# 操作系统内核编写项目报告

## 一、项目目的

本项目的目标是从零编写一个简化的操作系统内核。通过构建交叉编译环境、使用 GRUB 完成引导加载、利用链接脚本布置内核内存布局，并在内核中实现若干核心功能，使内核能够在 QEMU/Bochs 模拟器中成功运行。

主要功能包括：

1. 内核入口与启动流程：通过 GRUB 加载内核，利用 boot.S 建立执行环境，并跳转至内核主函数 main。
2. 内存管理：基于 GRUB 提供的内存信息，完成内存初始化与动态分配接口，实现内核级内存管理。
3. 中断与异常处理：构建全局描述符表（GDT）和中断描述符表（IDT），并完成时钟中断的初始化与处理中断机制。
4. 系统调用机制：基于 syscall 指令，建立用户态与内核态的交互通道，支持基础系统调用功能。
5. 多线程支持：设计并实现线程管理模块，能够创建并调度多个线程，实现内核级并发执行。

## 二、实验环境

操作系统：Ubuntu 20.04/22.04

编译工具链：x86\_64-elf-gcc、ld、nasm

构建工具：xmake

引导加载器：GRUB2 (Multiboot2)

模拟器：QEMU、Bochs

## 三、项目结构

实验工程的主要文件和目录如下：

src/：核心源码目录

boot/boot.S：启动汇编代码，提供 \_start 入口，调用 main

kernel/\*.c：内核核心模块（内存管理、进程/线程、系统调用等）

user/\*.c：用户态测试程序

lib/\*.c：辅助函数（如 printf、string 库）

build/：编译生成目录，保存 kernel.elf 和 kernel.iso

grub/grub.cfg：GRUB 配置文件，指定加载内核

main.c：内核主函数，负责初始化各子系统并创建测试线程

linker.ld：链接脚本，定义内核加载地址和内存布局

bochsrc：Bochs 模拟器配置文件

xmake.lua：构建脚本，自动完成编译、链接和 ISO 镜像生成

## 四、关键代码与原理

### 4.1 内核主函数（main.c）

内核的入口函数 main 完成以下工作：

1. 内存初始化：调用 memory\_init(magic, addr)，接收 GRUB 提供的启动信息，建立内存管理机制。
2. 终端与串口初始化：terminal\_init() 与 serial\_init() 用于屏幕输出和串口调试。
3. 中断与 GDT 初始化：设置中断描述符表和全局描述符表，保证内核能处理异常和中断。
4. 堆内存与时钟初始化：kheap\_init() 提供动态内存分配；timer\_init() 设置时钟中断。
5. 系统调用与线程初始化：注册系统调用表，建立线程管理机制。
6. 测试线程执行：通过 process\_execute(thread\_a, "proc\_a") 和 process\_execute(thread\_b, "proc\_b") 创建并调度两个线程，测试系统调用机制。

测试机制： thread\_a 调用 syscall，将 RAX=1； thread\_b 调用 syscall，将 RAX=0。 说明内核已正确支持系统调用号传递与调度。

### 4.2 链接脚本（linker.ld）

该文件定义了内核的段布局：

GRUB Multiboot2 Header：放置在低地址 1M 处，保证 GRUB 能识别并加载内核。

.text / .rodata / .data / .bss：分别存放代码、只读数据、已初始化数据、未初始化数据。

高地址映射：通过 0xFFFFFFFF80000000 将内核映射到内核空间，隔离用户态地址。

符号导出：如 \_kernel\_phys\_end，用于确定内核大小和内存管理边界。

### 4.3 Bochs 配置（bochsrc）

设置内存 megs: 32

选择 boot: cdrom，从 build/kernel.iso 启动

开启 magic\_break，支持 BOCHS\_BREAK 指令触发断点

串口 com1 输出到文件 com1\_log，便于调试

### 4.4 Xmake 构建脚本（xmake.lua）

定义 x86\_64-elf 交叉编译工具链

添加 src/boot、src/kernel、src/user 源文件

设置编译参数：-ffreestanding -nostdlib -mcmodel=kernel，保证无标准库依赖

链接时使用 linker.ld，生成 system.map

构建完成后：

1. 拷贝 kernel.elf 到 build/iso/boot/
2. 添加 grub.cfg，生成 kernel.iso

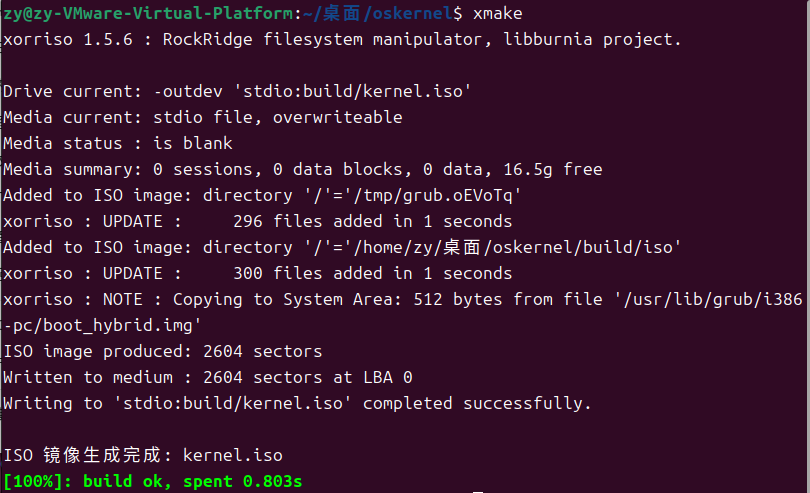
提供 on\_run 命令，可直接调用 qemu-system-x86\_64 -cdrom build/kernel.iso 启动内核

## 运行步骤和结果

### 5.1 内核的编译与运行

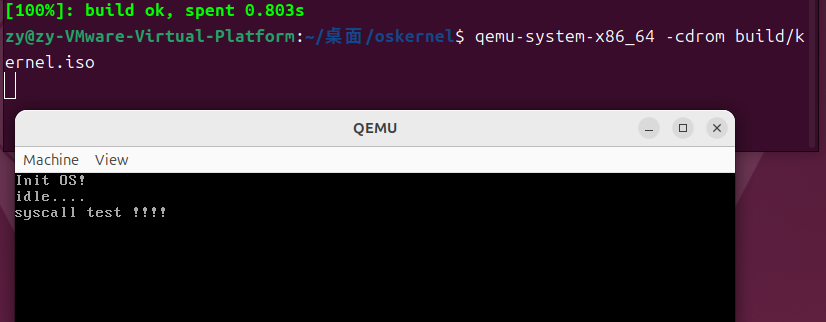
在 Ubuntu 环境下，进入工程根目录执行以下命令：

Xmake



启动内核

执行：qemu-system-x86\_64 -cdrom build/kernel.iso



GRUB 正确加载了内核，启动完成，并进入执行流程

### 5.2 时钟中断验证

修改 time.c，在 timer\_interrupt\_handler 里加入了一个打印输出语句：

void timer\_interrupt\_handler(uint8\_t vec\_nr){

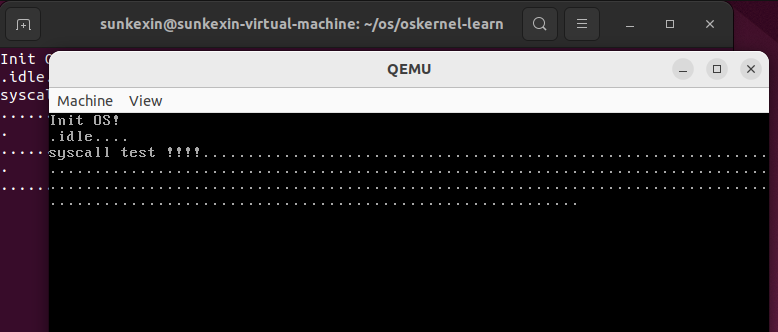
ASSERT(vec\_nr == 0x20);

pic\_send\_eoi(vec\_nr);

putchar('.'); // 每次时钟中断打印一个点

thread\_handler();

}

在每次时钟中断发生时，屏幕上打印一个 .  
运行内核时，看到屏幕不断出现 ....，说明时钟中断在周期性触发并输出点

### 5.3 验证进程调度

为了验证内核线程调度机制，在 main.c 中编写两个内核态线程：

// 内核态线程：不断打印 'A'

void thread\_a(void\* arg){

while (1){

putchar('A');

for (volatile int i = 0; i < 100000; ++i) { }

}

}

// 内核态线程：不断打印 'B'

void thread\_b(void\* arg){

while (1){

putchar('B');

for (volatile int i = 0; i < 100000; ++i) { }

}

}

在 main() 函数初始化完成后，调用 thread\_start() 创建两个内核线程：

thread\_start("thread\_a", 31, thread\_a, NULL);

thread\_start("thread\_b", 31, thread\_b, NULL);

然后开启中断：

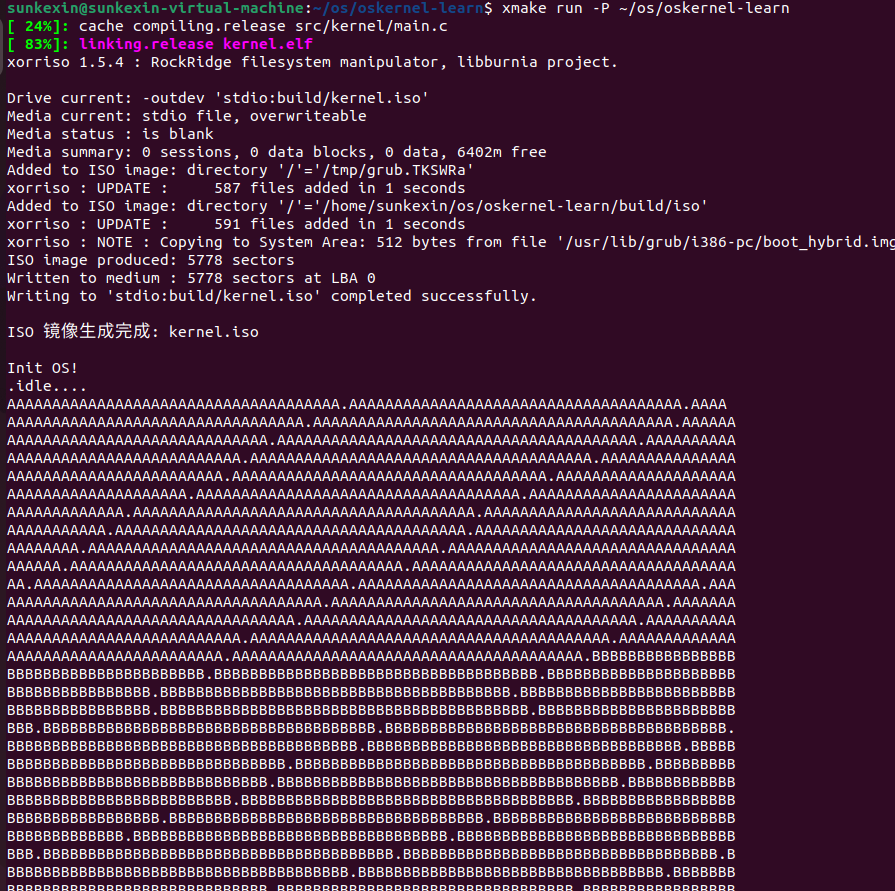
intr\_set\_status(true); // 允许中断

while (1) {

hlt(); // 进入低功耗等待中断

}

xmake运行后得到



创建了两个普通线程，每个线程循环打印不同的字符， A 和 B，能够交替运行；

验证时钟中断触发后，调度器能够切换线程。运行结果说明时钟中断与线程调度都能够正常运行。内核支持多线程并发执行的基本功能。

### 5.4 内存管理测试

为了验证内核堆内存分配与回收机制，在 main.c 中编写了 mem\_test() 函数：

// 内核内存管理测试函数

void mem\_test() {

printk("==== Kernel Memory Management Test Start ====\n");

void\* p1 = kmalloc(128);

void\* p2 = kmalloc(256);

void\* p3 = kmalloc(512);

printk("Allocated kernel memory addresses:\n");

printk("p1=0x%x\n", (uint64\_t)p1);

printk("p2=0x%x\n", (uint64\_t)p2);

printk("p3=0x%x\n", (uint64\_t)p3);

kfree(p1);

kfree(p2);

kfree(p3);

printk("Memory freed!\n");

void\* p4 = kmalloc(128);

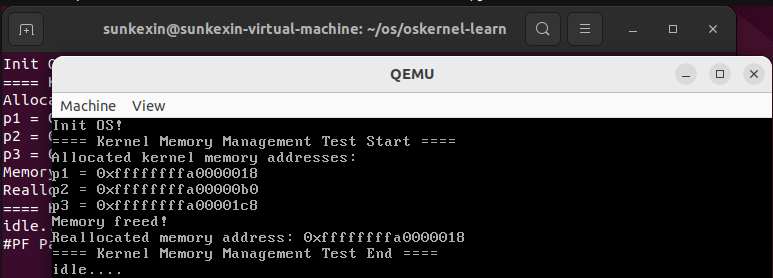
printk("Reallocated memory address: 0x%x\n", (uint64\_t)p4);

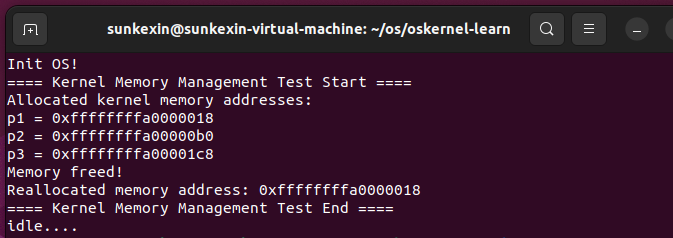
printk("==== Kernel Memory Management Test End ====\n");

}

在 main() 初始化阶段调用 mem\_test()，完成对 kmalloc/kfree 的测试

QEMU 运行时输出：





kmalloc/kfree 在虚拟地址上工作正常，地址落在内核堆区（高端内核空间）。

释放后再次分配的地址可能与之前释放的地址相同，说明复用正常

内核正常访问映射的虚拟地址，页表映射正确

## 六、项目总结