AIR University Islamabad

Final Project Report

Submitted By:

- Haris Sarfraz (231296)
- Uzair Farooq (231338)
- Swaira Urooj (231264)

Course: Operating System

Instructor: Miss Maryam Mehmood

Submission Date: 12-June-2025

Introduction

Operating systems are the backbone of modern computing, yet their inner workings often remain abstract to students. This project aimed to demystify that by building a minimal kernel using the Rust programming language. The kernel was designed to run on bare metal — meaning it communicates directly with the hardware, without an underlying operating system.

Rust was chosen due to its strong emphasis on safety and control, both crucial in low-level system programming. Our kernel demonstrates direct VGA memory writing, a custom panic handler, and the use of essential Rust libraries suited for system-level development.

Tools & Environment:

We utilized the following tools and libraries in our development process:

- ✓ **Rust** – A safe, modern systems programming language
- ✓ **QEMU** – A virtual machine emulator to test the kernel
- ✓ **bootimage** – Compiles Rust code into a bootable binary
- ✓ **cargo-xbuild** – Builds Rust code for custom OS targets
- ✓ **lazy_static** and **spin** – For global state and safe concurrency
- ✓ **volatile** – Ensures hardware memory is read/written correctly.

Design and Implementation:

Our kernel is designed to work without the standard library (#![no_std]) and doesn't use the usual main() function (#![no_main]). Instead, it starts from a custom function _start(), which is the actual entry point.

The kernel writes directly to the VGA text buffer — a section of memory starting at address 0xb8000. Each character displayed on the screen is represented using 2 bytes: one for the ASCII character and one for the color.

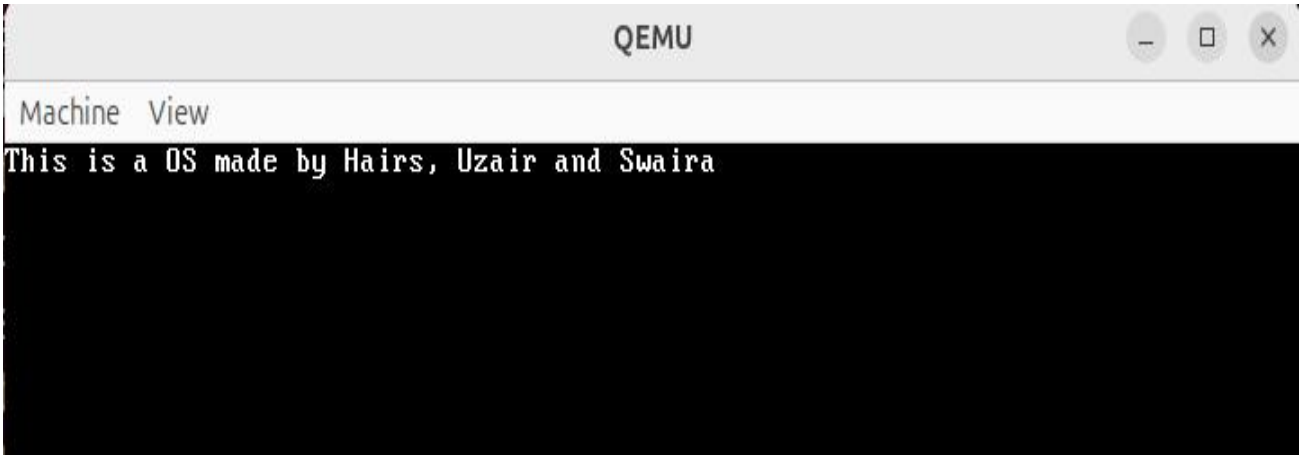
A custom panic handler is also written to handle runtime errors, which would normally crash a program. In our case, it simply enters an infinite loop to prevent the system from progressing in an undefined state.

Note:

The full source code is provided in the attached .zip file titled osproject.zip.
It contains all Rust files, configurations, and dependencies required to build and run the minimal kernel.
Please refer to the main.rs and vga_buffer.rs files inside the /src directory for implementation details.

Output Screenshot:

This is the result of running our kernel using QEMU.
The kernel displays a message by writing directly to the VGA text buffer.



Challenges Faced:

- ✓ **No debugging tools**: We couldn't use a traditional debugger, so we had to rely on printing to screen.
- ✓ **No standard library**: We had to reimplement basic things manually.
- ✓ **Manual memory management**: Writing to raw memory (like VGA buffer) requires precision and care.

What We Learned:

This project helped us understand:
- ✓ Boot processes of an OS
- ✓ How to write directly to memory using Rust
- ✓ Importance of safety in low-level systems
- ✓ How a minimal kernel functions without an OS

Conclusion:

Building a minimal kernel taught us the real-world complexity of low-level programming. Even writing text to a screen required working with memory directly, defining data structures, and handling edge cases manually. The project gave us valuable experience in system programming using Rust and a strong foundation to build more advanced operating system features in the future.

THE END