

**操作系统课设报告**

学 院：信息工程学院

专 业：计算机科学与技术

姓名/学号： 王新超（2021904517）

王一博（2021902655）

吴佩颖（2021900544）

日 期：2024.6.28

**实验1 Linux内核编译**

1. 实验目的

（1）学习重新编译Linux内核的方法

（2）理解Linux标准内核和发行版本内核的区别

1. 实验内容

**1、实验原理**

（1）Linux内核：内核指的是一个提供设备驱动、文件系统、进程管理、网络通信等功能的系统软件，内核并不是一套完整的操作系统，它只是操作系统的核心。

（2）Linux内核体系结构简析：

① 在系统层次结构中，最上面是用户（或应用程序）空间，这是用户应用程序执行的地方。

② Linux内核位于用户空间之下。

③ 由于内核和用户空间的应用程序使用的是不同的保护地址空间；每个用户空间的进程都使用自己的虚拟地址空间，而内核则占用单独的地址空间。因此，GNUCLibrary（glibc）的作用为提供连接内核的系统调用接口，并提供在用户空间应用程序和内核之间进行转换的机制。

④ Linux内核可以进一步划分成3层。最上面是系统调用接口，它实现了一些基本的功能，例如read和write。系统调用接口之下是内核代码，可以更精确地定义为独立于体系结构的内核代码。

**2、实验内容**

（1）下载统一发行版本的任意版本号内核；

（2）编译运行编译的内核；

（3）使用uname-r命令查看是否运行成功。

1. 实验步骤

**1、查看内核版本和下载好的内核压缩包**

#uname-r使用该命令查看当前内核的版本。例如，本次实验我们的版本为5.19.0-43-generitic，说明此时的内核版本为5.19.0。

**2、解压内核压缩包**

将压缩包解压到/usr/src，本实验中我们使用命令：sudo cp linux-5.19.0.tar.xz/usr/src进行解压,输入已设置好的密码后，利用cd指令跳转至/usr/src，利用ls查看是否成功。

**3、配置内核**

利用sudo make menuconfig该命令打开config菜单来配置哪些需要直接编译进内核，哪些编译成模块，哪些不编译。随后使用save保存对应的配置文件.config。

**4、编译内核**

利用sudo make-j8该命令命令开始编译内核。使用-j选项来多线程处理，可以更有效的利用CPU资源。而make-j8，即让make最多允许8个编译命令同时执行。

**5、安装模块和内核**

利用make指令安装模块和内核。

**6、重新启动，检查新内核**

利用uname-r指令，再次查看内核版本，检查是否成功。

1. 实验结果
2. **查看内核版本和下载好的内核压缩包**

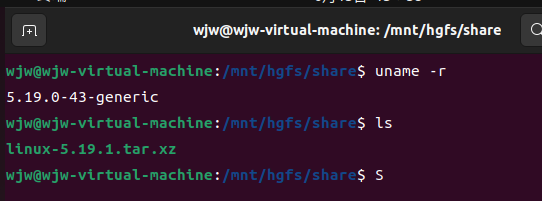
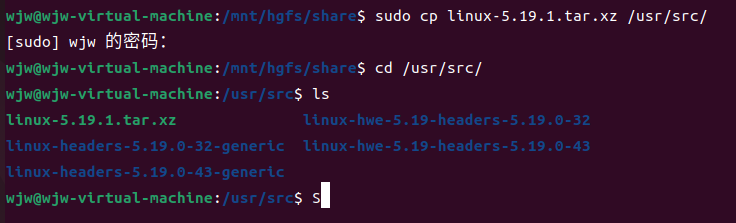


图1-1 查看内核版本和下载好的内核压缩包

1. **解压内核压缩包**



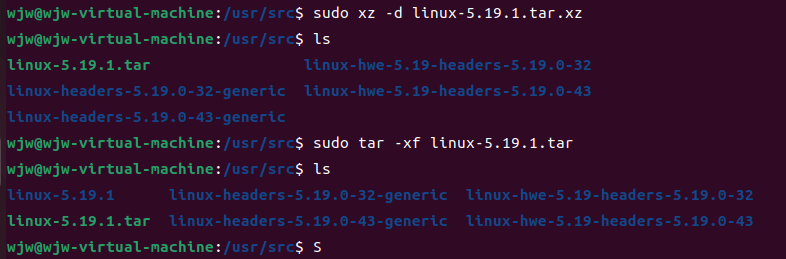
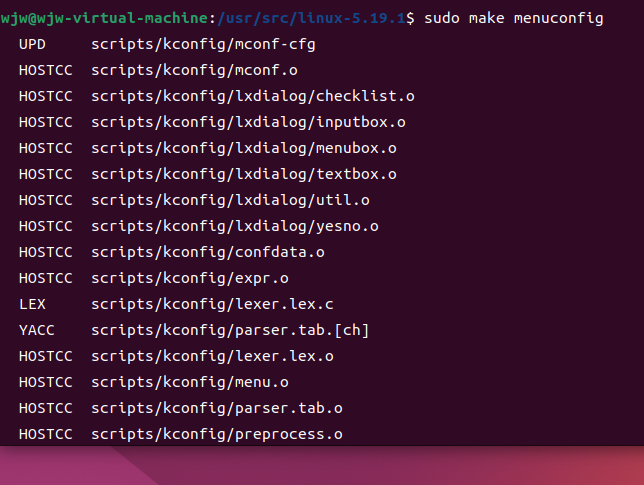
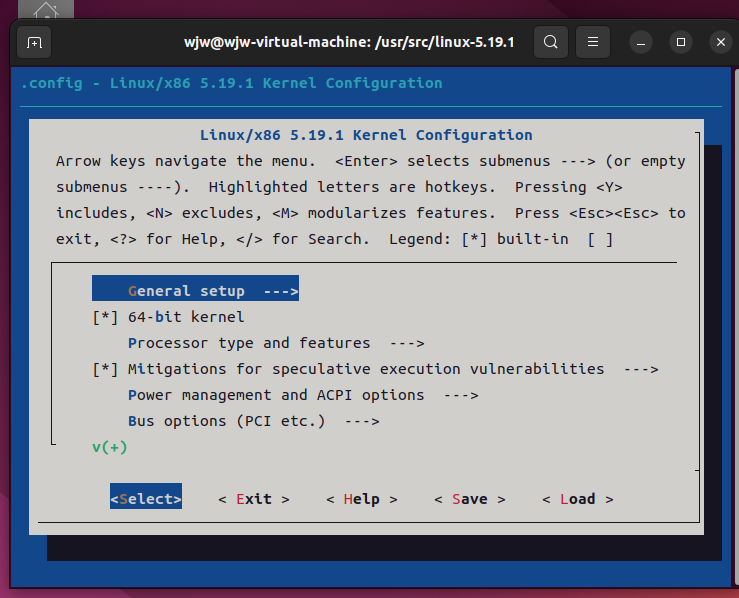


图1-2 解压内核压缩包

1. **配置内核**





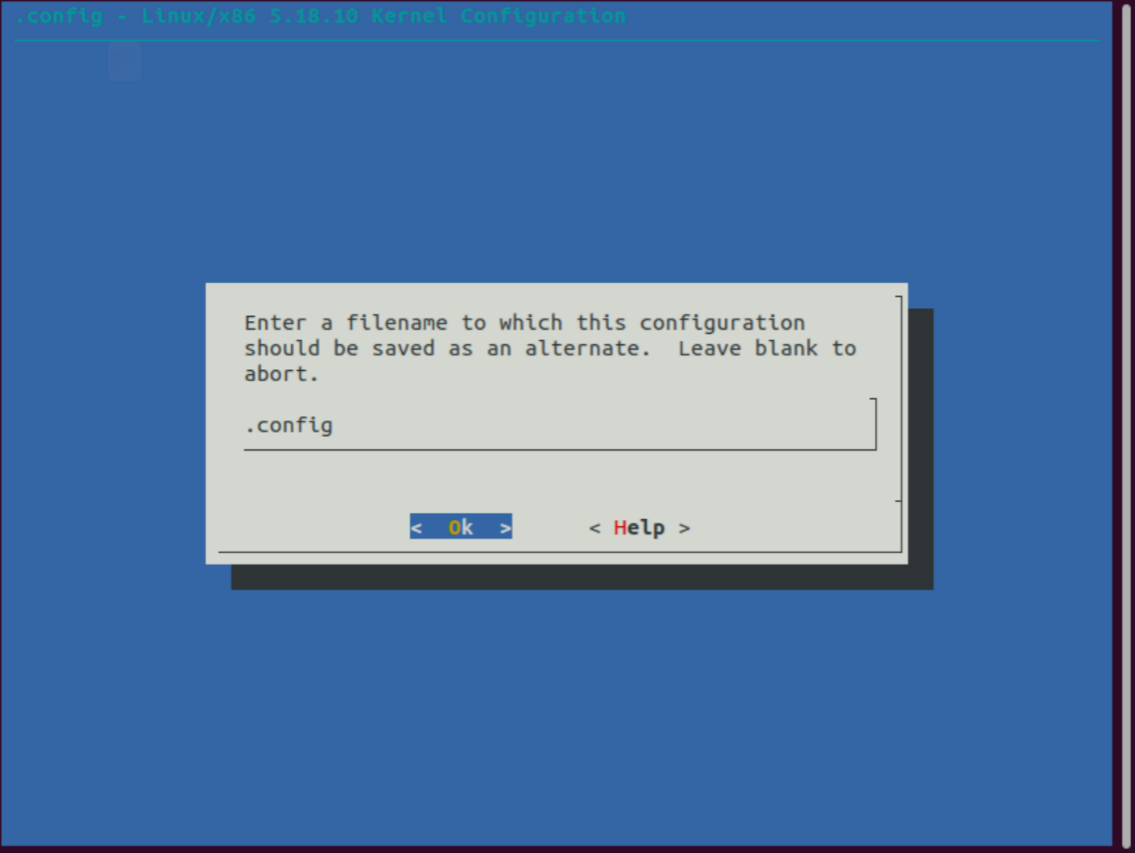
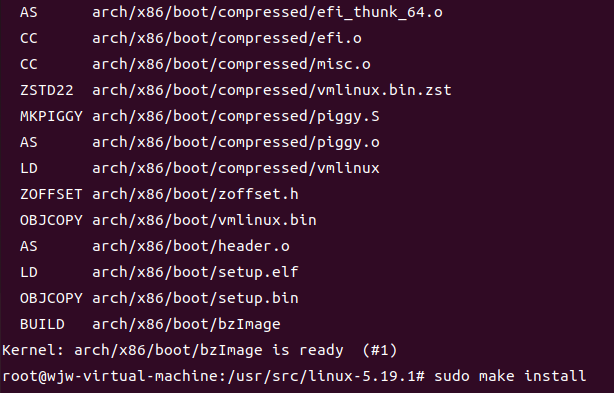


图1-3 配置内核

1. **编译内核**





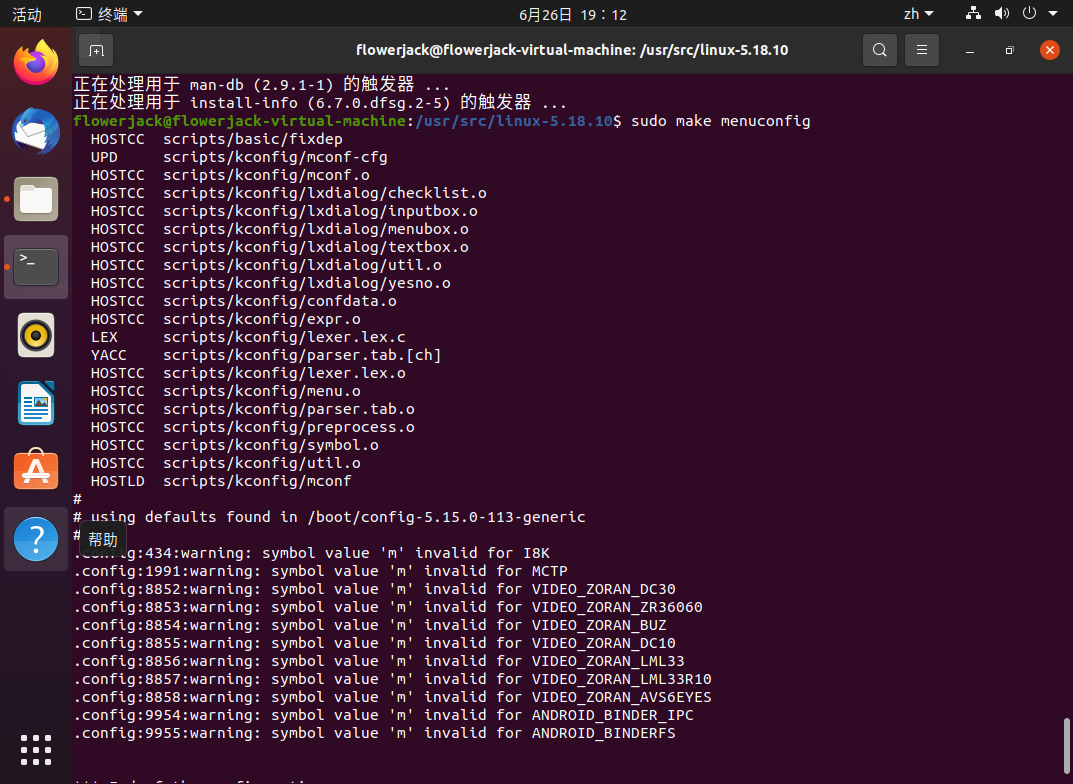


图1-4 编译内核

1. **安装模块和内核**

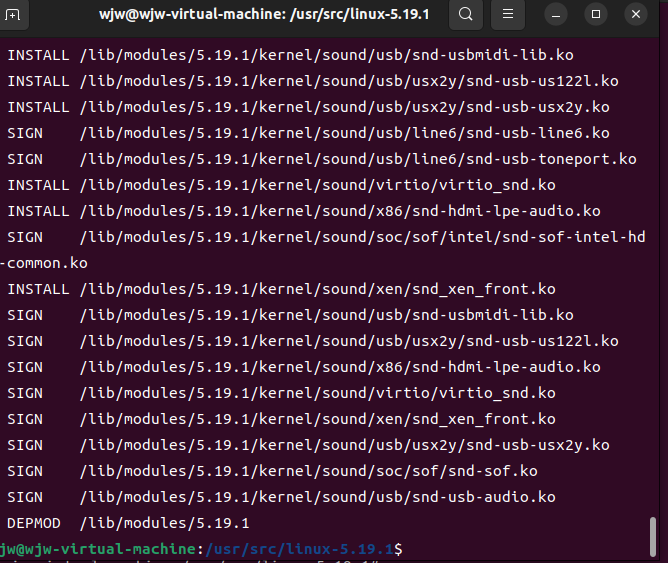


图1-5 安装模块完成

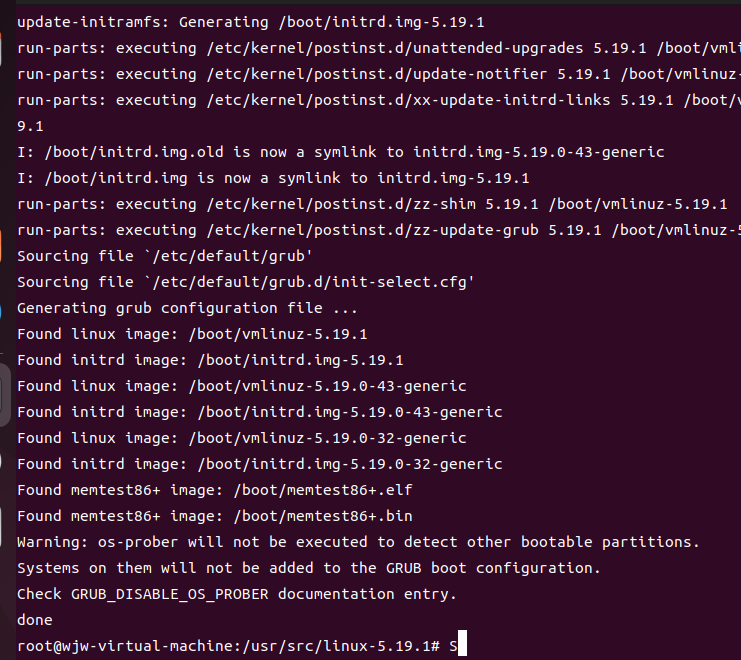


图1-6 安装内核完成

1. **重新启动，检查新内核**

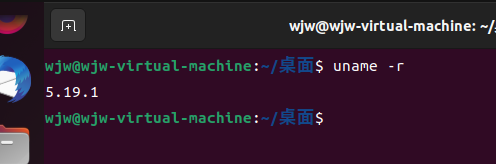


图1-7 检查新版本内核，内核更换完成

1. 实验总结

**1、实验小结**

在Linux内核编译的实验中，我学到了很多关于内核编译的知识，以下是我的总结：

首先我了解到内核编译的基本流程，第一步是获取内核源代码，第二步是选择要编译的内核版本和配置选项，第三步基于以上两步进行配置和编译，第四步需要安装内核和模块，最后一步为重启系统，选择新内核启动，观察是否安装成功。

然后，我了解到常用的内核配置工具，有make menuconfig、make xconfig和make gconfig。其中make menuconfig是在终端中进行的，使用简单，适合在服务器等没有图形界面的设备上使用；make xconfig和make gconfig采用图形界面，在Linux桌面环境下使用较为方便。

最后，我了解到在内核编译过程中，我们需要在编译前需要确定内核版本和配置选项，并进行配置。配置完成后，使用make命令进行编译。如果编译顺利完成，则可以使用make install命令安装内核。

通过本次实验，我对Linux内核编译有了更深入的认识，同时也学到了许多实用的操作技巧，对我Linux与操作系统的学习有所帮助。

**2、实验反思与体会**

在Linux内核编译实验中，我也遇到了一些困难和疑惑，同时也有了很多收获和体会。以下是我的总结：

在本次实验中我深切感受到阅读文档一定要仔细，在编译内核的过程中，我曾因为没有认真阅读内核文档，从而遇到了一些困难和错误。同时在进行实验的过程中，我们需要选择合适的内核版本，因为不同的系统环境需要选择适合的内核版本，否则会出现许多错误。在编译和安装的过程中会出现一些报错，需要我查阅有关资料进行解决。

在本次实验中，由于内核编译需要大量时间，有时需要几个小时来等待编译完成。在本次实验中我也有了很多收获，我学到了很多关于Linux内核的知识，对系统底层有了更深入的了解；体验了编译内核的整个过程，掌握了编译内核的基本流程和操作技巧；提升了解决问题的能力，学会了如何查找和解决编译过程中的错误。

通过本次实验，我体会到编译内核需要耐心和毅力，不能急于求成，需要一步一步来，想要达到理想的目标需要不断学习和实践。同时我也明白了阅读内核的文档和代码是必须的，要多看官方文档，多遵循开源精神，才能更好地学习和使用操作系统。

**实验2 添加系统调用**

1. 实验目的

（1）学习Linux内核的系统调用方法。

（2）理解并掌握Linux系统调用的实现框架、用户界面、参数传递、进入/返回过程。

1. 实验内容

**1、实验原理**

（1）系统调用：操作系统提供给用户程序调用系统服务(硬件设备)的一组"特殊"接口。

（2）执行系统调用的方法主要有三种：系统中断，glibc库函数，syscall调用

在本次实验中，我们重点了解glibc库函数与syscall调用的优劣，重点学习最后一个系统调用方法，并采用该方法进行实验。

表2-1 两种系统调用方法的优劣

|  |  |  |
| --- | --- | --- |
|  | 优势 | 劣势 |
| Glibc库函数 | 只需了解glibc提供的API原型，无须知道更多的细节，如系统调用号。其次，具有更好的移植性 | 没有封装某个内核提供的系统调用时无法使用 |
| Syscall调用 | 可定制自己的系统调用 | 使用不便 |

**2、实验内容**

通过内核编译法添加一个不用传递参数的系统调用，其功能可自定义。

（1）添加系统调用号，系统会根据这个号找到syscall\_table中的相应表项。具体做法是在syscall\_64.tbl文件中添加系统调用号和调用函数的对应关系。

（2）实现my\_syscall，在kernel/sys.c中添加自己的服务函数，然后为该函数在syscalls.h中添加函数声明。

（3）完成准备工作之后，就可以编译内核。

三、实验步骤

**1、获取root权限**

利用sudo指令获取root权限，输入密码后进入。

**2、进入kernel目录**

打开sys.c，并加入函数输出简单字符串，在vi中编辑。

**3、使用文本编辑器查看结果**

**4、添加声明**

在syscalls.h中插入asmlinkage long sys\_helloworld(void)。

**5、添加系统调用id**

添加系统调用id保存在syscall\_64.tbl文件中。

**6、配置内核**

修改配置文件。

**7、编译和安装内核**

先利用make clean指令清除上次的make命令所产生的object文件(后缀为“.o”的文件)及可执行文件。Make clean后重新编译内核文件make -j8。

**8、重启系统，并验证系统调用是否成功**

重启系统后，编写验证程序，将其命名为text.c，用于验证系统调用是否成功，最后对验证程序进行调用。

四、实验结果

**1、获取root权限**

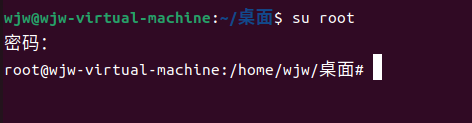


图2-1 获取root权限

**2、进入kernel目录**

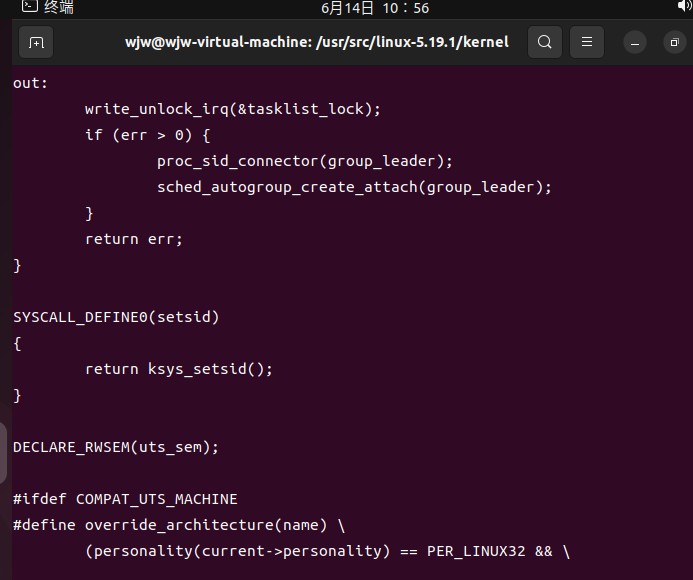


图2-2 进入kernel目录

**3、使用文本编辑器查看结果**

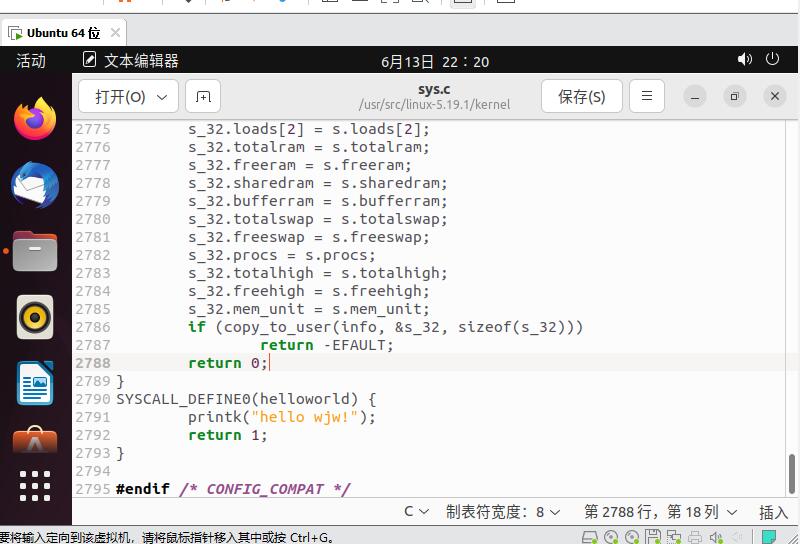


图2-3 使用文本编辑器查看结果

**4、添加声明**

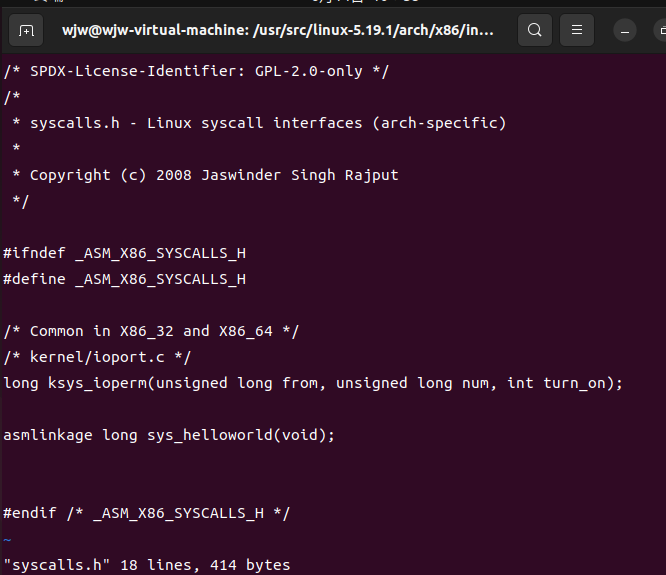




图2-4 添加声明

**5、添加系统调用id**

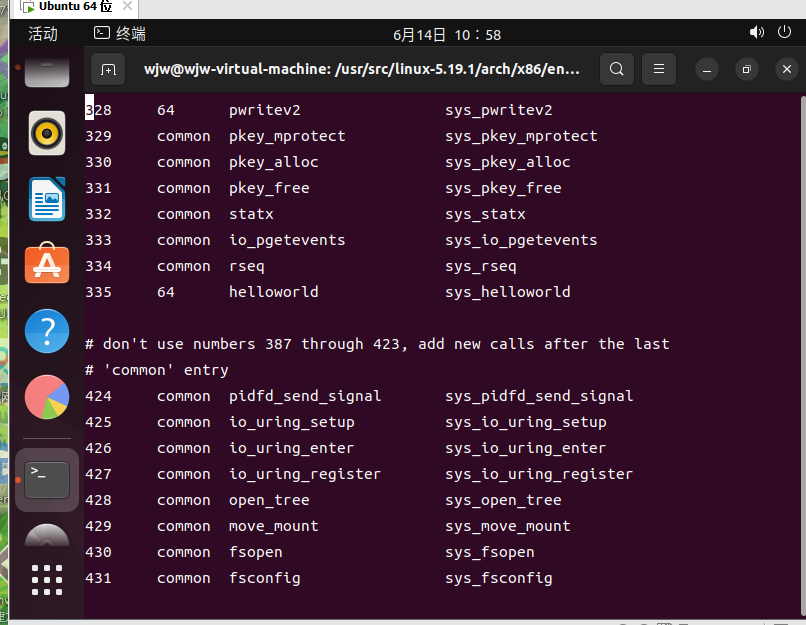


图2-5 添加系统调用id

**6、配置内核**

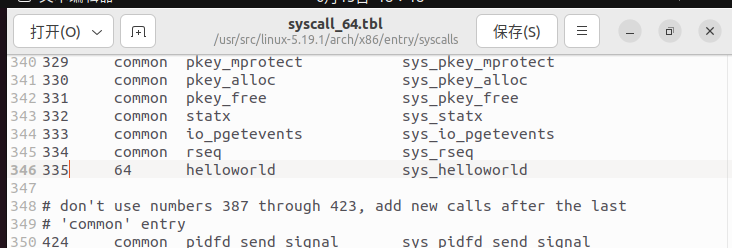


图2-6 修改配置文件

**7、编译和安装内核**

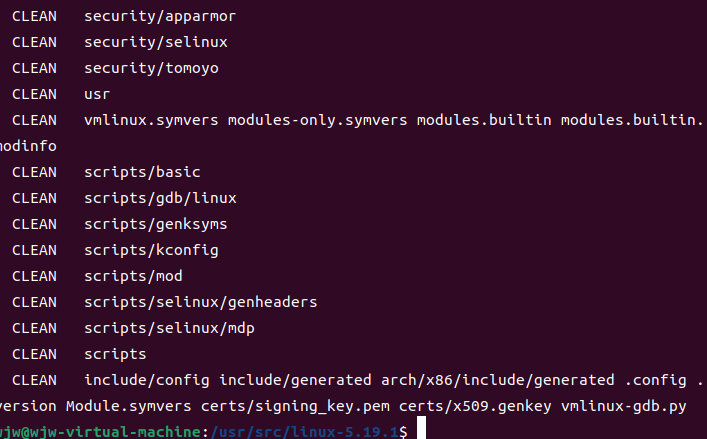


图2-7 makeclean命令

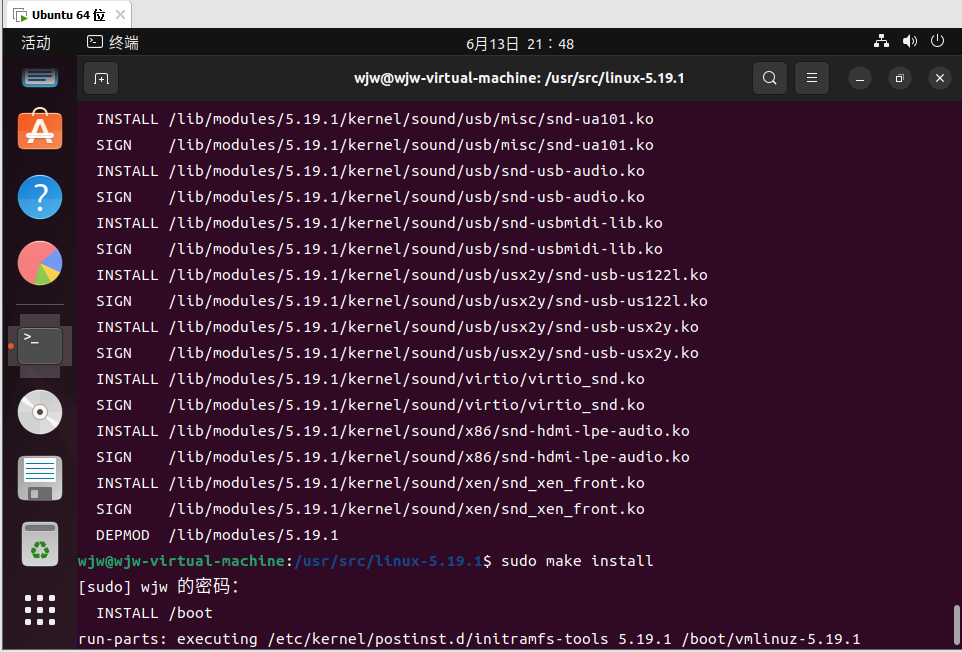


图2-8 make modules\_install命令

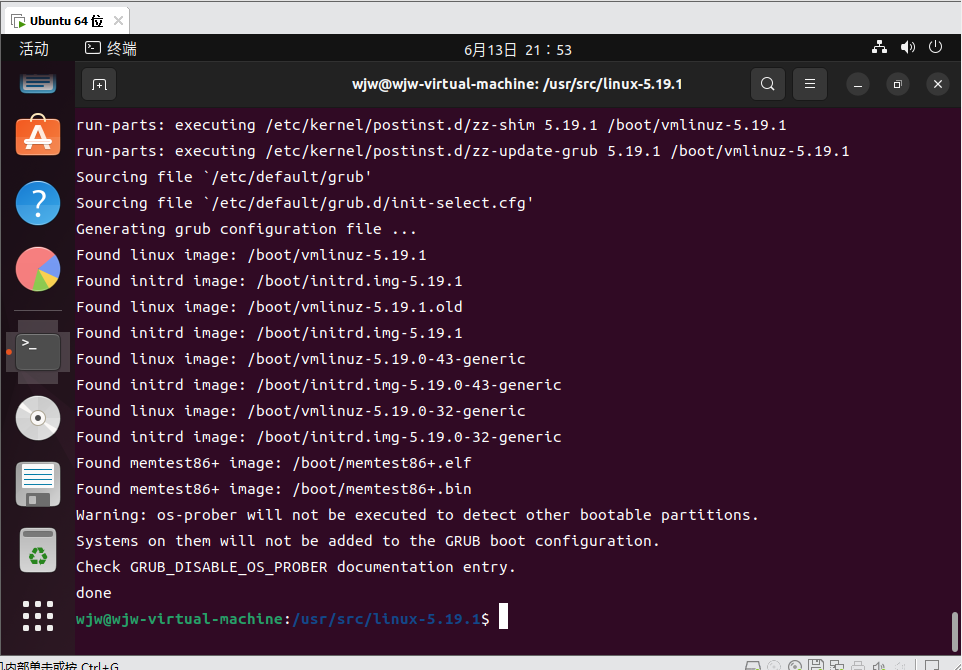


图2-9 Make install命令

**8、重启系统，并验证系统调用是否成功**



图2-10 编写验证程序

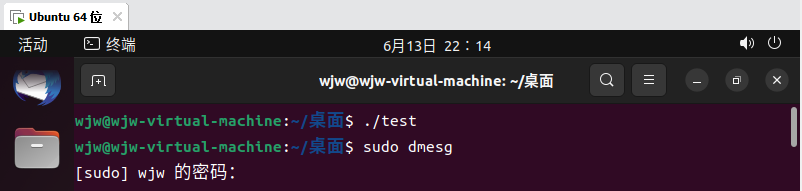


图2-11 调用验证程序

1. 实验总结

首先系统调用是Linux向用户程序提供服务的主要方式，它是用户程序和内核之间的接口。并且系统调用可以提供文件操作、进程控制、网络通信等多种服务。而系统调用的实现需要用户程序和内核之间进行协作。用户程序通过libc库间接调用系统调用，在库函数的帮助下，将参数传给内核，并通过软中断触发内核执行相应的操作。内核接收到软中断后，根据调用号去查找对应的系统调用服务函数，执行相应的操作，并将结果返回用户程序。

通过这次实验，我更深入地了解了系统调用的原理和实现方式，掌握了调试技巧。此外，我还对Linux系统有了更深入的了解，了解了Linux的内部实现。

通过这次实验，我对Linux的内部实现和系统调用机制有了更深入的了解，同时也充分感受到了自己的不足之处和需要继续提升的地方。我会继续学习，不断进步，为自己的技术水平打下坚实的基础。

**实验4 简单文件系统设计**

1. 实验目的

理解文件存储空间的管理、文件的物理结构和目录结构以及文件操作的实现。加深对文件系统内部功能和实现过程的理

1. 实验内容

通过二进制文件模拟简单索引式文件系统的实验。 通过二进制文件模拟磁盘；通过对二进制文件的读写，模拟对磁盘的读写。对于一个文件系统，我们分为内存内和磁盘上两部分。对于内存内，我们需要维护一个虚拟文件系统（VFS），对于磁盘上，我们需要维护一个实际的、完整的文件在磁盘上的内容布局。通过VFS，提供高层次的对文件的各种操作：打开、删除、链接、重命名等。通过文件系统，将 VFS 所表达的各种文件操作通过高效合理的方式组织保存在磁盘上。

磁盘整体布局, 如图4.1

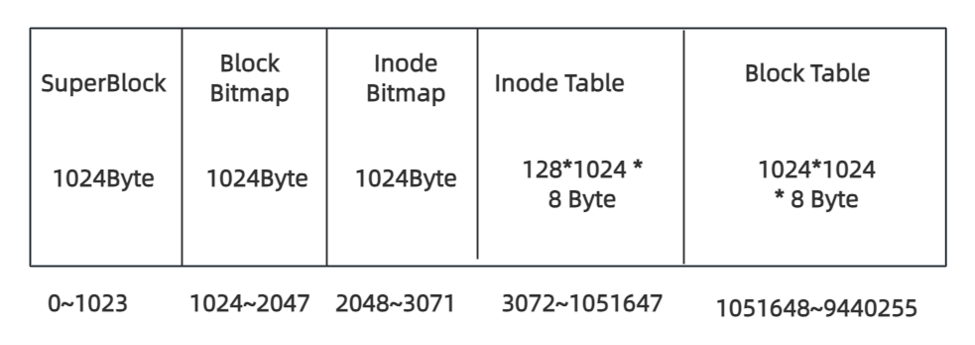


图4-1

各个组成部分的详细解释:

SuperBlock 记录文件系统整体资源情况

Block BitMap 记录 Block 使用情况, 记录文件系统中各个Block是否被分配使用，按Bit记录和操作。Bit为1时为被使用，为0时表示未被使用。在本文件系统中，有1024 \* 8个Block，所以对应BitMap有1024 \* 8 Bit，即1024Byte（一个Block）。

Inode BitMap记录 Inode使用情况, 同Block BitMap，只是对应管理的对象为Inode。

Inode Table 由连续的 Inode 组成，每个 Inode 代表一个文件

Block Table 由连续的 Block 组成，一个或多个离散 Block 记录一个文件的内容。

1. 实验步骤

1.准备一个文件作为二进制文件模拟磁盘

2.为磁盘中的数据结构设置响应的C++类

3.再VFS类中设置一些操作文件系统的命令

4.在主函数main中设置一些与用户交互的命令

1. 实验结果
2. 首先用户运行程序后, 程序先从磁盘文件中读取文件系统，如图4-2所示：

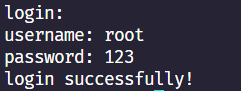


图4-2 读取文件系统

1. 读取文件系统后, 程序等待用户登陆，如图4-3所示：



图4-3

1. 登陆成功后, 显示当前目录, 用户可以使用help命令查看可用命令及其解释, 如图4-4所示:

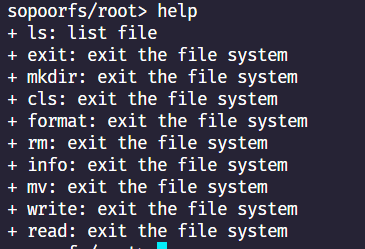


图4-4

1. 用户可使用format指令格式化磁盘, 格式化后的磁盘只有根目录和其下的root用户目录，如图4-5所示：

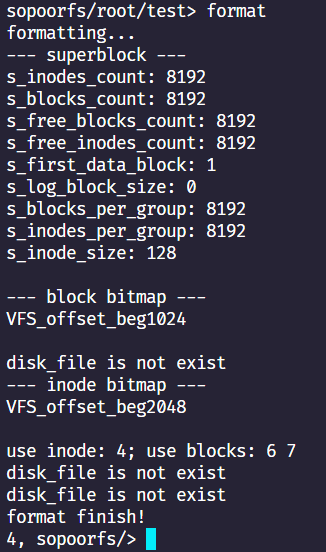


图4-5

1. 用户可用使用cd命令切换当前目录到其父目录或某个子目录，如图4-6所示：

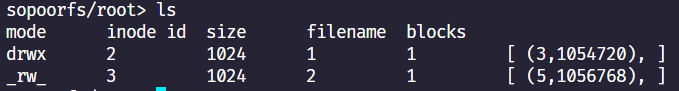


图4-6

1. 用户可用使用ls命令查看当前目录下的文件和目录, 包括具体的信息: 文件大小, inode编号, 文件占用的块数及其块(编号, 物理地址)的列表，如图4-7所示：

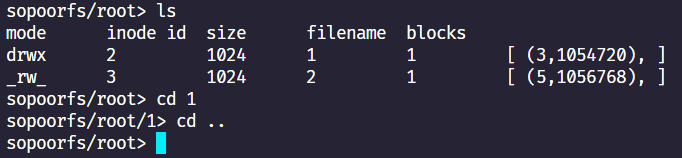


图4-7

1. 用户可用使用touch命令创建普通文件，如图4-8所示：

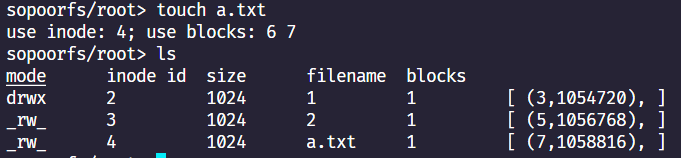


图4-8

1. 用户可用使用rm命令删除当前目录下的文件或文件夹，如图4-9所示：

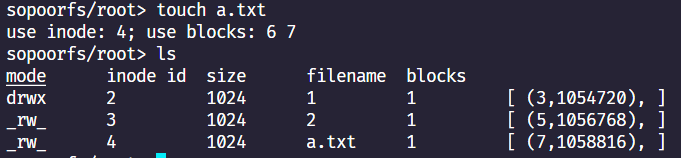


图4-9

1. 用户可用使用mv指令修改当前目录下的文件名或目录名，如图4-10所示：

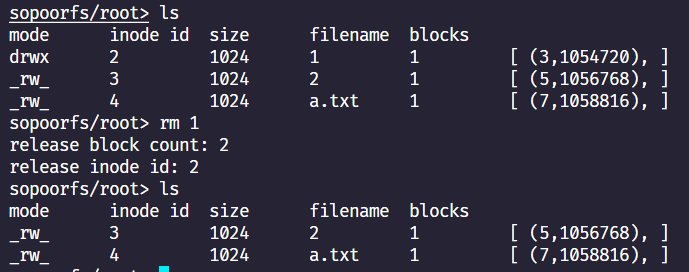


图4-10

1. 用户可用使用write指令向当前目录下的某个文件写入一个字符串, 用户可用指定文件指针的位置和写入的大小，如图4-11所示：

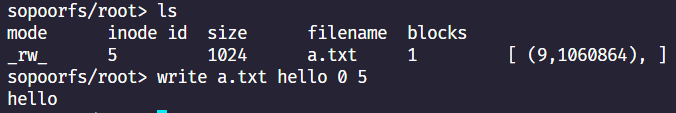


图4-11

根据a.txt的其实物理地址, 我们查看二进制文件的对应位置，如图4-12所示：

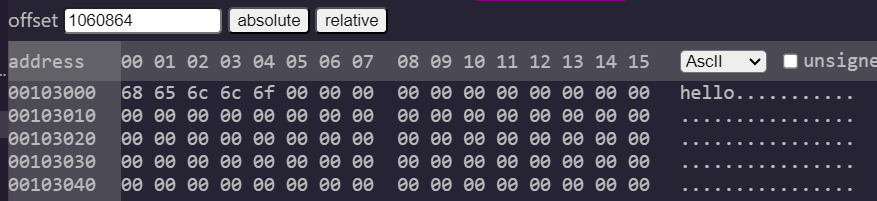


图4-12

1. 用户可用使用read指令向当前目录下的某个文件读入一个字符串, 用户可用指定文件指针的位置和读入的大小，如图4-13所示：

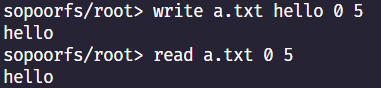


图4-13

1. 实验总结

设计本次文件系统时, 本人参考了经典的Linux文件系统ext2, 将其核心思想进行了实现。但是本次实验不能像那种文件系统一样大而全, 主要使用了inode 这个结构对文件的元数据进行保存, 后面的各种花活也都是建立在inode构造出的inode树的.总的来说, 本次实验并不是特别完整, 只实现了部分功能, 但是使得本人对文件系统的理解也逐渐加深。