Design and Implementation of EagleOS: A 64-bit Operating System Kernel

N. Rohan Gopal[1], Arnold A[2]
[1]Department of Electronics and Communication Engineering Bangalore, India
[2]Department of Computer Science Engineering Bangalore, India

***Abstract*—This paper presents the design and implementation of EagleOS, a custom-built 64-bit operating system kernel devel- oped from scratch using a freestanding environment on Ubuntu Linux. The project explores low-level system programming concepts including Multiboot compliance, stack initialization, Global Descriptor Table configuration, paging structures, and the 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, Multiboot, x86-64 Architecture**

I. INTRODUCTION

Modern operating systems are built upon complex ker- nel architectures. Understanding their internal operation re- quires hands-on implementation of fundamental mechanisms including bootloading, segmentation, and paging. EagleOS is designed as an educational 64-bit kernel to explore these concepts at the hardware interaction level. The implemen- tation 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 EagleOS illustrates the complete tran- sition from firmware execution to a fully operational 64-bit kernel environment.

A. Firmware Initialization

At system reset the firmware (BIOS or UEFI) performs hardware initialization, configures basic device state, and executes POST (Power-On Self-Test). UEFI systems provide richer services such as EFI runtime services and GUID par- tition tables, while legacy BIOS provides a simpler bootstrap environment.

The firmware locates a bootloader on the selected device and transfers control to it. During this stage the firmware may also provide a memory map and device enumeration.

B. GRUB Bootloader and Multiboot Handoff

EagleOS uses GRUB as the primary bootloader. GRUB reads the Multiboot header embedded in the kernel image and loads the kernel into memory.

The Multiboot information structure passed to the kernel contains:

- Memory map
- Boot device information
- Module addresses

The Multiboot header consists of:

- Magic Number: 0x1BADB002
- Flags
- Checksum

III. PROTECTED MODE STAGING

Although the target execution mode is 64-bit long mode, the canonical transition path uses 32-bit protected mode as an intermediate stage.

During this stage the kernel:

- Establishes a temporary Global Descriptor Table
- Allocates and initializes a stack
- Constructs initial paging structures

IV. STACK INITIALIZATION

A 16 KB stack is allocated in the BSS section before

entering long mode. The stack pointer (RSP) is initialized to the top of this region and aligned to a 16-byte boundary to satisfy the x86-64 ABI requirements.

V. GLOBAL DESCRIPTOR TABLE

The GDT provides segment descriptors required during the transition to long mode.
EagleOS defines a minimal GDT consisting of:

- Null descriptor
- 64-bit code segment descriptor
- 64-bit data segment descriptor

```
gdt_code: dq 0x00AF9A000000FFFF
gdt_data: dq 0x00AF92000000FFFF
```

The GDT is loaded using the lgdt instruction followed by a far jump.

VI. PAGING AND LONG MODE ACTIVATION

Long mode activation requires correct paging configuration.

A. Paging Structures

The x86-64 architecture uses a four-level paging hierarchy:

- PML4
- PDPT
- Page Directory
- Page Table

B. Identity Mapping

Identity mapping maps virtual addresses to identical physi- cal addresses. EagleOS identity maps the first 2MB of physical memory to ensure that kernel code and VGA memory remain accessible during the transition.

```
pml4_table: resq 512

pdpt_table: resq 512
```

VII. CONTROL REGISTER CONFIGURATION

The following sequence enables long mode:

```
mov eax, cr4
or eax, 1 << 5
mov cr4, eax

mov ecx, 0xC0000080
rdmsr
or eax, 1 << 8
wrmsr

mov eax, cr0
or eax, 1 << 31
mov cr0, eax
```

After these steps a far jump transfers control to the 64-bit kernel.

VIII. KERNEL EXECUTION

The kernel verifies correct execution by writing text to VGA memory.

```
void kernel_main() {

volatile char *video = (volatile char*)0xB8000;
const char *str = "Eagle OS - 64 Bit Long Mode";

for(int i=0; str[i]!=0; i++){
video[i*2] = str[i];
video[i*2+1] = 0x07;
}

while(1);
}
```

IX. DEVELOPMENT ENVIRONMENT

The EagleOS development environment includes:

- Ubuntu Linux
- NASM assembler
- GCC compiler
- GNU linker
- GRUB bootloader
- QEMU emulator

X. BUILD PROCESS

A. Assembly

```
nasm -f elf32 src/boot.asm -o boot.o
nasm -f elf32 src/gdt.asm -o gdt.o
nasm -f elf32 src/longmode.asm -o longmode.o
nasm -f elf32 src/paging.asm -o paging.o
```

B. Compile Kernel

```
gcc -c src/kernel.c -o kernel.o -m32 -ffreestanding
```

C. Link

```
ld -m elf_i386 -T linker.ld -o kernel.bin
boot.o gdt.o longmode.o paging.o kernel.o
```

XI. CREATING BOOTABLE ISO

```
mkdir -p iso/boot/grub
cp kernel.bin iso/boot/
```

GRUB configuration:

```
menuentry "EagleOS" {
multiboot /boot/kernel.bin
boot
}
```

Generate ISO:

```
grub-mkrescue -o EagleOS.iso iso/
```

RUNNING THE SYSTEM

```
qemu-system-x86_64 EagleOS.iso
```

XII. EVALUATION AND VERIFICATION

The system was verified using the following tests:

- Successful GRUB kernel loading
- Correct transition to long mode
- Valid memory mapping
- VGA output verification

XIII. FUTURE WORK

Planned improvements include:

- Interrupt Descriptor Table implementation
- Device drivers
- Memory management system
- Preemptive scheduler
- Filesystem support

XIV. CONCLUSION

EagleOS demonstrates the complete process required to create a minimal 64-bit operating system kernel. The project provides a clear platform for understanding processor archi- tecture, memory management, and low-level operating system design.

XV. ACKNOWLEDGMENT

The authors thank the OSDev community and Intel Software Developer's Manual for reference material and guidance.

REFERENCES

[1] Intel Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual, 2023.
[2] OSDev Wiki, https://wiki.osdev.org/Entering Long Mode
[3] A. S. Tanenbaum and H. Bos, Modern Operating Systems, 4th ed., Pearson, 2015.