MIT6.828 Lab4: Preemptive Multitasking

Zhuofan Zhang

April 2020

Contents

Chapter 1

Multiprocessor Support and Cooperative Multitasking

1.1 Multiprocessor Support

本次 Lab 的内容是对 JOS 进行补充以提供多处理器支持，并实现任务调度功能。

JOS 实现的多核支持属于对称多核支持 *(Symmertric Multiprocessing, SMP)*，即所有处理器对资源（内存、IO 总线等）的访问是平等的。SMP 的概念是在系统初始化完成后建立的，在 Boot 阶段，仍然需要选择一个 CPU 作为启动处理器 *(Bootstrap Processor, BSP)*，用来初始化资源并启动 OS，最后启动其他的 CPU（称为应用处理器 *(Application Processor, AP)*）。对 BSP 的选择是由 BIOS 完成的，在当前实验阶段，我们相当于完成了 BSP 的启动内容。

在 SMP 体系中，每个 CPU 都拥有一个称为 LAPIC(local Advanced-Programmable Interrupt Controller) 的可编程中断单元，用于在系统中传递中断信息，并提供 CPU 的唯一识别信息。因此，与多个 CPU 的通讯依赖于 LAPIC 单元。

在本次实验中我们使用了 LAPIC 的如下功能：

- 代码可以利用 APIC-id 确认自己运行于哪一个 cpu 上（见 cpunum() 的实现）
- 利用 BSP 来启动其他的 AP（见 lapic_startap() 的实现）
- 使用 LAPIC 中的时钟中断来实现抢断式调度（PartC，apic_init()）

CPU 对 LAPIC 单元的访问使用的是一段映射到 PAS 特定位置的区域：memory-mapped I/O (MMIO)。MMIO 是一段硬连接到部分 IO 设备寄存器的物理地址区域，常用于访问设备的寄存器。JOS 的 VAS 在 MMIOBASE 处留有 4MB 的空间用于映射这段物理地址区域。

Exercise 1

Implement mmio_map_region in kern/pmap.c. To see how this is used, look at the beginning of lapic_init in kern/lapic.c. You'll have to do the next exercise, too, before the tests for mmio_map_region will run.

这一个 Exercise 要求我们实现 mmio_map_region()。我们查看该函数的注释可知，它就是用来实现上文提到的 MMIO 在 VAS 中映射的函数。JOS 的 VAS 中预留了 [MMIOBASE, MMIOLIM) 可供使用。根据提示使用 boot_map_region() 并补上禁用缓存的 PWT-flag，实现函数如下：

```
void *
mmio_map_region(physaddr_t pa, size_t size)
{
    static uintptr_t base = MMIOBASE;

    size = ROUNDUP(size, PGSIZE);
    if(base + size > MMIOLIM)
        panic("mmio_map_region: MMIOLIM overflow.\n");
    boot_map_region(kern_pgdir, base, size, pa, PTE_PCD | PTE_PWT);
    base += size;
    return (void *)(base - size);

}
```

1.2 Application Processor Bootstrap

在启动 APs 前，先启动的 BSP 必须能够获取 APs 的信息，如 CPU 数量、APIC IDs 等。这些信息被保留在 BIOS 中的 MP config table 中，kern/mpconfig.c/mp_init() 将其读出。

kern/init.c/boot_aps() 是整个流程的核心函数，它将 AP 的入口程序加载到内存，并发送启动信号。AP 的入口程序被加载到任意可用的低位 640k 地址（JOS 将其加载到 MPENTRY_PADDR），它与 BSP 的入口程序有一定的区别。

boot_aps() 函数通过给每个 AP 的 LAPIC 发送 STARTUP 信号的方式唤醒它们，同时设置它们的 entry 位置，即为 AP 准备的入口程序（位于 MPENTRY_PADDR），执行完入口程序后 AP 将运行 mp_main()；与此同时，BSP 唤醒每一个 AP 时等待它的 CPU 状态被设置为 CPU_STARTED——即该 AP 成功启动后，再开始唤醒下一个 AP。

Exercise 2

Read boot_aps() and mp_main() in kern/init.c, and the assembly code in kern/mpentry.S. Make sure you understand the control flow transfer during the bootstrap of APs. Then modify your implementation of page_init() in kern/pmap.c to avoid adding the page at MPENTRY_PADDR to the free list, so that we can safely copy and run AP bootstrap code at that physical address. Your code should pass the updated check_page_free_list() test (but might fail the updated check_kern_pgdir() test, which we will fix soon).

这一个 Exercise 的编码任务比较简单：修改我们在 Lab2 实现的 page_init 函数，将现在存放有 AP 初始代码的那个物理页从空闲列表中拿出去：

```
void
page_init(void)
{
    // LAB 4:
    // Change your code to mark the physical page at MPENTRY_PADDR
    // as in use

    ...

    // 2) Base-memory
    size_t i;
    for (i = 1; i < npages_basemem; i++) {
        // Add for Lab4
        if(i * PGSIZE == MPENTRY_PADDR)
        {
            pages[i].pp_ref = 1;
            continue;
        }

        pages[i].pp_ref = 0;
        pages[i].pp_link = page_free_list;
        page_free_list = &pages[i];
    }

    ...

}
```

代码之外的任务是要求我们阅读 boot_aps() 及 mp_main() 源码以及 APs 的 entry 汇编代码，捋清执行流。boot_aps() 做的事情就是将 mpentry.S 搬运到前文提到的 MPENTRY_PADDR 物理页上，再循环唤醒 APs（使用 lapic_startaps()）；APs 被唤醒后从 mpentry.S 开始执行，再跳转到 mp_main() 上。

Question

Compare kern/mpentry.S side by side with boot/boot.S. Bearing in mind that kern/mpentry.S is compiled and linked to run above KERNBASE just like everything else in the kernel, what is the purpose of macro MPBOOTPHYS? Why is it necessary in kern/mpentry.S but not in boot/-boot.S? In other words, what could go wrong if it were omitted in kern/mpentry.S? Hint: recall the differences between the link address and the load address that we have discussed in Lab 1.

AP 的入口程序与 BSP 的入口程序差异主要表现在：

- 使用地址时需要使用 MPBOOTPHYS-macro 进行一步转换；
- 无需进行 A20 地址线的使能操作；
- 直接在入口处打开了分页功能，并使用已经被 BSP 设置好的内核页表目录

使用 MPBOOTPHYS 是因为 mpentry.S 被链接到了内核的高地址处（above KERNBASE）, 需要进行转换。