kacchiOS: Design and Implementation of a Minimal Bare-Metal Operating System

Operating Systems Laboratory Project

Group D

John Kumar (2103122)
Monjurul Islam (2103132)
Usman (2103145)
Umor Faruk (2103130)
Nurul Huda (2103141)
Rimon (2103181)

Submitted to:
Farhan Shakib
Lecturer

Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Submission Date: January 5, 2026

Abstract

This project describes the design and implementation of *kacchiOS*, a minimal bare-metal operating system developed as part of an operating systems laboratory assignment. Starting from a provided OS skeleton, three core components were implemented: memory management, process management, and CPU scheduling. A simple stack-based memory allocator, a fixed-size process table, and a Round Robin scheduler were designed to demonstrate fundamental operating system concepts such as process creation, context switching, and fair CPU sharing. The operating system boots successfully in a bare-metal environment and was tested using the QEMU emulator. The project emphasizes conceptual clarity and educational value rather than performance optimization.

1 Introduction

An operating system is responsible for managing hardware resources and providing an execution environment for programs. Understanding the internal working of an operating system is essential for students of computer science and engineering. This project aims to provide hands-on experience with core operating system mechanisms by extending a minimal bare-metal operating system called *kacchiOS*.

The base version of kacchiOS includes a bootloader, basic kernel initialization, serial input/output, and a null process. The objective of this assignment was to implement missing core subsystems—memory management, process management, and CPU scheduling—while keeping the design simple, modular, and easy to understand.

2 System Overview

kacchiOS is a 32-bit x86 bare-metal operating system that executes without relying on any host operating system services. It is loaded directly into memory and begins execution at the kernel entry point.

System Characteristics

- Architecture: x86 (32-bit)
- Execution Mode: Bare-metal
- Input/Output: Serial port (COM1)
- Testing Environment: QEMU Emulator
- Build Tools: GCC, GNU Assembler, GNU Linker

The system follows a modular design, where memory management, process management, and scheduling are implemented as separate components and integrated by the kernel.

3 Memory Management

3.1 Design Approach

The memory manager is responsible for providing stack memory to processes. To keep the design simple and deterministic, a static memory pool is used together with a bump-pointer allocation strategy. Each process is allocated a fixed-size stack from this pool.

This approach avoids memory fragmentation and is sufficient for demonstrating process execution and context switching in an educational operating system.

3.2 Implementation Details

- A global memory pool is statically allocated
- A memory offset tracks the next free location

- Stack memory is allocated during process creation
- Each process receives a private stack region

Heap memory management was not implemented, as it was not required for the core objectives of this assignment.

4 Process Management

4.1 Process Control Block

Each process is represented by a Process Control Block (PCB), which stores essential information required to manage and schedule processes. The PCB contains the process ID, stack pointer, and current process state.

4.2 Process States

The following process states are supported:

- **PR_FREE**: Unused process table entry
- **PR_READY**: Process is ready to be scheduled
- **PR_CURR**: Process is currently executing
- **PR_TERM**: Process has terminated

4.3 Process Lifecycle

Processes are created by allocating a free PCB and assigning stack memory. Newly created processes enter the READY state. When a process finishes execution, it transitions to the TERMINATED state.

5 CPU Scheduling

5.1 Scheduling Policy

kacchiOS implements a **Round Robin scheduling policy**, which is commonly used in time-sharing operating systems. In this approach, each process in the READY state is assigned the CPU for a fixed time slice in a cyclic order.

A key advantage of the Round Robin algorithm is that it **ensures fairness among processes and prevents starvation**. Since every ready process is placed in a FIFO ready queue and is guaranteed to receive CPU time within a bounded interval, no process can be indefinitely postponed.

5.2 Ready Queue Management

The scheduler maintains a FIFO ready queue that stores all READY processes. Processes are selected from the queue in the order in which they become ready.

5.3 Context Switching

Context switching is implemented by saving the stack pointer of the currently running process and restoring the stack pointer of the next scheduled process. Although simplified, this mechanism effectively demonstrates how execution can be transferred between processes.

6 Kernel Integration

During system startup, the kernel initializes the serial driver, memory manager, process manager, and scheduler. Multiple processes are created, and the scheduler is invoked repeatedly. The null process remains active to handle serial input and output.

7 Testing and Results

The operating system was tested using the QEMU emulator. The following results were observed:

- Successful system boot
- Correct display of welcome message
- Proper serial input and output
- Correct scheduling behavior

8 Challenges and Solutions

Challenges

- Implementing context switching in a freestanding environment
- Coordinating interaction between OS subsystems

Solutions

- Used a simplified stack-pointer-based context switch
- Adopted fixed-size data structures

9 Limitations and Future Work

Current Limitations

- No dynamic heap allocation
- No hardware timer for preemptive scheduling
- Limited process states
- No priority differentiation, although starvation is prevented

Future Enhancements

- Heap memory management
- Timer-based preemptive scheduling
- Priority scheduling and aging
- Inter-process communication

10 Conclusion

This project successfully demonstrates fundamental operating system concepts by implementing memory management, process management, and CPU scheduling in a bare-metal environment. The modular and minimal design of kacchiOS makes it well-suited for educational purposes and provides a strong foundation for further exploration of operating system internals.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, Wiley.

[2] OSDev Wiki. https://wiki.osdev.org

[3] XINU Operating System Documentation.