小结

通过一个月的课程实验，我收获了许多关于操作系统特别是关于ucore的知识。首先，作为一名大一的学生，我只掌握了c语言这一门编程语言，初次接触c这个编程语言后，我对编程产生了浓厚的兴趣，对于未学习过操作系统的我来说，选择ucore作为选题无非对于现在的我来说是一门不小的挑战。在确定选题之后，我便开始在慕课上学习有关操作系统的基本原理以及关于ucore的相关知识点。

在慕课的学习过程中，处理机的调度和死锁以及内存管理这两个章节使我映像很深刻。具体的生活中的例子如银行家算法--避免死锁。死锁的产生是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，他们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程成为死锁进程。但由于资源的占用是互斥的，当某个进程提出申请资源后，使得有关进程在无外力协助下，永远分配不到必需的资源而无法继续运行，这就产生了一种特殊现象死锁。我觉得操作系统所讲的死锁就好像两个人过独木桥的现象。原理即为共享资源。为提高系统资源的利用率，避免死锁并不严格限制死锁必要条件的存在，而是在资源的动态分配过程中，使用某种方法去防止系统进入不安全的状态，从而避免死锁的最终出现。然而，最有代表性的避免死锁的算法，是dijkstra的银行家算法。在该方法中把系统的状态分为安全状态和不安全状态，只要能使系统始终都处于安全状态，便可以避免发生死锁。银行家算法的基本思想是分配资源之前，判断系统是否是安

全的；若是安全的，才会进行分配。

从向老师第一讲的图灵机开始，我便对操作系统产生了浓厚的兴趣，最终决定了 ucore 的实验课题，在 ucore 的 lab 实验中，我完成了 lab1 和 lab2 两个实验。起初，刚拿到实验指导书，我就和看天书一样，对很多的专有名词和代码都一窍不通。在搭建完虚拟机等环境后，我便去 csdn 上查找有关操作系统的文献以及实验报告，对于看不懂的代码用电脑记录下来，用相关的查找工具库去查找相应的代码的含义以及其所构成的语法。比如最简单恰当的例子就是，lab1 中的练习 1-理解通过 make 生成执行文件的过程。因为是第一次做实验，所以印象肯定会很深刻。通过查阅了解到，ld 通过编译将目标文件转化成了一个执行程序——bootblock。make 命令执行时，需要一个 makefile（或 Makefile）文件，以告诉 make 命令需要怎么样的去编译和链接程序。然后就试着将代码凭着自己的理解去敲入虚拟机终端中，并达到了预期的效果，我发现，只有敢于去尝试，才能在无数次失败的代码中获得成功。这便是我第一次尝试 ucore 的经历，也是映像最为深刻的经历。Lab1 练习 2 的利用 gdb 调试 bootloader 实验过程中，也是初次练习了虚拟环境 qemu。通过学习了解到了 gdb 的地址断点。我们在 gdb 命令行中，可以使用 b*加地址在指定内存地址设置断点，当 qemu 中的 cpu 执行到指定地址时，便会将控制权交给 gdb，这对初学的我来说是一项新的收获。在 lab1 练习 3 分析 bootloader 进入保护模式的过程中，我学会了如何打开 A20、如何初始化 GDT 表（把 gdt 表的起始位置和界限装入 GDTR 寄存器中）以及如何进入保护模

式。在 lab1 的练习 4 中，通过实验指导书分析 readsect 函数和 bootmain 函数了解到，readseg 简单包装了 readsect，可以从设备读取任意长度的内容。从读取 ELF 头部首先从硬盘中将 bin/kernel 文件的第一页内容加载到内存地址为 0x10000 的位置，目的是读取 kernel 文件的 ELF Header 信息。

2.校验 ELF Header 的 e_magic 字段，以确保这是一个 ELF 文件

3.读取 ELF Header 的 e_phoff 字段，得到 Program Header 表的起始地址；读取 ELF Header 的 e_phnum 字段，得到 Program Header 表的元素数目。

4.遍历 Program Header 表中的每个元素，得到每个 Segment 在文件中的偏移、要加载到内存中的位置（虚拟地址）及 Segment 的长度等信息，并通过磁盘 I/O 进行加载

5. 加载完毕，通过 ELF Header 的 e_entry 得到内核的入口地址，并跳转到该地址开始执行内核代码

充分理解了 bootloader 加载 ELF 格式的 os 的流程。

在 lab1 的练习 5 实现函数调用堆栈跟踪函数中我们需要在 lab1 中完成 kdebug.c 中函数 print_stackframe 的实现，可以通过函数 print_stackframe 来跟踪函数调用堆栈中记录的返回地址。在如果能够正确实现此函数，可 lab1 中执行 “make qemu”后，在 qemu 模拟器中得到输出，查阅文献了解到，

一般而言，ss:ebp+4 处为返回地址，ss:ebp+8 处为第一个参数值（最后一个入栈的参数值，此处假设其占用 4 字节内存），ss:ebp-4

处为第一个局部变量，ss:ebp 处为上一层 ebp 值。由于 ebp 中的地址处总是“上一层函数调用时的 ebp 值”，而在每一层函数调用中，都能通过当时的 ebp 值“向上（栈底方向）”能获取返回地址、参数值，“向下（栈顶方向）”能获取函数局部变量值。如此形成递归，直至到达栈底。在 lab1 练习 6，完善中断初始化和处理中，中断描述符表中，一个表项占 8 字节，其中 015 位和 4863 为分别为 offset 的低 16 位和高 16 位，16~31 位为段选择子。通过段选择子获得段基址，加上段内偏移量即可获得到中断处理代码的路口。在整个 lab1 的学习过程中，我了解到的知识总结如下：计算机原理：CPU 的编址与寻址: 基于分段机制的内存管理、CPU 的中断机制。Bootloader 软件：编译运行 bootloader 的过程、调试 bootloader 的方法、PC 启动 bootloader 的过程、ELF 执行文件的格式和加载。ucore OS 软件：编译运行 ucore OS 的过程、ucore OS 的启动过程、调试 ucore OS 的方法、函数调用关系：在汇编级了解函数调用栈的结构和处理过程、中断管理：与软件相关的中断处理。

在 lab2 练习 1，实现 first-fit 连续物理内存分配算法中，在实现 first fit 内存分配算法的回收函数时，要考虑地址连续的空闲块之间的合并操作。这就要涉及修改函数。在查看和理解 default_pmm.c 中的注释后，便开始对函数进行修改，default_init_memmap 函数：对最初的一整块未被占用的物理内存空间中的每一页所对应的 page 结构（状态）进行初始化。相邻物理页对应的 page 结构在内存上也是相邻的，因

此可以直接通过第一个空闲物理页对应的 page 结构加上一个偏移量的方式，来访问所有的空闲的物理页的 page 结构。具体初始化方式：遍历所有空闲物理页的 page 结构，将 page 结构中描述空闲块数目的变量置零（故该成员变量只有在整个空闲块的第一个 page 中才有意义），然后清空这些物理页的引用计数，再通过设置 flags 的位的方式将其标记为空闲 。再对空闲块的第一个页的 page 结构进行初始化，具体实现为，将其表示空闲块大小的成员变量设置为参数环路的空闲块大小（单位为页），然后更新存储所有空闲页数量的全局变量，再将这个空闲块插入到空闲内存块链表中（只需将第一个 pagepage_link 插入即可）

default_alloc_pages：分配指定页数的连续空闲物理空间，并将第一页的 page 结构的指针作为结果返回。具体实现方式：

对参数进行合法性检查，并查询总的空闲物理页数目是否足够进行分配。如果不足够进行分配，直接返回 NULL，分配失败。

从头开始遍历保存空闲物理内存块的链表（按照物理地址从小到大的顺序），如果找到一个连续内存块的大小满足当前需要的内存块，则说明可以成功匹配。优先选择第一个遇到的满足条件的空闲内存块来完成内存分配。如果内存块的大小大于需要的内存大小，则将空闲内存块分裂为两块，一块大小为所需内存大小，另一块则重新进行初始化。重新初始化包括对第一个 page 中表示空闲块代销的成员变量进行设置，其应当设置为剩下的空闲块大小，并将这个分裂出来的空闲块插入到空闲块链表中（链表中的空闲块按照物理地址从小到大排

序）。如果内存块大小与所需大小相同，则不用分裂，对分配出去的物理内存的每一个描述信息（对应的 page 结构）进行初始化，修改 flags 成员变量，设置成非空闲，再将原始空闲块在空闲链表中删除，更新表示总空闲页数量的全局变量；最后返回用于分配到的物理内存的 page 结构指针。

default_free_pages：释放指定的某一物理页开始的连续物理页，并且完成 first-fit 算法中需要的一些信息维护。具体实现如下：

考虑遍历需要释放的物理页的 page 结构，对其进行更新。更新方式：

1.判断原先这些物理页是否被占用，如果释放未被占用的物理页，说明出现异常 。

2.设置 flags 将这些物理页标记为空闲

3.清空这些物理页的引用计数

4.将这一新的空闲块插入到空闲块链表中

5.对空闲块跟其相邻的空闲块合并。通过 merge_backward 函数实现，将指定的某一个空闲块与其链表后的空闲块进行合并，如果合并失败返回 0，否则返回 1.

对于 lab2 的练习 2 通过设置页表和对应的页表项，可建立虚拟内存地址和物理内存地址的对应关系。对于 lab2 的练习 3 当释放一个包含某虚地址的物理内存页时，需要让对应此物理内存页的管理数据结构 Page 做相关的清除处理，使得此物理内存页成为空闲；另外还需把表示虚地址与物理地址对应关系的二级页表项清除。Lab2 的实验

主要理解基于段页式内存地址的转换机制、理解页表的建立和使用方法、理解物理内存的管理方法。

我学习的地方还有很多，操作系统的很多知识还值得我去研究，也希望我在今后的学习中能再接再厉。谢谢大家！