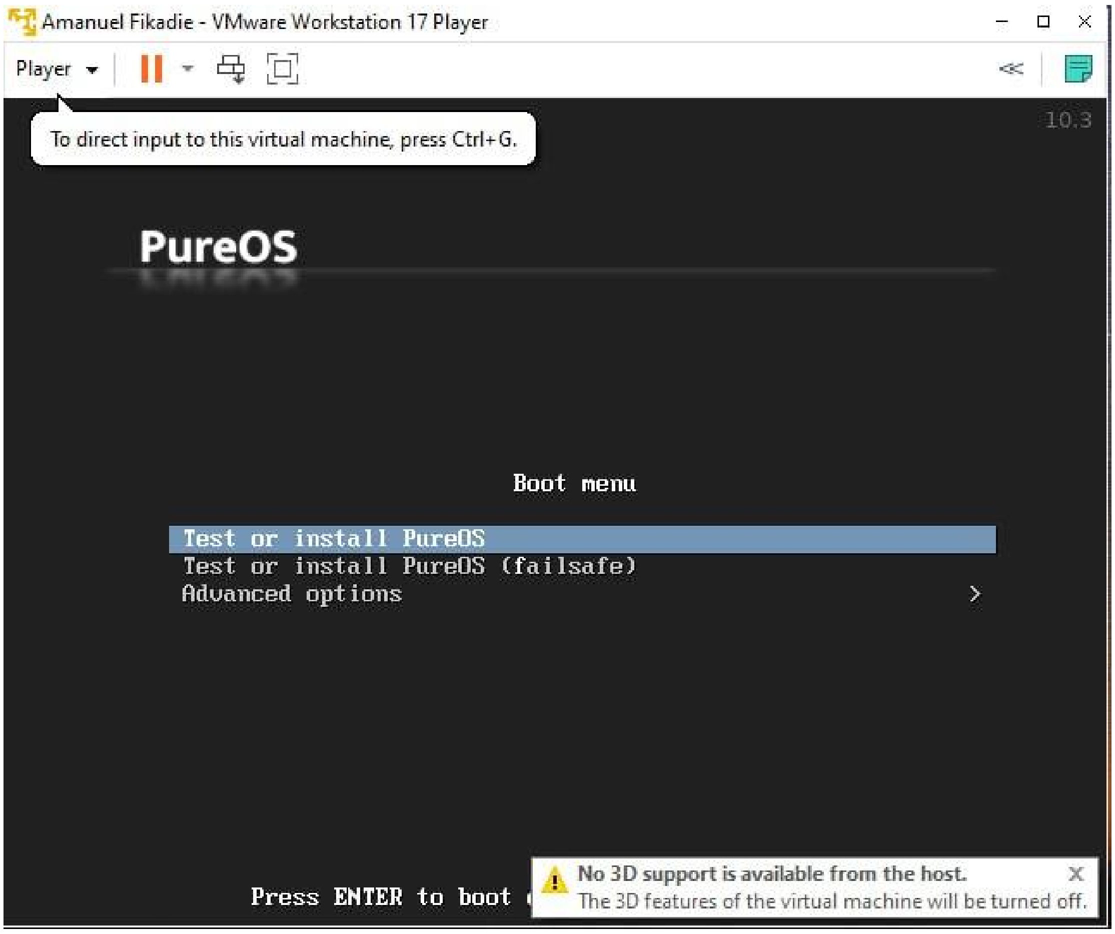
- Click "Power on this virtual machine"

Step 9: Boot into PureOS Live Mode

- When the boot menu appears, select "Live Boot PureOS"
- Press Enter

Step 10: Start Installation

- On the desktop, click "Install PureOS"
- Follow the installation steps:
  - Choose language
  - Choose region and timezone
  - Select keyboard layout
  - Use the entire disk for installation



To direct input to this virtual machine, press Ctrl+G.

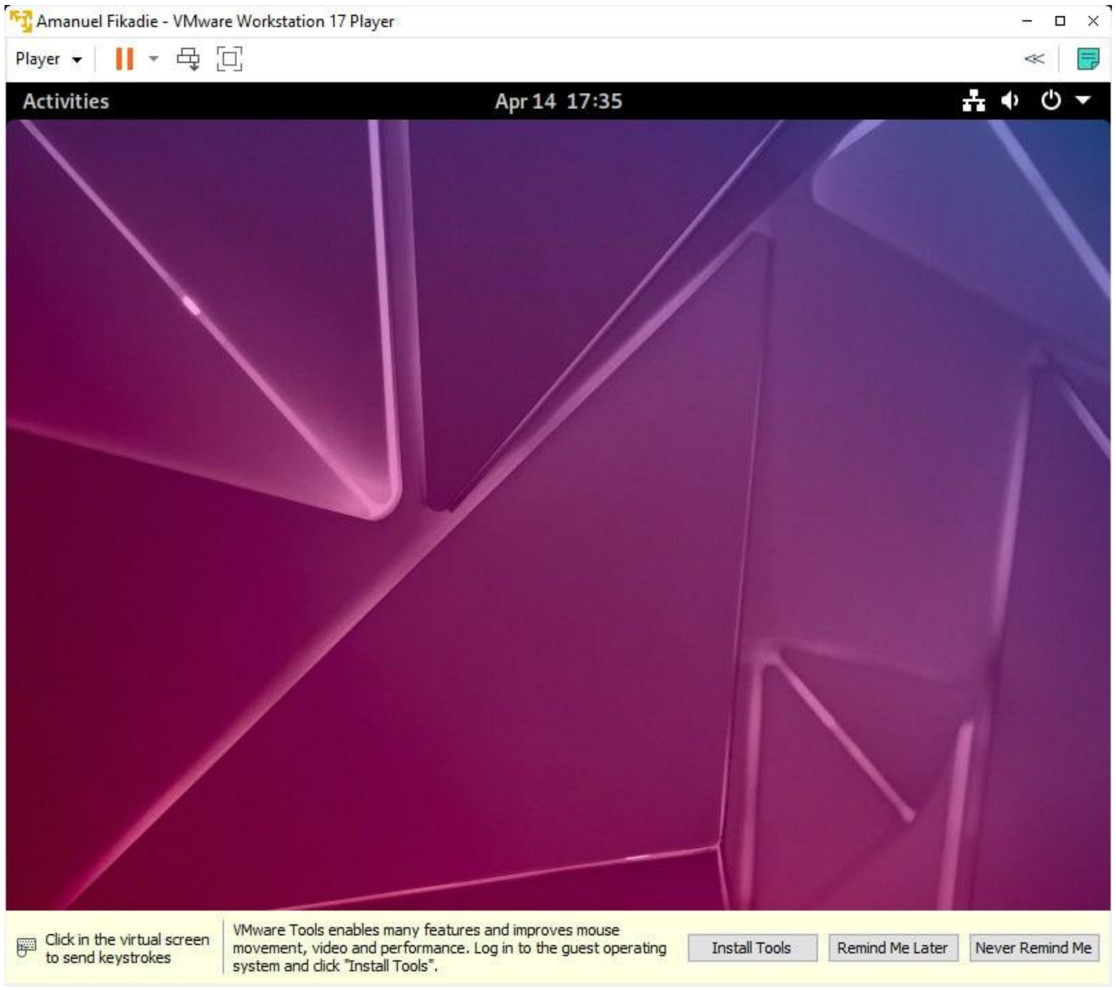
Boot menu

Test or install PureOS
Test or install PureOS (failsafe)

Press ENTER to boot

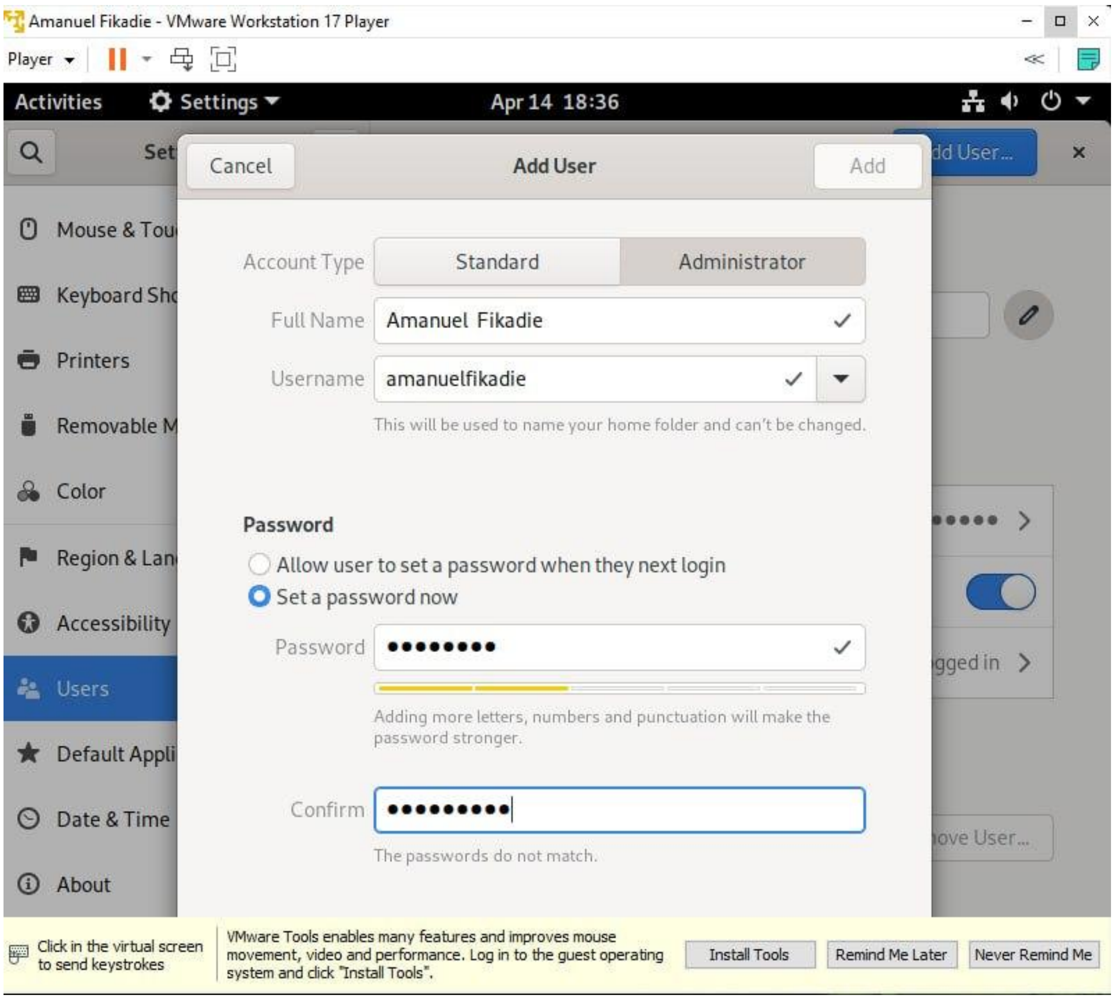
No 3D support is available from the host.
The 3D features of the virtual machine will be turned off.

and this is the interface of this operating system which is **pure operating system**


Amanuel Fikadie - VMware Workstation 17 Player
Player
Activities
Apr 14 17:35
Click in the virtual screen to send keystrokes
Install Tools
Remind Me Later
Never Remind Me

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

Step 11: Create User Account
- Fill in your full name
- Fill in a username (used for login)
- Create a password


Amanuel Fikadie - VMware Workstation 17 Player

Add User

Account Type: Standard [✓ Administrator]

Full Name: Amanuel Fikadie ✓

Username: amanuelfikadie ✓

This will be used to name your home folder and can't be changed.

Password

Allow user to set a password when they next login
[✓ Set a password now]

Password: •••••••• ✓

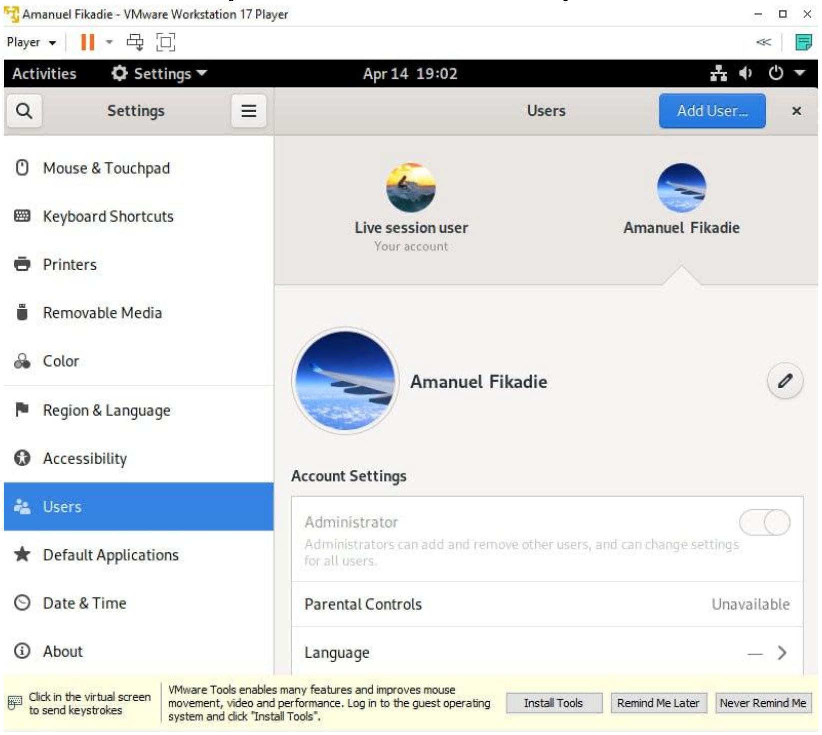
Adding more letters, numbers and punctuation will make the password stronger.

Confirm: •••••••••

The passwords do not match.

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

So this is my account with my full name and password


Amanuel Fikadie - VMware Workstation 17 Player

Users

Live session user — Your account

Amanuel Fikadie

Account Settings

Administrator — Administrators can add and remove other users, and can change settings for all users.

Parental Controls: Unavailable

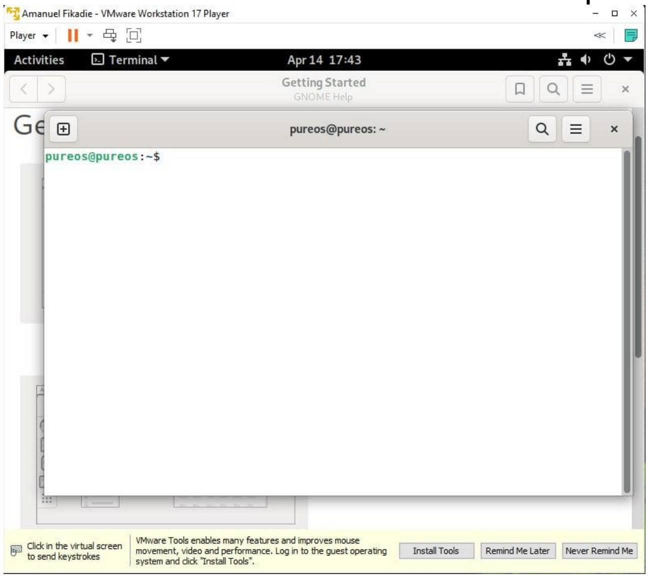
Language: —

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

Step 12: Complete Installation

- Once the installation is finished, click on "Restart Now"
- Move to VM settings and unmount the ISO file from the CD/DVD drive
- The system will reboot into the installed PureOS

This is the defualt terminal of this operating system

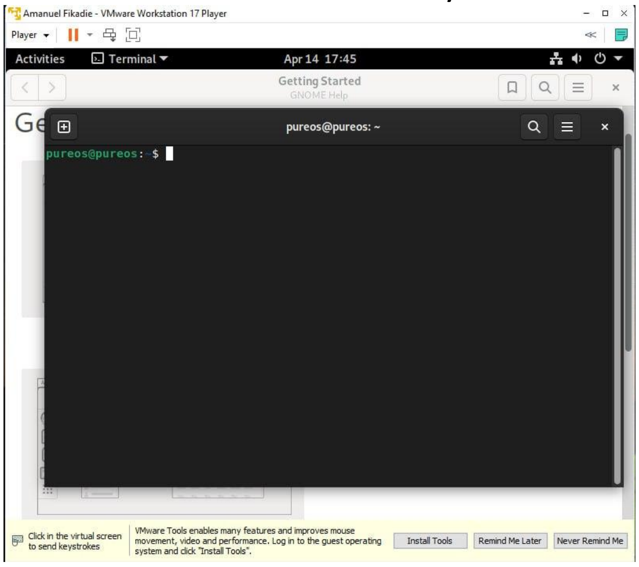


```
pureos@pureos:~$
```

Click in the virtual screen to send keystrokes

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

We must change this termminal as a proffesional man we should use black screen because it is very faster than the previous one



```
pureos@pureos:~$
```

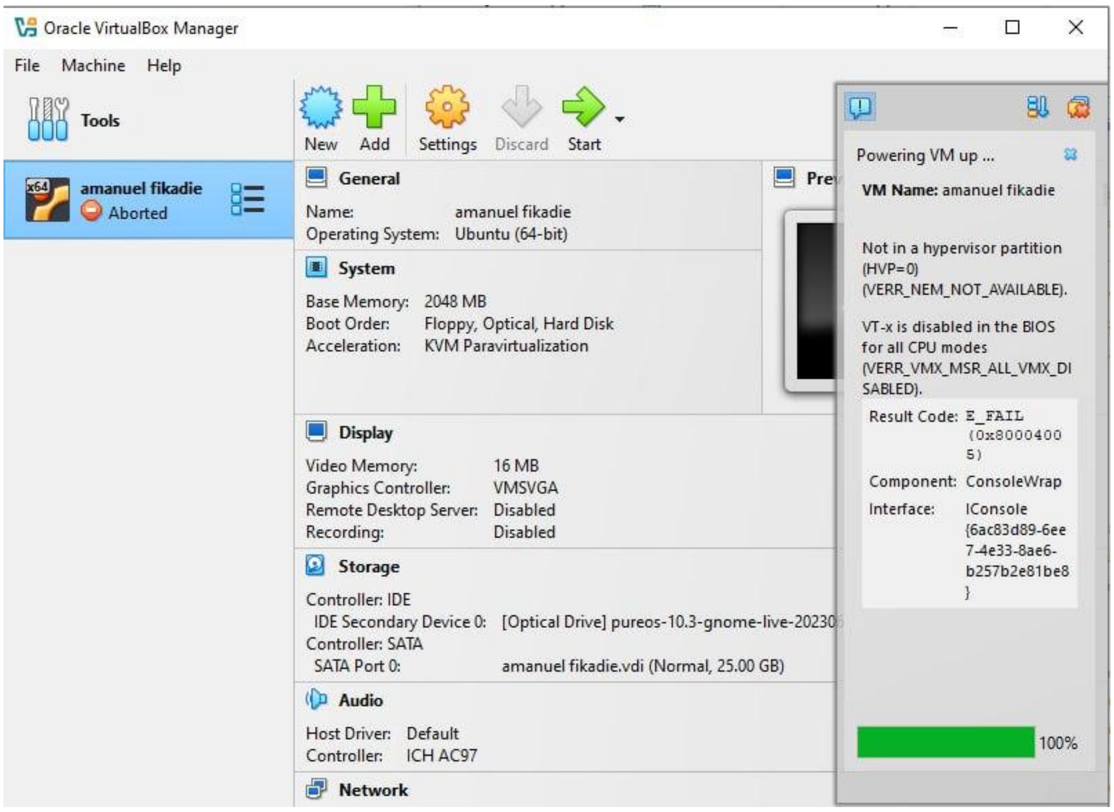
Click in the virtual screen to send keystrokes

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

Problem faced in this proccess-

network connection.

When I installed the file into the virtualbox it doesn't work it displays this error....And I solve it by using **VMware workstation.**



amanuel fikadie
Aborted

General
Name: amanuel fikadie
Operating System: Ubuntu (64-bit)

System
Base Memory: 2048 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: KVM Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Secondary Device 0: [Optical Drive] pureos-10.3-gnome-live-20230
Controller: SATA
SATA Port 0: amanuel fikadie.vdi (Normal, 25.00 GB)

Audio
Host Driver: Default
Controller: ICH AC97

Network

Powering VM up ...
VM Name: amanuel fikadie

Not in a hypervisor partition (HVP=0) (VERR_NEM_NOT_AVAILABLE).

VT-x is disabled in the BIOS for all CPU modes (VERR_VMX_MSR_ALL_VMX_DISABLED).

Result Code: E_FAIL (0x80004005)
Component: ConsoleWrap
Interface: IConsole {6ac83d89-6ee7-4e33-8ae6-b257b2e81be8}

100%

In **VMware workstation** there is another problem like “No 3D support is available from the host”.but it doesn’t have extra problem for this operating system I omit it because I can’t eliminate this problem.

Best Filesystem for PureOS Installation (System Partition)

- **ext4**

☑ Best option for Linux system installation

☑ Default filesystem in PureOS and most Linux distributions

☑ High-performance, stable, and journaling-enabled (prevents data corruption)

☑ Optimized disk usage and excellent community support

☑ Totally free and open source — aligns with PureOS philosophy

✔ Use ext4 when installing PureOS on the virtual hard disk.

Advantages and Disadvantages

PureOS is a Debian-based GNU/Linux operating system developed by Purism around three main goals: security, software freedom, and privacy of the user. It is approved by the Free Software Foundation (FSF) as a 100% free OS, making it unique among modern Linux distributions.

This is a critical overview of PureOS from the perspective of its technical design, usability, community, and applicability in the real world—particularly in the context of a virtualized, pure OS environment.

PureOS strengths

1. Total Commitment to Software Freedom

PureOS features only free/libre and open-source software (FOSS), with no proprietary drivers, firmware, or code. By doing this, it is one of the few operating systems to respect all four freedoms as established by the FSF:

The freedom to run the program for any purpose.
The freedom to study how the program works and modify it.
The freedom to distribute copies.
The freedom to distribute modified versions.

Example: Even Wi-Fi or graphics card firmware is not excluded unless under a free software license. This gives users full transparency into everything that is executing on the system.

2. Privacy-Focused by Design

PureOS is built with user privacy by default, rather than by option. It includes:

DuckDuckGo as the default search engine, with tracking-free search results.

The PureBrowser (formerly), and now the GNOME Web (Epiphany) browser with privacy extensions pre-installed.

No background data collection or telemetry.
Optional use of Librem One services integration (encrypted email, chat, VPN).
Impact: This makes PureOS ideal for users concerned about government surveillance, data exploitation, or corporate tracking—e.g., researchers, journalists, and activists.

3. Transparent and Secure Architecture
Since everything is open-source, PureOS provides complete auditability. Security experts and users can review any aspect of the system, something that cannot be done in proprietary operating systems such as Windows or macOS.

Features AppArmor for mandatory access control.
Promotes encrypted installations and secure boot.
Use Case: When executed on Purism's Librem hardware, the OS can be paired with hardware kill switches for Wi-Fi, camera, and microphone, making it one of the most secure consumer systems available.

4. Based on Debian – Inherits Stability and Package Availability
PureOS is based on Debian Testing, which offers access to thousands of free software packages while striking a balance between stability and newer updates.

Package manager: APT (Advanced Package Tool)
Compatible with.deb packages
Secure updates signed with GPG

Result: You obtain a stable OS with long-term support, ideal for development environments, education, and daily use — if hardware compatibility is guaranteed.

5. Lightweight and Efficient
PureOS does not contain unnecessary background services, bloatware, or unnecessary packages.

GNOME desktop is optimized.
System resources are conserved, making it suitable even for old hardware or virtual machines with low specs.

Boot time and shutdown are quick because of few services.
In VM Environments: This minimalism serves us well when PureOS is run within VirtualBox or other hypervisors, by lowering memory and CPU overhead.

6. Perfect for Developers, System Administrators, and Educators
PureOS offers a clean, scriptable, and customizable environment for:

Developing shell scripts, packaging, compiling software
Teaching students about how a Linux OS works internally
Performing security research in a free and verifiable environment

Educational Value: It promotes learning about fundamental OS concepts like init systems (systemd), file permissions, services (systemctl), and networking (ip, netplan, etc.).

PureOS Disadvantages

1. Limited Hardware Support
Since PureOS doesn't ship with proprietary firmware and drivers, a great number of hardware devices may not work out of the box without manual configuration or replacement.

Common issues: Wi-Fi chipsets, Bluetooth adapters, NVIDIA GPUs, fingerprint readers.

No support for closed-source GPU drivers like NVIDIA's proprietary driver.

Real Impact: Users will have to deal with no Wi-Fi, mediocre screen performance, or no peripheral support unless they switch to supported hardware or install non-free drivers (against PureOS policy).

2. Steep Learning Curve for Newbies
PureOS is not for Linux beginners. While the GNOME interface is intuitive, terminal knowledge and manual tweaking are required for a lot of tasks.

No out-of-the-box multimedia support (e.g., for MP3 or YouTube in some cases).
No advanced task GUI tools like disk partitioning post install.

Educational Challenge: The average user without Linux experience will struggle with operations like mounting drives, service management, or troubleshooting errors.

3. Less Preinstalled Software
To maintain its free and minimalistic nature, PureOS includes:

A browser
GNOME desktop
Standard system utilities

Missing: Office suite software, image editors, video players, PDF readers — must be installed by hand using the terminal.

Implication: Ideal for power users but not ideal for casual or non-technical users seeking a "ready-to-use" system.

4. Smaller Community and Fewer Support Resources
Compared to Ubuntu or Fedora:

Fewer tutorials and troubleshooting guides
Third-party support is scarce

Community forums are less responsive and smaller

Impact: It may take longer to fix unique issues or require more extensive research, especially for those users who install PureOS on non-supported hardware.

5. Multimedia and Proprietary Format Limitations
As PureOS avoids proprietary software, it doesn't support media codecs like MP3, H.264, or Flash out of the box.

Users must install free alternatives like gstreamer-plugins or vlc along with free codec packs.
Certain streaming websites may not work as desired.

Example: Watching a video on YouTube or Netflix may require installing additional software, which might not align with the FSF standards.

6. Unsuitable for Gaming or Graphics-Intensive Applications
PureOS doesn't support:

Steam (requires proprietary binaries)
Most modern games (rely on non-free graphics drivers)
High-end GPU acceleration (especially in VMs)

Use Case Limitation: This makes PureOS unsuitable for users who want to game, edit 4K video, or run intensive 3D modeling applications.

Overall Conclusion advantage and disadvantage
PureOS is a privacy-respecting, security-focused, and ethically developed Linux distribution. It is ideal for developers, students, researchers, and system programmers who are interested in transparency and control. However, because of its strict adherence to free software principles, it is less ideal for mainstream consumers, beginners, or individuals with newer proprietary hardware.

It is an excellent system for OSSP students to learn:

System architecture
Control based on the terminal
Filesystems
User management
System calls and shell scripting

Conclusion

The project investigated the installation and configuration of PureOS, a privacy-focused Linux distribution, within a virtual environment. With the use of virtualization technology, the project provided for safe and simple testing of the OS without altering the host system. The configuration of the virtual environment provided an environment of control to examine the inner workings and parts of an OS within a real-world but safe manner.

A significant part of the project concerned analyzing and implementing system calls—the vital interface through which user-level programs interact with the OS kernel. This labwork provided an in-depth comprehension of how operating systems allocate resources, schedule processes, interact with hardware, and offer protection. It also exposed basic concepts in system programming such as context switching, memory management, process scheduling, and file operations.

In addition, by engaging with PureOS—a Debian-based open-source operating system committed to freedom, privacy, and security—directly, the project allowed an opportunity to discover how modern Linux systems are structured, maintained, and shared. It highlighted the importance of open-source software in academic and professional research to allow transparency, personalization, and community engagement.

Overall, this project succeeded in fulfilling its objectives by:
Installing and configuring PureOS in a virtual environment.
Demonstrating the core concepts of virtualization, including abstraction, resource isolation, and performance compromise.

Testing and analyzing system calls, thereby solidifying knowledge of kernel-user space interaction.
Explaining how operating systems provide the low-level support for upper-level programs and user applications.
The real-world exposure that one gets by implementing this project sets a solid platform for advanced subjects in operating systems, security, and system programming.

Future Outlook / Recommendations

With the progression of technology, system-level programming and operating systems remain the core area of software development, especially in such areas as virtualization, cloud computing, embedded systems, and cybersecurity. Below are some suggested directions and future ideas based on the findings of this project:

1. Move ahead with Kernel Development
Having covered the essentials of system call implementation, the next activity can be to delve into Linux kernel modules. Kernel modules make it possible to add functionality to the kernel without recompiling or restarting the whole OS. As a possible next step, you might also want to look into:

Implementing your own device drivers.

Patching or modifying kernel-level elements.

Creating a minimal, bare-bones Linux kernel for learning purposes.

This would bring greater insight into the heart of the OS and the responsibilities of the kernel when it comes to managing hardware, memory, and system calls.

2. Containerization Technologies
While virtualization was at the center of this project, containerization is a more recent solution to environment isolation and deployment. Docker, Podman, and Kubernetes allow you to

wrap up applications with their dependencies in an extremely lightweight container, making them more portable and scalable. Knowing about:

Container vs VM architectures

How containers use OS-level virtualization (namespaces, cgroups),

Image building and orchestration

…will greatly enhance your understanding of system resource management and deployment practices in commercial environments.

3. Monitoring and Debugging Performance
System programming is usually all about tracking down sneaky bugs and performance issues. Excellent use of tools such as:

strace – tracing system calls,
gdb – debugging programs at the instruction level,

htop and vmstat – tracking process and memory usage,

perf – profiling CPU performance,

.can equip you with effective debugging and optimization abilities.

4. System Security Focus
PureOS places significant emphasis on privacy and security. Building from there, consider investigating how modern OSes implement:

User permission and access controls (e.g., POSIX permissions, ACLs),

Secure boot and kernel integrity checks, File system and process-level encryption,

Sandboxing techniques (e.g., AppArmor, SELinux, seccomp),

Kernel hardening and vulnerability mitigation.

Security is an ever-growing field, and awareness of OS-level security is highly critical to any system programmer or administrator.

5. Open-Source Contributions
The most significant benefit of using PureOS or any Linux-based environment is being able to read, modify, and contribute to the source code. Contributing to the PureOS or Debian communities by:

Bug fixing,

Improvement in documentation,

Feature development,

...can provide professional feedback and teamwork experience, and add to your resume and GitHub profile through real projects.

6. Automate System Tasks with Scripting
System administration and OS management effectiveness typically occurs through scripting. Learn to:

Write Bash scripts for automating installation and monitoring procedures.

Use Python for advanced system automation (e.g., file parsing, log analysis).

Write cron jobs or systemd timers for scheduling.

Automation skills are in highest demand and can be applied to other system-related fields.

7. Explore Filesystems and Bootloaders
More exploration of how filesystems (e.g., ext4, Btrfs) and bootloaders (e.g., GRUB) work will give you a general idea of the boot process, partitioning, mounting, and OS initialization.

8. Build Your Own Mini-OS
Subsequently, you can attempt to develop a minimalist OS using tools like Xv6 or Linux From Scratch (LFS). This is a more advanced task that involves applying all the system programming concepts, compilation, linking, memory management, and process scheduling.

Virtualization

Virtualization is a technology that creates a virtual (not real) replica of something, such as a computer system, server, storage device, network resource, or operating system. That is, it's the process of using software to simulate hardware functionality, allowing multiple operating systems and applications to be executed on one physical machine independently.

Instead of committing a single machine to a single use, virtualization allows one physical machine to host multiple virtual machines (VMs)—each behaving like an independent physical device.

The Core Idea

Essentially, virtualization separates the logical function of a computer from the physical hardware. Virtualization offers an abstraction layer where multiple operating systems can run on a single unit of physical hardware by sharing the underlying physical resources like CPU, memory, storage, and network.

Types of Virtualization (Categories)

There are different types of virtualization, each with a different purpose:

Hardware Virtualization: Most widespread kind; creates virtual machines on underlying hardware through a hypervisor.

Operating System Virtualization: Uses containers (e.g., Docker or LXC) to offer multiple isolated user-space instances atop one OS kernel.

Server Virtualization: Bundles multiple servers into fewer physical boxes by running VMs.

Storage Virtualization: Aggregates physical storage from many devices into one logical storage pool.

Network Virtualization: Combines hardware (e.g., switches and routers) with software to create virtual networks.

Desktop Virtualization: Allows users to run desktop environments remotely or virtually from a server (e.g., VDI - Virtual Desktop Infrastructure).

Main Elements of Virtualization

1. Host Machine:
The physical machine that provides hardware resources for virtualization.

2. Guest Machine (VM):
A software emulation of a physical computer that runs its own operating system and applications, independent of the host.

3. Hypervisor:
A critical software layer to enable multiple VMs on a host computer. It manages the creation, execution, and assignment of VMs to resources.

Two notable types:

Type 1 (Bare-metal hypervisors): Run directly on hardware. Examples: VMware ESXi, Microsoft Hyper-V, Xen.

Type 2 (Hosted hypervisors): Run on top of a previously installed OS. Examples: VirtualBox, VMware Workstation, Parallels.

4. Virtual Resources
Each VM is allocated virtualized hardware—virtual CPU, memory, disk space, network adapters—that simulate real hardware elements.

Virtualization in Action
When you install a hypervisor on a host computer, it adds a layer of abstraction between the operating systems and hardware. This abstraction layer allows the hypervisor to allocate slices of CPU time, memory, disk I/O, and other resources to each VM as needed.

Every VM runs its own guest operating system, which is unaware that it is running on virtualized hardware. The hypervisor intercepts all the hardware instructions and emulates them to corresponding physical hardware.

New processors have virtualization support through hardware-assisted virtualization extensions:

Intel VT-x
AMD-V

These features allow VMs to run more effectively by allowing the hypervisor to take advantage of the processor's built-in support for virtualization, reducing overhead.

Benefits of Virtualization
Resource Efficiency: Maximizes hardware use, eliminating waste.

Cost Savings: Fewer physical machines needed → less hardware, power, and maintenance costs.
Isolation: Each VM is isolated, which enhances security and stability.
Scalability: Easily scale resources up/down by adding or removing VMs.
Portability: VMs can be cloned, moved, or restored with ease.
Testing & Development: Isolated environments to test new software or systems.
Disaster Recovery: Easy backup, snapshot, and recovery facilities.
Legacy Support: Run old OSes and applications on newer hardware.

Real-Life Example of Virtualization

Assume you have a powerful laptop with 16 GB RAM and a quad-core processor. Without virtualization, you may be able to run only one operating system at a time—Windows 11.

With virtualization:
You can install a hypervisor (like VirtualBox).
You can run Linux, Windows 10, macOS, or any OS in isolated VMs.
You can allocate 4 GB RAM and 2 cores to each VM.
Each VM runs its own OS and apps, independently and concurrently.

This is the life of developers, testers, IT professionals, and cloud professionals nowadays.

Virtualization and Cloud Computing

Virtualization is what cloud computing relies on. When you consume services such as Amazon EC2 (AWS), Google Compute Engine (GCP), or Azure Virtual Machines, you are leasing a virtual machine that is situated on a physical server within a data center. The cloud provider uses virtualization to provide resources to different customers, all on shared hardware.

This enables multi-tenancy, elastic scaling, and rapid provisioning in the cloud—thus making it quick, cheap, and responsive to users worldwide.

Why Virtualization?

Virtualization is today a core technology in modern computing environments—especially in data centers, development environments, cloud infrastructure, and enterprise IT. Here is why it is being used and the benefits it offers:

1. Better Use of Hardware Resources (Efficiency)
Without virtualization, each physical machine used to run one operating system and perform a single function. This typically leaves sufficient CPU, memory, and storage unused.

✅ Virtualization enables you to run multiple virtual machines (VMs) on a single physical server, with each VM running yet another task. It utilizes the hardware to its maximum, thereby making systems more efficient.

2. Cost-Saving
Physical hardware is expensive to buy and maintain—both in terms of money and space. You need to spend money on:

Hardware (servers)
Electricity
Cooling
Maintenance
Physical space

✅ Virtualization reduces the need for physical servers, and that reduces all those costs. One powerful server can host several virtual machines instead of needing several separate machines.

3. Isolation and Security
Each virtual machine has its own isolated environment. If one VM crashes or gets malware, it doesn't affect other VMs or the host system.

☑ This isolation improves security and stability, especially in an environment where you need to test unsafe programs or permit several users to use the same computer.

4. Flexibility and Scalability
Virtual machines are simple to create, modify, or erase within a brief duration. You can:

Clone a VM
Make a template for reuse
Scale up/down resources (CPU, RAM)
Add or remove VMs according to demand

☑ This is what makes virtualization perfect for cloud computing, testing, and dynamic workloads that need to scale up or down rapidly.

5. Disaster Recovery and Backup
Virtualization software allows you to take snapshots or backups of virtual machines in most cases. When anything goes wrong, you can restore to a previous state in seconds.

☑ This makes recovery fast and sure, and simplifies backup procedures.

6. Easy Testing and Development
Developers often need to test their apps on different operating systems or environments.

☑ With virtualization, they can run multiple OSes on one machine (e.g., Windows, Linux, macOS) without needing extra physical devices.

7. Supports Legacy Systems
Some companies still utilize legacy software that can only run on older operating systems.

✅ Virtualization enables them to run older OSes on new hardware, so they don't need to keep older physical machines.

8. Portability
A virtual machine is merely a group of files. You can migrate or copy a VM to another machine or server without reinstalling everything.

✅ This simplifies migration, load balancing, or even moving systems from on-premise servers to the cloud.

9. Foundation of Cloud Computing
Cloud computing environments such as AWS, Azure, and Google Cloud make use of virtualization to run thousands of virtual machines for users across the globe.

✅ Virtualization would have made cloud computing at the scale and flexibility we enjoy today impossible.

How Virtualization Works in Today's Operating Systems

Virtualization is used in today's operating systems to run several virtual environments (virtual machines or containers) on one physical machine, which share its hardware resources like CPU, memory, and storage. Here is how it all works together:

1. The Hypervisor – The Virtualization Engine
At the heart of virtualization is a hypervisor. The hypervisor is a special software layer that creates and manages virtual machines (VMs) by acting as a bridge between the hardware and the guest operating systems.

There are two types of hypervisors:

Type 1 (Bare-metal hypervisor):

Executes directly on the physical hardware.
Example: VMware ESXi, Microsoft Hyper-V, Xen.

Used in enterprise data centers and cloud computing.
Type 2 (Hosted hypervisor):

Runs in a host operating system like any native application.
VirtualBox, VMware Workstation, etc.
Used by developers as well as home users for testing.
Role of Hypervisor:
Allocates hardware resources (CPU, RAM, disk) to each VM.
Keeps VMs isolated from one another.
Manages the communication between guest OS and hardware.

2. Virtual Machines (VMs) – Your Mini-Computer
Each virtual machine acts like a real, standalone computer. It includes:

A guest operating system (like Windows, Linux, etc.).

Virtual hardware (virtual CPU, RAM, network card, storage, etc.).

Even though it's running on another system, the guest OS is sure that it's running on a physical machine. This is because the hypervisor is fooling it by emulating hardware.

You may run multiple VMs on a single machine, depending on resources available.

3. Virtual Hardware – Imitating Real Components
Each virtual machine has its own:

Virtual CPU
Virtual Memory
Virtual Disk (usually stored as a file)
Virtual Network Interface

Software implementations but functioning exactly like physical hardware. Hypervisor manages allocating these virtual devices to the underlying hardware.

4. Hardware-Assisted Virtualization

The CPUs of modern times by AMD and Intel do support hardware virtualization:

Intel VT-x
AMD-V

These technologies allow VMs to run more natively on the CPU, making it more secure and perform better. They allow the hypervisor to offload some of the effort to the hardware, speeding up virtualization and making it more efficient.

5. Isolation and Security
Every virtual machine is isolated from others. That is:
One VM crashing does not affect others.

Malware in one VM cannot infect another VM.
You can test viruses, software updates, or test apps safely.

6. Snapshots and Cloning
Operating systems today include features like:

Snapshots: Save the exact state of a VM (memory, disk, settings) at a point in time. In case it breaks, roll it back instantly.

Cloning: Make an exact copy of a VM for testing or deployment.
This is highly useful for development, testing, and disaster recovery.

7. Virtualization and Containers (OS-Level Virtualization)
Modern OSes (especially Linux) also offer another form of virtualization through containers instead of virtual machines.

Containers (like Docker, LXC) are the same kernel of an OS but execute different applications and environments.

They are lightweight, faster to boot, and less resource-intensive than full VMs.

Containers have widespread use in cloud computing, DevOps, and microservices architecture.

8. Cloud and Data Centers Integration
Virtualization in modern OSes forms the foundation of cloud computing. Virtualization is used by cloud providers like AWS, Azure, and Google Cloud for:

Running thousands of VMs on enterprise servers.

Relieving resource constraints through dynamic resource allocation to customers.

Providing high availability, scalability, and cost-effectiveness.

Modern operating systems like Linux, Windows Server, and macOS are natively built to offer these virtualization features.

System calls

What is `nice()`?

The `nice()` system call lets you **change the priority** of a process. Lower niceness = higher priority. Higher niceness = lower priority.

Formula:

- Default niceness = `0`
- Min = `-20` (highest priority)
- Max = `19` (lowest priority)

✓ How to Implement **nice()** in PureOS

Step-by-step: Write a C Program that uses `nice()`

---

1. Open Terminal and Create a C File

```
bash
Copy code
nano nice_example.c
```

---

2. Add the Following Code

```
c
Copy code
#include <stdio.h>#include <unistd.h>#include
<sys/resource.h>#include <errno.h>
int main() {
    int current_nice = getpriority(PRIO_PROCESS, 0);
    printf("Current process nice value: %d\n",
current_nice);

    int result = nice(10);  // Increase niceness
(lower priority)
    if (result == -1 && errno != 0) {
        perror("nice failed");
        return 1;
    }

    int new_nice = getpriority(PRIO_PROCESS, 0);
    printf("New process nice value: %d\n",
new_nice);

    // Dummy task
    for (int i = 0; i < 5; i++) {
        printf("Running process work loop %d\n", i
+ 1);
        sleep(1);
    }

    return 0;
}
```

---

3. Save and Compile

Press:

`Ctrl + O` → Enter (to save)

`Ctrl + X` (to exit)

Then compile:

```
bash
Copy code
gcc nice_example.c -o nice_example
```

---

4. Run the Program

```
bash
Copy code
./nice_example
```

You will see output like:

```
arduino
Copy code
Current process nice value: 0
New process nice value: 10
Running process work loop 1
...
```