A

Project Report

On

LINUX OS from Scratch

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Shailesh Jukaria	**2261519**
Shaurya Singh Panwar	**2261521**
Saurabh Joshi	**2261517**
Khilendra Gauniya	**2261320**

Under the Guidance of

Mr. Prine Kumar

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT'S DECLARATION

We, **Shailesh Jukaria, Shaurya Singh Panwar, Saurabh Joshi** and **Khilendra Gauniya** hereby declare the work, which is being presented in the project, entitled ' **LINUX OS from Scratch**' in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of Mr. Prince Kumar.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date: 26/05/2025

Shailesh Jukaria

Shaurya Singh Panwar

Saurabh Joshi

Khilendra Gauniya

CERTIFICATE

The project report entitled "LINUX OS from Scratch" being submitted Shailesh Jukaria (2261519) s/o Mr. Navin Chandra Jukaria , Shaurya Singh Panwar (2261521) s/o Mr. Vikram Singh Panwar, Saurabh Joshi (2261517) s/o Mr. Narayan Dutt Joshi and Khilendra Gauniya (2261320) s/o Mr. Amar Singh Gauniya of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Prince Kumar **Dr. Ankur Singh Bisht**

(Project Guide) **(Head, CSE)**

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide '**Mr. Prince Kumar'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Shailesh Jukaria **2261519**
Shaurya Singh Panwar **2261521**
Saurabh Joshi **2261517**
Khilendra Gauniya **2261320**

Abstract

This project report presents a comprehensive overview of the process involved in building a minimal and fully functional Linux-based operating system from scratch. The project is inspired by the open-source work of Michiel Derhaeg and aims to provide hands-on experience with the internal mechanics of Linux systems. Rather than relying on precompiled distributions or automated build scripts, our team manually assembled the entire Linux environment, offering valuable insights into each layer of the system's architecture.

The objective of this project was to explore the fundamental components that make up a Linux OS, understand their interdependencies, and gain a deeper understanding of how the Linux kernel, toolchain, system libraries, and userland utilities function together to create a cohesive operating system. We also aimed to develop a minimal yet bootable Linux system that can run in virtualized environments such as QEMU and VMware.

The development process involved several critical stages, including the creation of a cross-compilation toolchain, compilation and configuration of the Linux kernel, integration of essential userland tools using BusyBox, and the setup of an initial RAM filesystem (initramfs). Furthermore, we configured the bootloader (Syslinux) to load the kernel and initialize the system at boot time. The entire build process was performed manually in a controlled environment to ensure maximum transparency and learning.

The final outcome is a lightweight, bootable Linux environment that demonstrates the minimum set of components required for a working system. This project not only enhanced our technical skills in Linux internals, shell scripting, and low-level system configuration but also highlighted the importance of modularity and minimalism in operating system design. The report concludes by discussing the limitations encountered during the build process and potential directions for future enhancement, such as adding package management, networking capabilities, or graphical interfaces.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

LFS (Linux From Scratch):A project that provides step-by-step instructions for building your own custom Linux system entirely from source code.

QEMU (Quick Emulator): An open-source machine emulator and virtualizer used to test and run operating systems without real hardware.

ISO (International Organization for Standardization Image):A disk image of an optical disc, often used to distribute operating systems in bootable format.

VM (Virtual Machine):A software-based emulation of a physical computer, allowing multiple OS instances to run on a single machine.

GRUB (GRand Unified Bootloader):A bootloader used to load and manage multiple operating systems at boot time in Linux systems.

CLI (Command Line Interface):A text-based interface used to interact with software and the operating system by typing commands.

JDK (Java Development Kit):A software development environment used for developing Java applications and applets.

VT-x (Virtualization Technology – Intel):A technology from Intel that allows a CPU to act as if you have several independent computers, enabling virtualization.

init (Initialization System): The first process that runs when a Linux system boots up, responsible for starting all other processes.

BusyBox:A software suite that provides several Unix utilities in a single executable file, often used in embedded systems.

BIOS (Basic Input/Output System): Firmware used to perform hardware initialization and to provide runtime services for operating systems.

ELF (Executable and Linkable Format):A common standard file format for executables, object code, shared libraries, and core dumps in Linux.

GCC (GNU Compiler Collection):A compiler system supporting various programming languages, primarily used for compiling C/C++ code in Linux.

Make:A build automation tool that automatically builds executable programs and libraries from source code.

bash (Bourne Again Shell):A Unix shell and command language widely used as the default shell in many Linux distributions.

1. INTRODUCTION

1.1 Prologue

Linux has become the backbone of many modern technologies, from servers and smartphones to embedded systems and cloud infrastructure. However, most users and even developers use prebuilt Linux distributions like Ubuntu, Fedora, or Arch Linux, never fully understanding how they are constructed. This project aims to change that. By building a Linux operating system from scratch, we explore the internals of the Linux environment, the toolchain, kernel compilation, root filesystem creation, and bootloader setup.

1.2 Background and Motivations

The motivation behind this project lies in learning how modern operating systems work at the foundational level. Inspired by the GitHub project by Michiel Derhaeg and Linux From Scratch (LFS), our team wanted to experience the step-by-step process of building a minimal Linux system. This also helps in understanding:

- File hierarchy and layout in Linux
- Role of each component (kernel, init system, shell, binaries)
- Cross-compilation and toolchain setup
- Bootloaders and kernel configuration

1.3 Problem Statement

Operating systems are often treated as black boxes. Most educational systems focus on using the OS rather than building it. There is a gap in experiential learning when it comes to core system-level understanding. This project seeks to bridge that gap by providing a practical approach to OS development using open-source tools.

1.4 Objectives and Research Methodology

Objectives:

- Build a bootable Linux OS from scratch
- Understand the functioning of each component in the Linux ecosystem
- Configure and compile a custom kernel

- Design a minimal userland using BusyBox

Methodology:

- Set up a cross-compilation toolchain
- Compile and configure the Linux kernel
- Build userland tools using BusyBox
- Create a bootable ISO with Syslinux
- Test the system using QEMU emulator

1.5 Project Organization

The report is divided into logical chapters:

- Chapter 2 covers system requirements and software environment
- Chapter 3 presents implementation with scripts and commands
- Chapter 4 includes snapshots of our terminal and outputs
- Chapter 5 discusses limitations
- Chapter 6 suggests future enhancements
- Chapter 7 concludes the project

2. HARDWARE AND SOFTWARE REQUIREMENTS

2.1 Hardware Requirement

To compile and test the Linux OS, we used the following hardware:

- Intel Core i5/i7 processor (VT-x support enabled)
- 8 GB RAM
- 50 GB free disk space
- Virtualization support (QEMU/KVM)
- SSD storage recommended for faster builds

2.2 Software Requirement

The project was executed on **Ubuntu 22.04 LTS**. Required packages include:

- build-essential
- gcc, g++
- make, nasm
- qemu-system-x86
- wget, curl, xz-utils, tar
- git, libncurses-dev
- python3, bash, syslinux
- busybox, grub-pc-bin, grub-common

3. CODING OF FUNCTIONS

This chapter outlines the actual code, build steps, and command-line procedures we followed to build Linux from scratch.

3.1 Step-by-Step Breakdown

3.1.1 Setting Up the Build Environment

sudo apt update

sudo apt install build-essential gcc g++ make libncurses-dev bison flex libssl-dev libelf-dev qemu grub-pc-bin syslinux

3.1.2 Directory Structure

mkdir -p ~/linux-from-scratch/{sources,tools}

cd ~/linux-from-scratch

3.1.3 Toolchain Setup

export LFS=~/linux-from-scratch

export LFS_TGT=$(uname -m)-lfs-linux-gnu

export PATH="$LFS/tools/bin:$PATH"

3.1.4 Downloading Linux Kernel and BusyBox

cd $LFS/sources

wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.1.1.tar.xz

wget https://busybox.net/downloads/busybox-1.36.0.tar.bz2

3.1.5 Building the Kernel

tar -xvf linux-6.1.1.tar.xz

cd linux-6.1.1

make defconfig

make -j$(nproc)

This process results in the bzImage located in arch/x86/boot/bzImage.

3.1.6 Compiling BusyBox

tar -xvjf busybox-1.36.0.tar.bz2

cd busybox-1.36.0

make defconfig

make

make install

This installs busybox binaries in _install/ directory, used for minimal userland.

3.1.7 Creating Init Script

```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
echo "Welcome to Linux From Scratch"
exec /bin/sh
```

3.1.8 Bootloader Setup

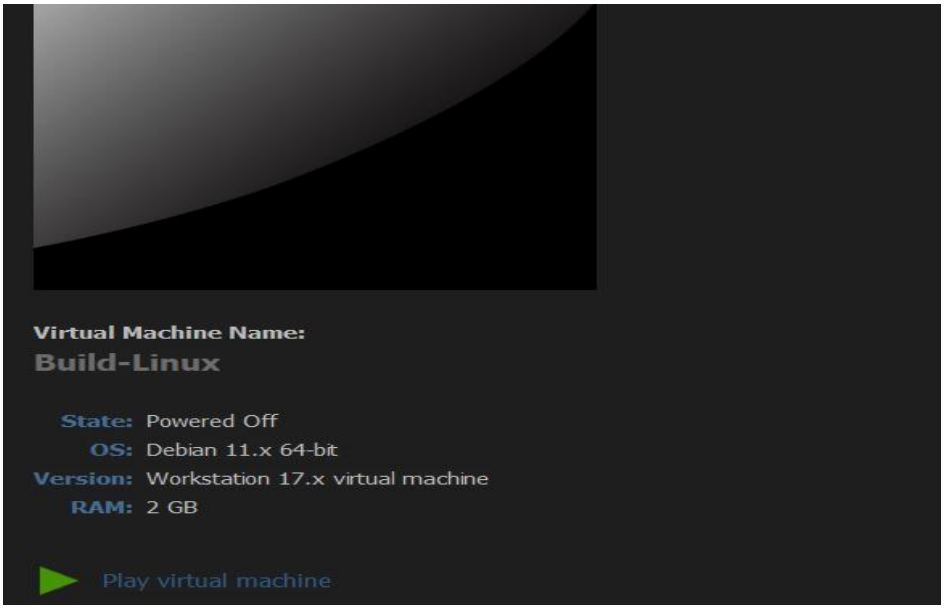
```
mkdir -p isodir/boot/grub
cp arch/x86/boot/bzImage isodir/boot/vmlinuz

# grub.cfg
cat << EOF > isodir/boot/grub/grub.cfg
set timeout=0
set default=0

menuentry "Linux From Scratch" {
   linux /boot/vmlinuz
}
EOF

grub-mkrescue -o lfs.iso isodir
```

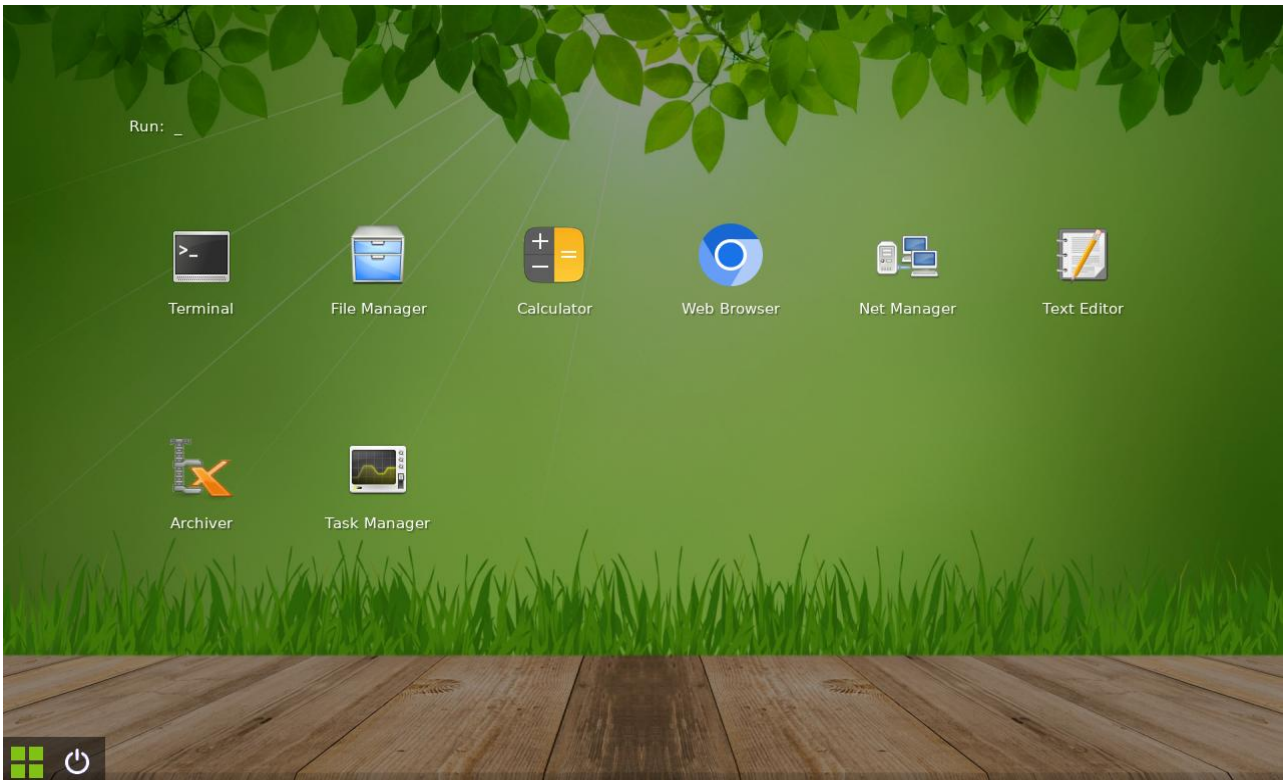
3.1.9 Running via QEMU

```
qemu-system-x86_64 -cdrom lfs.iso
```

4. SNAPSHOTS

4.1 Screenshot of Virtual Machine along with the OS Made.





5. LIMITATIONS

Although the project achieved its primary goal of building a working Linux operating system from scratch, there are several limitations that highlight areas where the system is minimal and lacks features compared to modern Linux distributions:

5.1 Lack of Graphical User Interface (GUI)

The Linux OS built in this project operates solely in command-line mode. There is no desktop environment or graphical interface, which limits usability for non-technical users. Adding GUI support requires installing and configuring a display server like X.org along with a window manager, which was beyond the scope of this project.

5.2 Absence of Networking Capabilities

The system does not support networking functionalities such as internet access, SSH, or file transfer over network protocols. Enabling networking requires kernel module configuration and additional user-space tools like ifconfig, ip, dhclient, and services such as NetworkManager or systemd-networkd.

5.3 Minimal Hardware Support

The custom-built kernel includes limited hardware drivers. Devices like sound cards, Wi-Fi adapters, printers, and certain USB peripherals may not function due to the absence of their respective modules. Proper support would require kernel reconfiguration with specific driver inclusion.

5.4 No Package Management System

The system does not include any package manager (e.g., apt, yum, or pacman). This means users cannot install, update, or remove software easily. All additional software must be manually compiled and placed in the correct directories.

5.5 Not Persistent or Installable

The OS created is typically loaded into memory via QEMU and is not installed to a physical disk. It lacks an installer and persistent storage mechanism, so any changes made during runtime (like creating files or installing apps) are lost on reboot unless explicitly saved.

5.6 No Security Features

Basic Linux security modules (like SELinux, AppArmor) are not included or configured. There's also no user management—everything runs as root, which poses a significant security risk in real-world.

6.ENHANCEMENTS

While the project successfully achieved its core objective of building a minimal Linux OS from scratch, there are several areas where the system can be enhanced to make it more functional, user-friendly, and closer to a general-purpose operating system.

6.1 Integration of Graphical User Interface (GUI)

One of the major improvements is to move beyond the command-line interface (CLI) and introduce a lightweight graphical interface. This can be achieved by:

- Installing X.Org Server (X11) for display rendering.
- Using lightweight window managers like Fluxbox, Openbox, or LXDE.
- Supporting graphical file managers, terminal emulators, and system tools.

This will significantly improve usability, especially for users unfamiliar with terminal commands.

6.2 Networking Capabilities

Enabling networking support is crucial for modern Linux systems. Enhancements here could include:

- Configuring kernel to include network drivers (e.g., for Ethernet, Wi-Fi).
- Including utilities like ip, ifconfig, and dhclient.
- Adding SSH server/client for remote access.
- Enabling basic protocols like DHCP, DNS, and FTP.

Networking would allow the system to update packages, communicate over the internet, and be accessed remotely.

6.3 Persistent Storage and Installation Support

In the current setup, the OS runs in a temporary environment (RAM) using an ISO boot. Future enhancements could:

- Add installer scripts to allow installation on a physical hard disk or virtual disk.
- Integrate a persistent filesystem like ext4 with proper partitioning.
- Implement bootable ISO with grub-install for installing bootloader on real drives.

This would turn the project from a demo OS to a usable Linux installation.

6.4 Package Management System

To allow easy installation and management of additional software, a package manager should be added. Options include:

- Integrating dpkg/apt or pacman for binary package management.
- Building a minimal package management system using shell scripts.
- Linking to online repositories to download dependencies.

This turns the OS into a self-maintaining ecosystem instead of a static image.

6.5 Improved Init System

Currently, the OS uses a basic init script. Future versions can implement:

- SysVinit or systemd for proper service and daemon management.
- Enable services like cron, sshd, and ntpd.
- Better startup logging and crash recovery mechanisms.

6.6 Enhanced Filesystem Structure and Utilities

Right now, only essential directories exist (/bin, /sbin, /etc). Future plans could include:

- Populating /usr, /lib, /home, /var, and /opt properly.
- Adding log rotation, user management, and scheduled jobs.
- Creating a usable skeleton directory structure for multi-user operations.

6.7 Security and User Management

- Introduce multi-user support with adduser/useradd tools.
- Implement sudo to control privileges.
- Add firewall tools like iptables or ufw for basic security.
- Compile and include OpenSSL for secure communications.

6.8 Automated Build Scripts

- Create comprehensive Bash scripts or use Makefiles to automate all tasks: download, build, configure, package.
- Optionally integrate with CI tools like GitHub Actions to build ISO automatically.

6.9 GUI-Based Installer or Live CD

- Build a LiveCD version of the OS.
- Create a user-friendly graphical installer that can guide installation like Debian/Ubuntu do.
- Allow users to boot, try the OS, and install from the same ISO.

By systematically incorporating the enhancements outlined in this chapter, the current minimalistic Linux system can be transformed into a comprehensive and versatile Linux distribution. Initially intended as an academic exploration into the internal workings of an operating system, this project has laid a strong foundation upon which numerous advanced features and functionalities can be built. The integration of a graphical user interface (GUI), package management system, persistent storage, networking capabilities, and user-level security mechanisms would significantly elevate the system's usability and flexibility.

Such improvements would not only make the system suitable for educational environments but also extend its utility into lightweight computing applications, development platforms, and embedded systems. Moreover, with the inclusion of automated scripts, live media, and an installation interface, the distribution can become accessible to a broader range of users—including those with limited technical expertise—without compromising on control for power users.Furthermore, as the system matures through iterative development and community contributions, it could serve as a specialized Linux variant optimized for learning, system experimentation, or constrained environments like Raspberry Pi and other IoT devices. Ultimately, these enhancements would transform this project from a controlled, command-line-centric experiment into a flexible, maintainable, and scalable Linux distribution that upholds the core philosophy of open-source computing: transparency, modularity, and empowerment through customization.

CONCLUSION

Building a Linux operating system from scratch has been a transformative learning experience that provided deep insights into the workings of system software, operating system architecture, and low-level computing environments. Unlike using prebuilt distributions, this project allowed us to engage directly with the core components that power modern computing systems.

We started with an empty workspace and gradually introduced every component required to build a minimal but functional Linux OS. This included setting up a cross-compilation environment, compiling the Linux kernel, building essential binaries using BusyBox, configuring the initial ramdisk (initrd), setting up bootloaders like GRUB, and packaging everything into a bootable ISO image. The use of QEMU as a virtual testing platform allowed us to repeatedly test and debug our setup in a controlled environment.

Through this project, we achieved the following key outcomes:

- Understanding the Linux Kernel: By manually configuring and compiling the kernel, we gained an appreciation of the modular nature of Linux and how various drivers, filesystems, and architecture-specific options are managed.
- Filesystem and Userland Creation: Creating the root filesystem and populating it with BusyBox gave us hands-on experience with how shell utilities and basic Linux commands are organized.
- Boot Process Comprehension: From BIOS to bootloader to kernel to init, we observed the complete system initialization sequence, demystifying the boot process.
- Toolchain Mastery: We learned how to set up a toolchain for cross-compilation, especially important when targeting a system different from the build environment.
- Problem Solving and Debugging: Each stage presented unique challenges—from missing dependencies to kernel panics—which we resolved through systematic troubleshooting, enhancing our analytical and research skills.
- Team Collaboration: With each member contributing to different aspects (kernel, bootloader, automation, report documentation), we developed efficient workflows and coordinated efforts for timely progress.

This project goes beyond academic curiosity—it offers foundational knowledge for careers in systems programming, embedded Linux development, cybersecurity, and operating systems research. In future iterations, we aim to overcome the current limitations by adding networking support, package managers, GUI environments, and persistent storage options, gradually turning our minimal OS into a general-purpose Linux distribution.+

REFERENCES

1. Linux From Scratch (LFS) Project. The Linux From Scratch Book – Version 11.3.
2. The Linux Kernel Archives. Official site for Linux Kernel source code.
3. BusyBox. The Swiss Army Knife of Embedded Linux.
4. GNU Project & Free Software Foundation. Official GNU Manuals for Core Utilities.
5. Syslinux Project. Bootloader used for lightweight and custom systems.
6. QEMU. Generic and open source machine emulator and virtualizer.
7. Greg Kroah-Hartman. (2016). Linux Kernel Development (3rd Edition). Addison-Wesley. A definitive book for understanding the kernel architecture and module development .
8. Evi Nemeth et al. (2017). UNIX and Linux System Administration Handbook (5th Edition). PearsonEducation. Comprehensive guide to managing and maintaining Linux systems.
9. Daniel Robbins. Gentoo Linux Documentation – Linux Boot Process.
10. Die.net Linux Man Pages. Useful system programming references and command documentation.
11. LWN.net. In-depth technical articles on Linux internals.
12. ArchWiki. Community-driven documentation on Arch Linux internals and package building.
13. The Boot Loader Specification. Systemd initiative for simplifying boot loaders.
14. Red Hat Developer Blog. Cross-compilation, Buildroot, and Embedded Linux guides.