Design and Implementation of Eagle OS: A 64-bit Educational Operating System Kernel

N. Rohan Gopal
Department of Electronics and Communication Engineering Bangalore, India

***Abstract*—This paper presents the design and implementation of Eagle OS, a custom-built 64-bit operating system kernel devel- oped from scratch using a freestanding environment on Ubuntu 24.04 LTS. The project explores low-level system programming concepts including Multiboot compliance, stack initialization, Global Descriptor Table (GDT) configuration, paging structures, and transition from 32-bit protected mode to 64-bit long mode. The kernel is executed using QEMU virtualization. The objective is to provide a practical understanding of processor architecture and operating system internals.**

***Index Terms*—Operating Systems, Kernel Development, Long Mode, Paging, GDT, Multiboot, x86-64 Architecture**

I. INTRODUCTION

Modern operating systems are built upon complex kernel architectures. Understanding their internal operation re- quires hands-on implementation of fundamental mechanisms including bootloading, segmentation, and paging. Eagle OS is designed as an educational 64-bit kernel to explore these concepts at the hardware interaction level. The implementation emphasizes clarity, correctness of the mode transition sequence, and minimal but verifiable kernel functionality to demonstrate successful long mode activation.

II. BOOT PROCESS ARCHITECTURE

The boot process of Eagle OS is designed to illustrate the complete transition from firmware execution to a fully operational 64-bit kernel environment. The sequence follows a layered approach where each stage prepares the processor and memory subsystem for the next:

A. Firmware Initialization (BIOS/UEFI)

At system reset the firmware (BIOS or UEFI) performs hardware initialization, configures basic device state, and exe- cutes POST (Power-On Self-Test). UEFI systems may provide richer services (e.g., EFI runtime services and a GUID Partition Table) while legacy BIOS provides a simpler bootstrap. The firmware locates a bootloader on the selected device and transfers control to it. During this stage the firmware also provides a memory map and device enumeration that the bootloader can pass to the kernel.

B. GRUB Bootloader and Multiboot Handoff

Eagle OS uses GRUB as the primary bootloader. GRUB reads the Multiboot header embedded in the kernel image and, if the header is valid, loads the kernel into memory and prepares a Multiboot information structure. This structure contains the memory map, module addresses, and boot device information. GRUB sets up a minimal execution environment and transfers control to the kernel entry point in 32- bit protected mode (or a GRUB-provided entry state). The Multiboot header (magic number, flags, checksum) ensures GRUB recognizes the image and provides the required handoff information.

C. Staging in 32-bit Protected Mode

Although the target execution mode is 64-bit long mode, the canonical transition path uses 32-bit protected mode as an intermediate stage. Protected mode enables the use of the Global Descriptor Table (GDT) and allows the kernel to set up paging structures that are required for long mode. In this stage the kernel:

- Establishes a temporary GDT with code and data descriptors suitable for the far jump into long

mode.
- Allocates and initializes a stack in the BSS area.
- Constructs the initial page tables (PML4, PDPT, PD, PT) for identity mapping of low memory.

D. Long Mode Activation

Transition to long mode requires enabling several processor features in a precise order (detailed in Section VI). After paging and control registers are configured, a far jump into the 64-bit code segment is performed. Once in long mode, the kernel switches to a native 64-bit C runtime and continues initialization.

E. Kernel Entry and Verification

After the mode transition the kernel entry point executes in 64-bit mode. A minimal verification step writes a short string to the VGA text buffer at physical address 0xB8000 to confirm correct execution and memory mapping. This simple test validates that the CPU is in long mode and that identity mappings for the VGA region are correct.

III. MULTIBOOT HEADER IMPLEMENTATION

The kernel binary includes a Multiboot header containing:
- Magic Number: 0x1BADB002
- Flags: indicating required features and alignment
- Checksum: to validate header integrity

These fields allow GRUB to recognize and correctly load the kernel image and to pass the Multiboot information structure to the kernel.

IV. STACK INITIALIZATION

A 16 KB stack is allocated in the BSS section prior to enabling long mode. The stack pointer (RSP) is initialized to the top of this region. The stack is aligned to a 16-byte boundary to satisfy the x86-64 ABI requirements for later C code execution. During the protected-mode staging code the stack is used for temporary function calls and for the far jump sequence; after entering long mode the same stack region is used by the 64-bit kernel.

V. GLOBAL DESCRIPTOR TABLE (GDT)

The GDT provides segment descriptors required before entering long mode. Eagle OS defines a minimal GDT con- taining:
- Null descriptor
- 64-bit code segment descriptor (with appropriate flags for long mode)
- 64-bit data segment descriptor

The GDT is loaded using the lgdt instruction. A far jump is performed to load the new code segment selector and to ensure the CPU uses the intended descriptor base for subsequent operations. Although segmentation is largely unused in long mode, a correct GDT is required during the transition.

I. PAGING AND LONG MODE ACTIVATION

Long mode activation depends critically on correct paging setup and control register configuration. The sequence implemented in Eagle OS is:

A. Paging Structures

Eagle OS constructs the canonical 4-level page table hierarchy required for x86-64:
- PML4 (Page Map Level 4): top-level table
- PDPT (Page Directory Pointer Table): second level
- PD (Page Directory): third level
- PT (Page Table): fourth level (used when 4 KB pages are required)

For the initial bootstrap the kernel uses 2 MB pages (via PD entries with the PS bit) to identity-map the first 2 MB of physical memory. This reduces the number of page table entries required and simplifies the early mapping of the VGA buffer and kernel code.

B. Identity Mapping

Identity mapping maps virtual addresses to the same physical addresses for the low memory region. This is essential during the transition because the CPU fetches instructions from physical addresses that must remain valid after paging is enabled. Eagle OS identity-maps:
- The first 2 MB of physical memory (kernel code, data, and VGA buffer)
- Any additional regions required by the bootloader handoff (e.g., Multiboot modules)

C. Control Register and MSR Configuration

The following register operations are performed in the specified order:
1) Load the PML4 base physical address into CR3.
2) Enable Physical Address Extension (PAE) by setting the PAE bit in CR4.
3) Set the Long Mode Enable (LME) bit in the

Extended
Feature Enable Register (EFER) MSR.
4) Enable paging by setting the PG bit in CR0.
After these steps a far jump into the 64-bit code segment is executed to begin 64-bit execution. Any misconfiguration (incorrect table alignment, wrong CR3, missing PAE) can cause a triple fault and system reset.

VII. KERNEL EXECUTION

After entering long mode, control transfers to a 64-bit C function. A minimal kernel entry is shown below:
Listing 1: Minimal kernel entry used to verify long mode execution.

```
void kernel_main(void) {
    volatile char* video = (volatile char*) 0xB800
    const char* str = "Eagle_OS ■—■64-Bit ■Long ■Mod
    for (int i = 0; str[i] != 0; i++) {
        video[i * 2] = str[i];
        video[i * 2 + 1] = 0x07;
    }
    while (1) ;
}
```

This routine writes directly to the VGA text buffer to provide a visible confirmation that the kernel is executing in 64-bit mode and that the identity mapping for the VGA region is correct.

VIII. DEVELOPMENT ENVIRONMENT AND TOOLCHAIN

The Eagle OS toolchain and environment include:
- NASM for assembly sources (boot and early setup)
- GCC configured for freestanding compilation of 64-bit code
- LD with a custom linker script to place sections and define the entry point
- GRUB as the Multiboot-compliant bootloader
- QEMU for virtualization and iterative testing

The build process uses a Makefile that assembles the boot sector, links the kernel image, and produces a Multiboot- compliant binary that GRUB can load. QEMU is used with serial and VGA output enabled to observe kernel behavior and to debug early-stage faults.

IX. EVALUATION AND VERIFICATION

To verify correctness, the following checks were performed:
- Boot validation: GRUB successfully loads the kernel and passes the Multiboot information structure.
- Mode transition: The CPU transitions from protected mode to long mode without triple faults.
- Memory mapping: The VGA buffer and kernel code are readable and writable after paging is enabled.
- Stability: The kernel remains in a stable loop after printing the verification string.

Future work will add automated unit tests for page table correctness and a small serial console to capture early kernel logs.

X. CONCLUSION

Eagle OS demonstrates a minimal but complete path to a 64-bit kernel: Multiboot-compliant loading, protected-mode staging, correct GDT and paging setup, and successful long mode activation. The implementation provides a clear educational platform for exploring CPU architecture, memory management, and low-level OS design.

XI. FUTURE WORK

Planned extensions include:
- Implementing an Interrupt Descriptor Table (IDT) and basic interrupt handling
- Writing device drivers for serial and block devices
- Implementing a memory manager with dynamic allocation
- Adding a preemptive scheduler and basic process abstraction
- Integrating a simple filesystem and module loader

XII. ACKNOWLEDGMENT

The author thanks the OSDev community and the Intel Soft- ware Developer's Manual for reference material and guidance.

REFERENCES

[1] Intel Corporation, Intel® 64 and IA-32 Architectures Software Developer's Manual. Intel Press, 2023.
[2] OS Dev Wiki, "Entering long mode." [Online]. Available: https://wiki.osdev.org/Entering_Long_Mode
[3] A. S. Tanenbaum and H. Bos, Modern Operating Systems, 4th ed. Boston, MA, USA: Pearson, 2015.