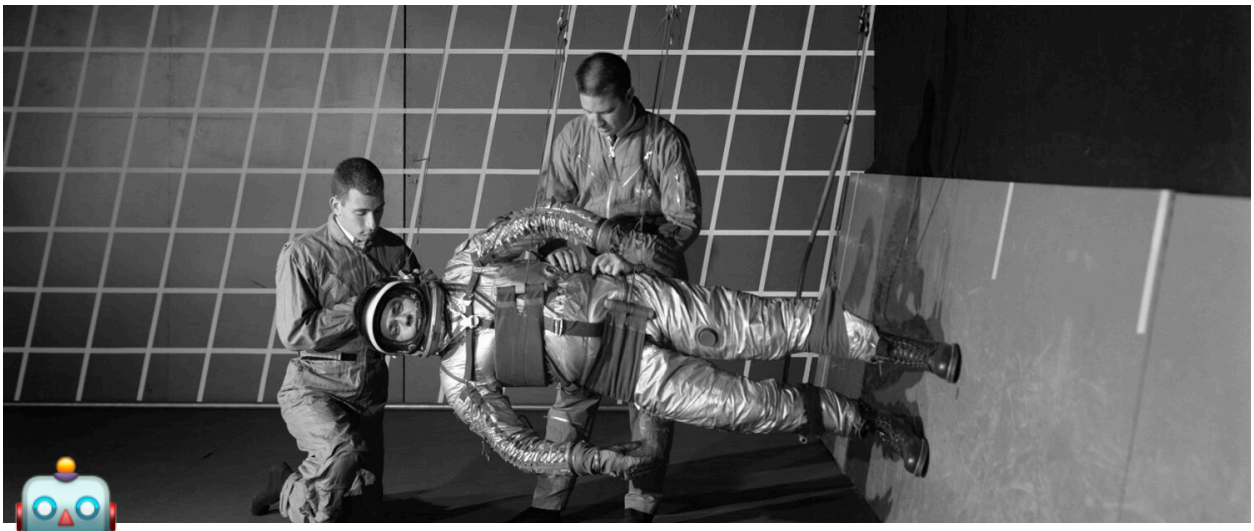


BMO OS - SRS

Document Purpose and Audience

> The purpose of this document is to describe the functional and non-functional requirements for a BMO **Operating System (BMO OS)**.

> This document is intended for **Students and Instructors:** who will use the (BMO OS) as a learning tool for understanding OS concepts

Introduction

Software Purpose

> The BMO OS aims to simulate basic features of an operating system, including process management, memory management, and graphic user interface (GUI), to help students understand the core principles of OS design.

Software Scope

The **BMO OS** will provide a simple environment that can execute commands, manage programs, and demonstrate key OS concepts.

It will run on a simulated environment or directly on hardware with limited functionality.

Definitions, and abbreviations

- OS : operating system
- BMO OS : project name
- GUI ：graphic user interface
- CPU : Central Processing Unit
- **CLI : Command-Line Interface**

References

Linux From Scratch Version 12.4-systemd

Overview

This document defines the specifications for BMO **OS**, a lightweight operating system designed to perform core system functions on minimal hardware. It provides an efficient, secure, and user-friendly environment for executing programs, managing files, and controlling devices.

Overall Description

This project aims to develop a minimal, lightweight operating system (Mini OS) for educational purposes and for understanding the fundamentals of operating systems. It is suitable for embedded systems or environments with minimal resource requirements. The OS will focus on providing a basic, bootable, and usable environment.

Product Perspective

Mini OS acts as the **bridge** between hardware and user-level applications. It directly controls the CPU, memory, and input/output devices while offering a minimal API and shell interface. It fits into the broader system as a **core lightweight platform**, not dependent on external operating environments

Functional Requirements

The BMO **OS** is a simplified operating system designed for educational or research purposes.

It provides basic functionalities such as process management, memory management, file handling, device control, graphic user interaction through , and i/o

1. Process Management

Description: Responsible for creating, scheduling, running, and terminating processes. It ensures the CPU is shared efficiently among active programs.

GUI Representation: A “Processes” window showing a table of all processes with columns like *PID*, *Name*, *State* (Running, Ready, Waiting), *CPU time*, and *Priority*.

Buttons: **Run**, **Pause**, **Resume**, **Kill**, and **Details**.

Inputs / Outputs:

- **Input:** User clicks “Run Program” or selects a program file.
- **Output:** New process appears in the table with a live state update.

Internal Data:

- **Process Control Block (PCB):** holds PID, state, registers, memory map, etc.
- **Queues:** Ready Queue, Waiting Queue, Running pointer.

Algorithms:

- *Process Creation*: allocate PCB, initialize memory, set state to Ready.

- *Scheduling*: Round Robin or Priority Scheduling.
- *Context Switching*: save and restore CPU registers.

Error Handling / Testing:

- If process creation fails → pop-up message “Process creation failed: insufficient memory.”
- Test by launching multiple processes and verifying proper state switching.

---

2. Memory Management

Description: Manages allocation and deallocation of memory blocks to processes, avoiding overlap and optimizing usage.

GUI Representation: A “Memory” panel showing a visual **Memory Map** — colored blocks for each process.

Buttons: **Allocate**, **Free**, **Refresh**, **View Details**.

Inputs / Outputs:

- **Input:** User requests memory allocation via GUI.
- **Output:** Updated map or error message “Out of Memory”.

Internal Data:

- Free list or bitmap for memory tracking.
- Allocation table mapping process → memory blocks.

Algorithms:

- *First Fit* or *Best Fit* allocation.
- Optional: simple paging system.

Error Handling / Testing:

- If memory overflow occurs, process is stopped and user is notified.
- Test by running programs until memory is full.

3. Graphical User Interface (GUI)

Description: A graphical interface that allows users to interact with the OS through windows, buttons, and menus instead of text commands.

Main Components:

- **Top Bar:** OS name and “Shutdown” button.
- **Tabs:** Dashboard, Processes, Memory, Files, Logs.
- **Windows:** “Run Program” dialog, “Settings” window.

User Flow Example:

Click **Run Program → Select file → Process created → Appears in Process Tab**.

All panels update dynamically to reflect system changes.

Implementation (for BMO OS):

Can be a simulated GUI (ASCII GUI or framebuffer-based), not full desktop graphics.

Error Handling / Testing:

- If program fails to load → pop-up “File not found.”
- Test that all GUI buttons respond and reflect state changes in real time.

4. File Management

Description: Allows creation, reading, writing, and deletion of small text files. Implements a simple file system for demonstration.

GUI Representation:

A “Files” tab showing a file list. Buttons: **New**, **Open**, **Delete**, **Save**.

A mini text editor for viewing/editing file contents.

Inputs / Outputs:

- **Input:** User creates or opens a file.
- **Output:** File appears in the directory list or editor.

Internal Data:

- File Table: filename → block list, size, owner.

- Storage area simulated as a block array.

Algorithms:

- *Read/Write:* sequential access.
- *Create/Delete:* update file table and blocks.

Error Handling / Testing:

- Show “No Space Left” when storage is full.
- Test by creating and reopening files to confirm data persistence.

5. Command-Line Interface (CLI)

Description: The **Command Line Interface (CLI)** is a **text-based interface** that allows the user to interact with the Mini Operating System by typing commands.

- **built-in command:**
  - help: display available commands
  - ls [list] : list files
  - cd [change directory]: Changes the current working directory
  - run: execute a program or process
  - clear: clear the screen
  - exit: Logs out or shuts down the system
  - date: Displays the current date and time
  - etc.. *(Linux main CLI command)*

Command Execution:

- The CLI shall send valid commands to the kernel or relevant system module for execution, shall display the output or result of each command after execution, shall maintain system stability even if a command fails during execution.

Error Handling:

- The CLI shall display appropriate messages for syntax errors, invalid commands, or incorrect arguments.
- The CLI shall recover gracefully from errors without restarting the terminal.
- The CLI shall return to the prompt after any error.

5. I/O Management

Description: Handles input/output devices like the keyboard, display, and simulated peripherals. Manages how processes communicate with devices.

GUI Representation:

Logs or visual indicators showing keyboard events and device activity.

Inputs / Outputs:

- **Input:** Key presses, mouse clicks, or process I/O requests.
- **Output:** Updated display, messages, or log entries.

Internal Data:

- Simple device drivers (keyboard, display).
- I/O queue for pending requests.

Algorithms:

- Blocking I/O: process waits until input is ready.
- Non-blocking I/O (optional): process continues after request submission.

Error Handling / Testing:

- Invalid device request → error log message.
- Test continuous input without freezing the system.

6. System Clock & Scheduling

Description: The system clock generates interrupts that drive time-based operations like context switching, CPU scheduling, and timers.

GUI Representation:

- A small timer in the **Dashboard**.

- A “Settings” option to adjust the time slice duration.
- CPU usage graphs.

Inputs / Outputs:

- **Input:** User changes time slice.
- **Output:** Updated scheduling behavior.

Internal Data:

- Timer Interrupt routine.
- Scheduler structure with ready queue.

Algorithms:

- **Round Robin:** Each process gets a fixed CPU time.
- **Priority Scheduling (optional):** High-priority processes run first.

Error Handling / Testing:

- Too small time slice → frequent switching (test responsiveness).
- Verify smooth CPU switching among processes.

Non Functional Requirements

1. Performance

- The system should manage **multiple lightweight processes** efficiently without noticeable delay.
- Boot time should not exceed **5 seconds**.

2. Memory Efficiency

- Memory allocation and deallocation should be optimized to reduce waste.
- The system should handle low-memory situations gracefully.

3. Reliability

- The system must remain stable during continuous operation.

- If a process crashes, the OS should **not crash entirely**.
- The OS should recover from minor errors automatically

4. Usability

- The GUI must be **intuitive and user-friendly**.
- Buttons and menus should be labeled clearly.
- The user should be able to execute tasks **without needing technical knowledge**.

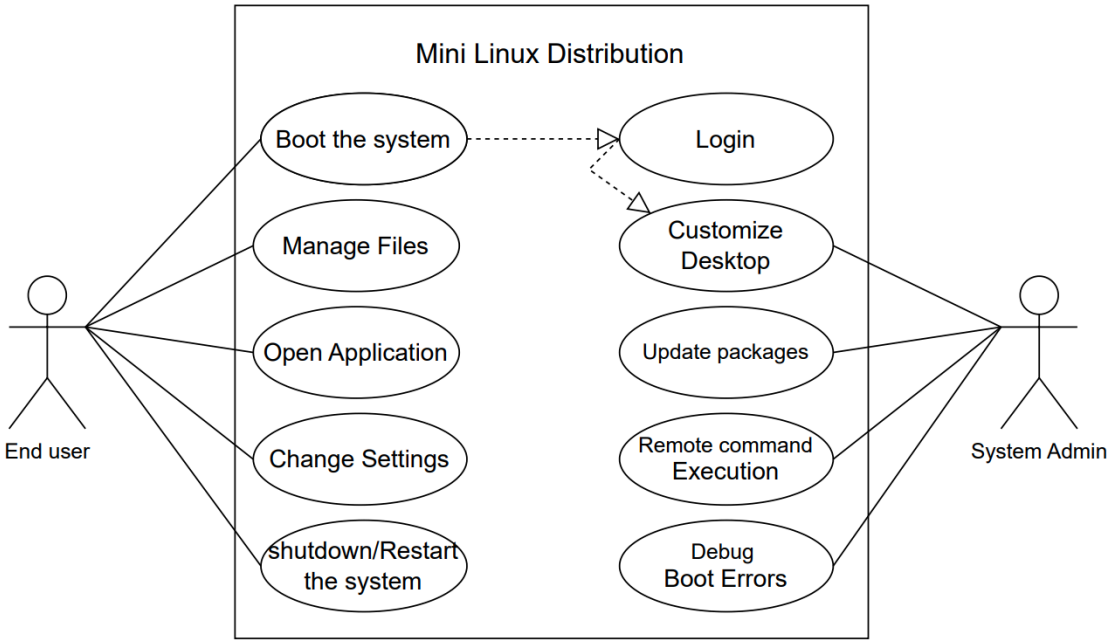
5. Portability

- The OS should run on different environments (e.g., VirtualBox, VMware, or x86 hardware).
- It should be compatible with common compilers and architectures.

6. Maintainability

- The source code should be **modular and well-commented**.
- Developers should be able to **update or fix** features without breaking the system.
- All core modules (GUI, memory, processes) should be **separated logically**.

Basic System Models



System Navigation Map

https://www.figma.com/design/dVsgKYMbLgFfvoFYwngmEQ/Untitled?node-id=2-2&t=EeIF7WTW3ohg4ETf-1

Tools

- **Programming Languages:** C, C++, Bash, Python, Dart
- **Build Systems:** Make, CMake, Meson, Ninja
- **Compiler and Linker:** GCC, Binutils
- **Init System:** Systemd
- **Package Sources:** Linux From Scratch (LFS), Beyond Linux From Scratch (BLFS)

- **Version Control:** Git + GitHub
- **UI Framework:** Flutter (planned)
- **Bootloader:** GRUB2
- **Network Tools:** OpenSSH, IPRoute2, Wget
- **Filesystem Tools:** E2fsprogs, Util-linux
- **Testing Tools:** DejaGNU, GDB
- **Host OS:** Fedora Linux
- **Development Tools:** Vim, VS Code, nano