**中国矿业大学计算机学院**

**2020 级本科生课程报告**

课程名称 操作系统课程设计

报告时间 2022年6月3 日

学生姓名 杨再润

学 号 06203203

专 业 数据科学与大数据

任课教师 鲍 宇

目录

[实验八 地址映射与内存共享 3](#_Toc105159437)

[【实验目的】 3](#_Toc105159438)

[【实验内容】 3](#_Toc105159439)

[【实验步骤】 4](#_Toc105159440)

[【运行结果】 4](#_Toc105159441)

[【思考练习】 12](#_Toc105159442)

[【实验感受】 13](#_Toc105159443)

# 实验八 地址映射与内存共享

## 【实验目的】

* 深入理解物理内存的分页管理方式。
* 深入理解操作系统的段、页式内存管理。包括理解段表、二级页表，以及逻辑地址、线性地址、物理地址的映射过程。
* 编程实现段、页式内存管理上的内存共享，从而深入理解操作系统的内存管理。

## 【实验内容】

**3.1 任务（一）：分配和释放物理页**

首先在Linux 0.11 内核项目中添加一个系统调用函数，该函数可以在终端设备上显示物理存储器的信息，包括物理页总数、空闲页数量和占用页数量。

**3.2 任务（二）：跟踪Linux应用程序中的逻辑地址、线性地址、物理地址的映射过程**

首先需要使用Bochs虚拟机提供的Debug调试功能启动Linux 0.11内核项目，然后在Linux终端上运行一个Linux应用程序，此应用程序的main函数中主要包括了一个死循环，使用此应用程序创建的进程开始运行后，会很快进入死循环导致进程无法结束。此时需要使用Bochs调试器提供的暂停虚拟机的命令让应用程序暂停（一般会暂停在死循环对应的指令中），然后在Bochs提供的调试命令的帮助下，根据终端上输出的变量i的逻辑地址计算出它的线性地址，再计算出它的物理地址。最后，使用Bochs调试命令直接修改物理内存让变量i的值变为0，从而结束死循环使进程结束运行。在此实验过程中可以通过一个实际的例子，体验到逻辑地址到线性地址，再到物理地址的转换过程。

**3.3 任务（三）：输出应用程序进程的页目录和页表**

前面的实验过程中主要是通过可视化窗口以图形化的方式展示二级页表。在下面的实验中，会通过编写源代码的方式使一个应用程序将当前进程的二级页表使用文本方式打印出来。读者应该仔细学习这部分源代码，尝试从编程的角度更加深入的理解二级页表的组织方式，并为后面通过共享物理页的方法实现内存共享功能打下基础。

**3.4 任务（四）：用共享内存做缓冲区解决生产者—消费者问题**

将一个共享的物理页映射到不同的逻辑地址空间，从而实现在进程间共享内存的方法，进而使用共享内存作为缓冲区来解决生产者—消费者问题。

本实验需要在Linux 0.11的内核中添加shmget与shmat两个共享内存的系统调用函数。在“学生包”本实验对应的文件夹下提供了一个sem.c文件，其中包含了信号量的四个系统调用函数（与实验七中的相同），以及共享内存的两个系统调用函数所对应的内核函数sys\_shmget和sys\_shmat，这里只是实现了一个简化后的版本：即只能将一个物理页映射到不同进程的逻辑地址空间的末尾。

## 【实验步骤】

**3.1 任务（一）：分配和释放物理页**

* 物理内存的管理
* 通过编程的方式练习分配物理页和释放物理页

**3.2 任务（二）：跟踪Linux应用程序中的逻辑地址、线性地址、物理地址的映射过程**

* 跟踪Linux应用程序中的逻辑地址、线性地址、物理地址的映射过程
* 启动调试Linux内核以及Linux应用程序
* 调试应用程序并暂停
* 处理器通过段表（GDT 和 LDT）将应用程序中的逻辑地址映射为线性地址
* 处理器通过页目录和页表将线性地址映射为物理地址
* 深入研究Linux 0.11应用程序进程内存的管理方式
* 内核逻辑地址空间的初始化

**3.3 任务（三）：输出应用程序进程的页目录和页表**

* 添加一个系统调用号为87的系统调用
* 源代码修改完毕后生成项目，确保没有语法错误和警告
* 按F5启动调试，待Linux011完全启动后，使用vi编辑器新建一个main.c文件
* 保存main.c文件后退出vi编辑器
* 通过命令 vi a.txt 打开vi编辑器查看文本内容，分析输出的结果

**3.4 任务（四）：用共享内存做缓冲区解决生产者—消费者问题**

* 将sem.c文件中的四个信号量的系统调用和两个共享内存的系统调用添加到内核中
* 生成项目，确保没有语法错误和警告。
* 将“学生包”本实验文件夹下的pc.c文件放入软盘B中。
* 按F5启动调试。使用mcopy工具将软盘B中的pc.c文件复制到硬盘的当前目录。
* 使用命令vi pc.c打开源代码文件，这些源代码仍然是使用文件作为生产者和消费者之间的共享缓冲区，请读者在此基础上将其修改为使用共享内存作为缓冲区。
* 代码修改完毕后，退出vi编辑器。
* 在Linux中依次执行下面的命令，运行app

## 【运行结果】

**3.1 任务（一）：分配和释放物理页**

**物理内存的管理**

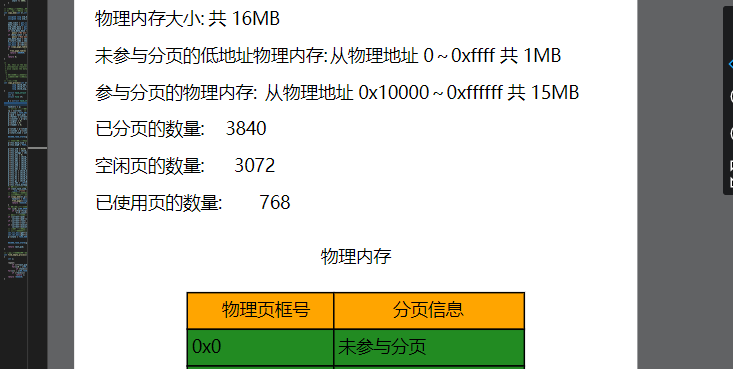


图 物理内存在刚刚完成初始化后的情况

物理内存刚刚完成初始化，其中1MB以下的物理内存未参与分页，1M以上的物理内存在完成分页后包含了已使用的物理页和空闲页。

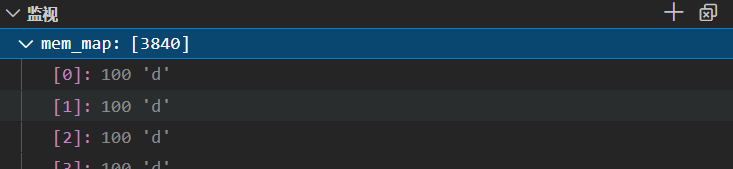


图 页表数组

将全局变量“mem\_map”添加到“WATCH”窗口查看其内容，如图所示。数组mem\_map的大小是3840，也就是说由mem\_map管理的物理页数量是3840，如果加上不由mem\_map管理的0x100个物理页（即1MB以下的未参与分页的物理内存），则物理页的总数正好是4096，4096个大小为4KB的物理页正好是16MB。



图 释放物理页的p值

记录下参数p的值，p指向一个已经结束运行的进程的进程控制块，同样也是一个物理页的基地址，将p的值除以4096就可以得到对应的物理页框号。

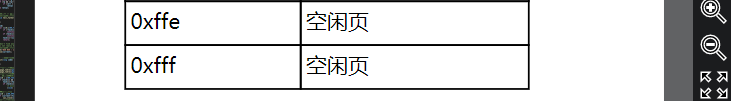


图 内存块被释放

在可视化窗口中刷新物理内存的信息，可以看到p指向的物理页已经被释放了，也就是说进程结束后，用于存放其进程控制块信息的那一整页物理内存被系统回收了。

**通过编程的方式练习分配物理页和释放物理页**

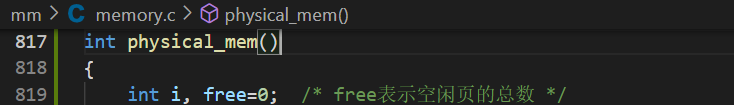


图 添加函数

将其中的函数physical\_mem复制到Linux 0.11 内核项目下的mm/memory.c文件的末尾处。



图 添加函数声明

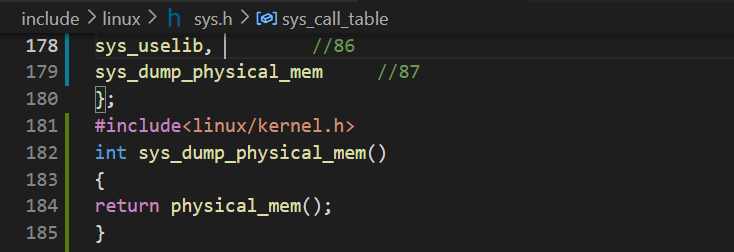


图 添加系统函数调用

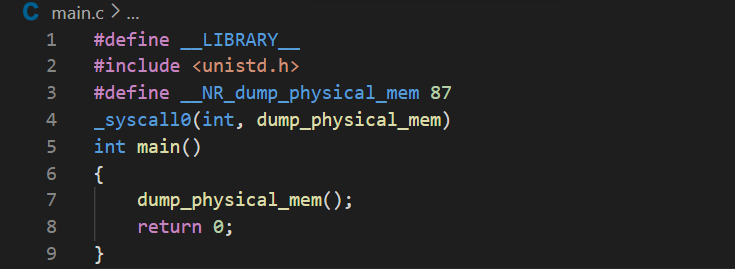


图 新建一个main.c文件

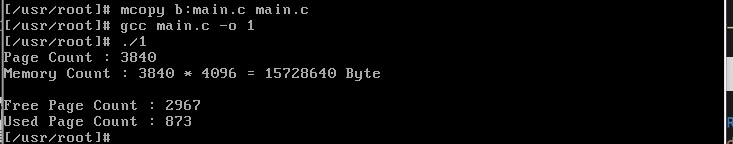


图 运行结果

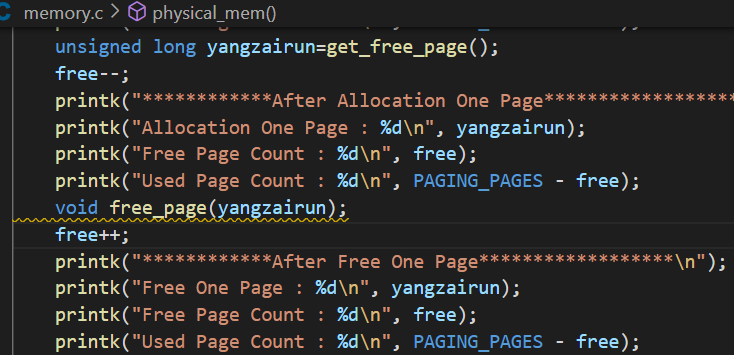


图 自己编写添加的代码部分

在输出物理内存的信息后调用一次get\_free\_page函数分配一个空闲的物理页，然后输出空闲页和占用页的数量，最后再调用一次free\_page函数将刚刚分配的物理页进行回收，同样需要输出空闲页和占用页的数量。

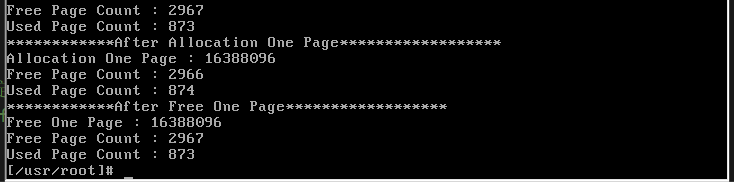


图 修改后的运行结果

图中内存地址的格式为十进制，换算成十六进制后为0XFA1000.

**3.2 任务（二）：跟踪Linux应用程序中的逻辑地址、线性地址、物理地址的映射过程**

**深入研究Linux 0.11应用程序进程内存的管理方式**

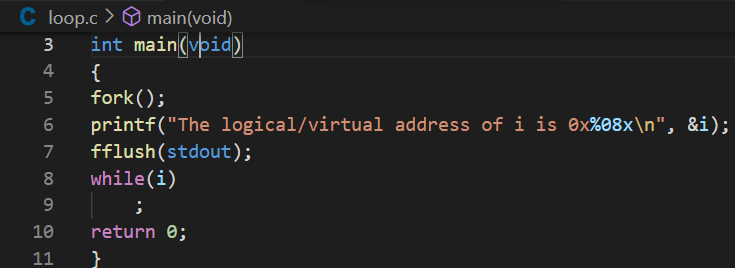


图 loop.c代码内容

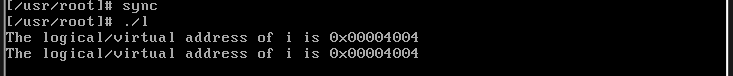


图 loop.c运行结果

使用loop命令运行可执行文件，观察打印输出的信息可以发现父进程和子进程中全局变量i的逻辑地址是相同的，说明父进程和子进程使用了完全相同的逻辑地址空间。

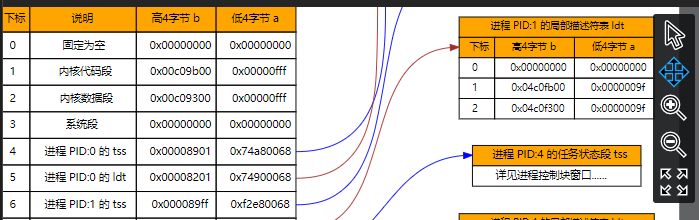


图 命中断点后

在右侧可视化视图顶部的编辑框中输入命令“#gdt”后按回车，可以看到全局描述符表的可视化数据。

**共享物理页**

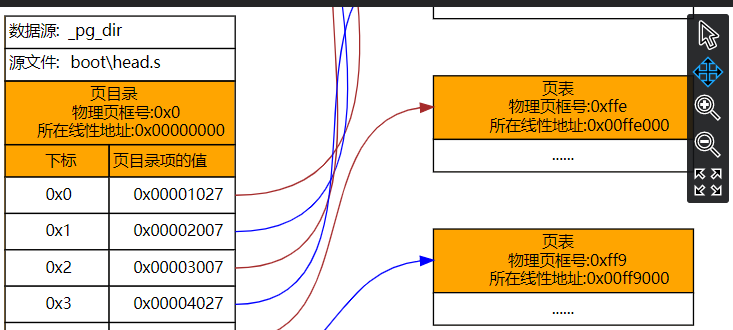


图 进程6的二级页表

在右侧可视化视图顶部的编辑框中输入命令“#vm2”后按回车，可以看到进程6的完整的二级页表映射关系，其中最左边的是页目录（只显示有效的页目录项），中间的是所有页目录项映射的页表（只显示下标是0x40的页目录项所映射的页表中的有效的页表项，其它的页表中也存在有效的页表项并映射有物理页，但是没有显示出来），最右侧的是一个页表（其页目录项的下标是0x40）映射的所有物理页。记录下最右侧各个物理页的页框号，根据这些物理页的线性地址可以确认这些物理页是进程6所使用的。

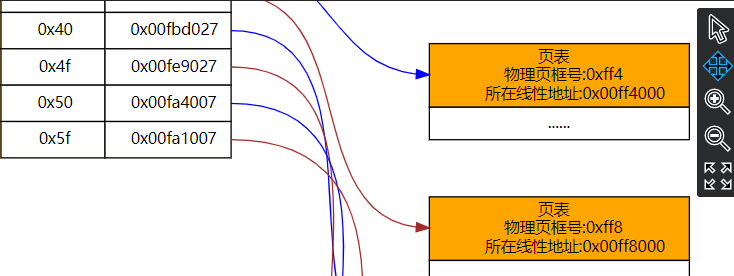


图 进程7的二级页表

使用同样的方法，在“DEBUG CONSOLE”窗口底部的调试命令编辑器中输入调试命令“-exec call vm2(0x50)”后按回车，然后在右侧可视化视图顶部的编辑框中输入命令“#vm2”后按回车，可以看到进程7的完整的二级页表映射关系，根据最右侧物理页的线性地址可以确认这些物理页是进程7所使用的。进程7使用的物理页的页框号与进程6使用的物理页的页框号是完全相同的，说明子进程与父进程共享了相同的物理页，在这些物理页中存储了loop应用程序的指令和数据。

**内核逻辑地址空间的初始化**

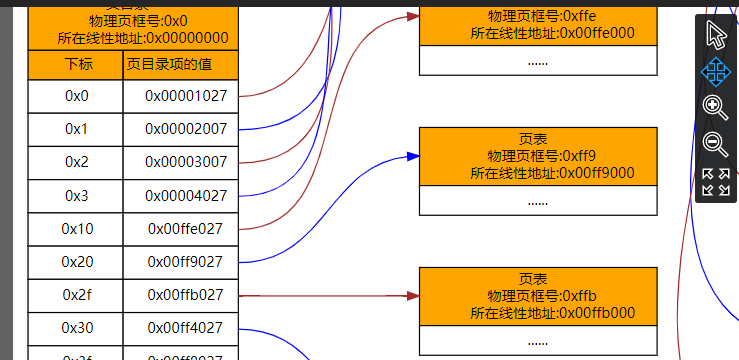


图 二级页表（局部）

根据可视化的图片回答问题：

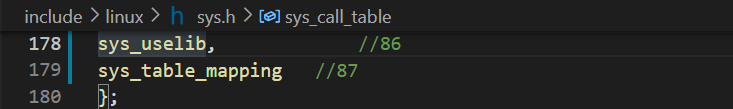
页目录占用的物理页框号：0x0对应线性地址0x00000000

页表 占用的物理页框号：0x1对应的线性地址0x00001000

物理页的对应线性地址：0x00000000到0x

00013000

**3.3 任务（三）：输出应用程序进程的页目录和页表**



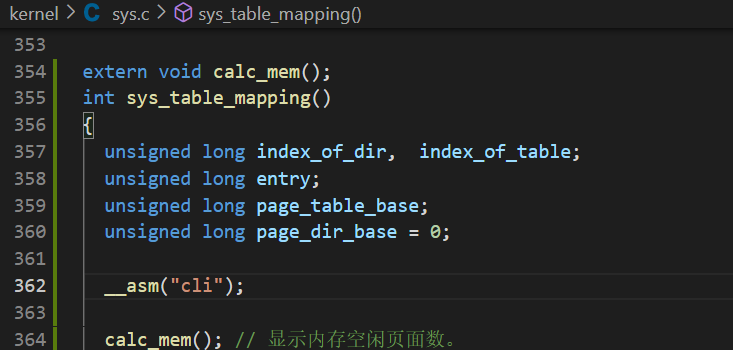


图 添加系统调用

添加一个系统调用号为87的系统调用，该系统调用的内核函数sys\_table\_mapping写在kernel/sys.c文件的末尾。

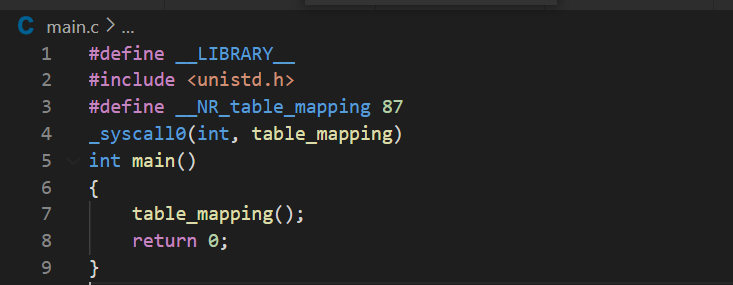


图 编写的测试文件

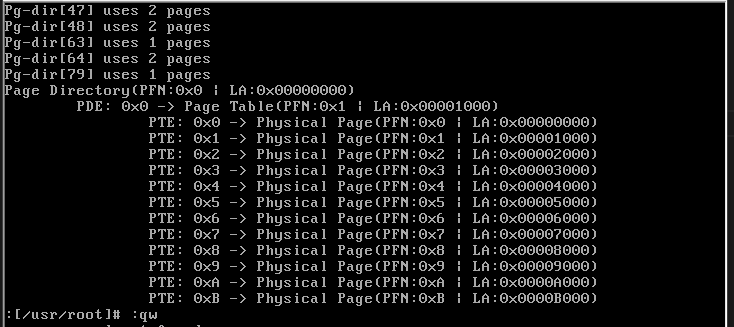


图 测试运行结果（局部）

此行代码的作用是得到一个页表的逻辑地址，其中entry是页目录项，取其中的高20位就可以得到页表的物理地址，而之前的实验中也介绍过，页表的物理地址与其线性地址是相同的，而且在内核模式下段基址为0，所以就可以直接将页表的物理地址作为其逻辑地址了。

**3.4 任务（四）：用共享内存做缓冲区解决生产者—消费者问题**

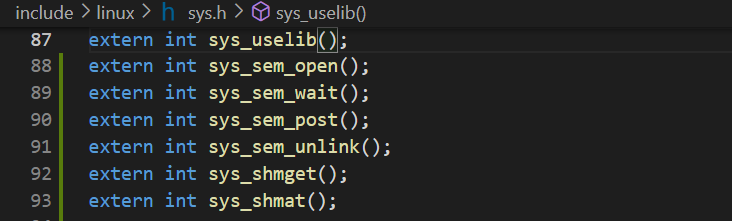


图 添加六个系统调用

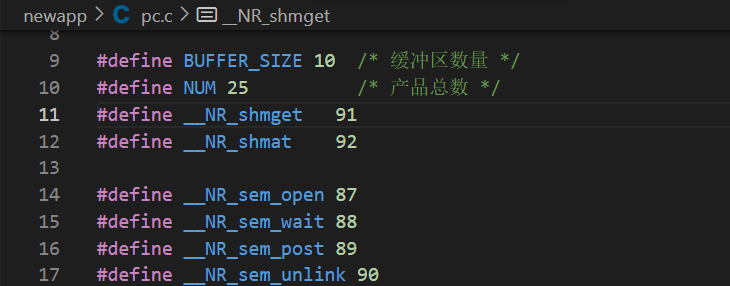


图 编写的pc.c

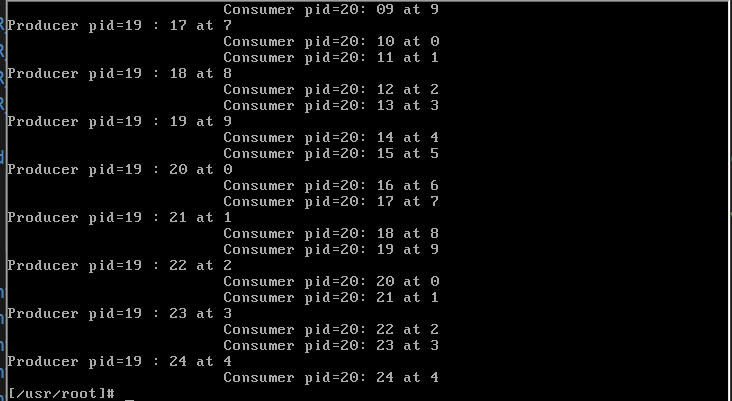


图 运行结果

此应用程序执行到最后会出现错误提示“trying to free free page”，这是由于sys\_shmget函数在使用get\_free\_page函数申请到一个物理页时，将此物理页在mem\_map中记录的引用计数设置为1，而再次调用sys\_shmget函数使用key从vector中获取共享的物理页时，没有增加此物理页在mem\_map中的引用计数。所以，当生产者进程和消费者进程退出时，都会调用free\_page函数释放此物理页，每调用一次就会将mem\_map中的引用计数减1，导致一个物理页的引用计数被减为了负数，这是Linux物理页管理程序所不允许的，所以执行了mm/memory.c中free\_page函数的最后一行（第310行）代码打印出错信息。

## 【思考练习】

* + - 1. 需要实现一个将共享的物理页从当前进程的二级页表映射中移除的内核函数,其原型可以为：

void cancel\_mapping(const void\* linearaddr)

参数linearaddr是需要取消映射的物理页的线性地址，根据此线性地址找到对应的页表项，然后将页表项置为0即可。

* + - 1. 之前的源代码中只是用全局的vector数组保存了共享内存的物理页的基址，现在需要为共享内存定义一个结构体，其中除了保存物理页的基址外，还需要保存共享内存的引用计数（可以参考信号量的引用计数），再使用此结构体定义一个全局的共享内存数组。当进程调用shmdt函数关闭共享内存时，首先将共享内存的引用计数减1，然后调用cancel\_mapping函数将共享的物理页从二级页表映射中移除。由于shmdt的第二个参数startaddr是共享内存的逻辑地址，需要为其加上current->start\_code转换为线性地址后，才能作为cancel\_mapping函数的参数。最后，判断如果共享页的引用计数的值大于0，说明仍然有其他进程在使用此共享内存，就结束（不释放物理页）；否则，需要调用free\_page函数释放物理页（只释放这一次）。

## 【实验感受】

* 系统采用的老版本C语言与我所学的在语法上有些许出入，常常带来意想不到的麻烦。
* 通过这次实验，也掌握了vi编辑文本的技能，还有一些其他的常用Linux命令
* 通过本次实验，掌握了VSCode的调试功能
* 本次实验掌握了系统调用的实现，深入内核直达硬件，加深了对计算机体系结构的认识
* 对操作系统将逻辑地址和物理地址的相互转换的过程更加了解