重庆邮电大学

大作业报告书

|  |  |
| --- | --- |
| 课程名称： | Linux操作系统 |
| 题 目： | Linux期末大作业 |
| 专 业： | 信息安全 |
| 班 级： | 04042101 |
| 学 号： | 2021212687 |
| 学生姓名： | 梁俊勇 |
| 完成人数： | 1 |
| 起始日期： | 2023.12.29 |
| 成 绩： |  |
| 任课教师： | 杨志刚 |
| 完成时间： | 2024.1.10 |

2023年12月

说 明

|  |
| --- |
| 1. 要求独立完成大作业项目（题目附后）  2. 虚拟机以自己的学号+名字拼音首字母命名，如张三的虚拟机命名为“2020001zs”，确保Shell命令提示符能看到自己的名字信息。不允许有雷同和抄袭情况，否则双方成绩均按不及格处理。  3. 所有的操作结果都要有截图，程序要有完整的代码。  4.完成大作业后必须提交以下内容：  1）大作业报告（电子版）为doc或docx格式，必须以自己的学号+名字命名，如“2020001张三”。  2）必须给出所有项目内容的实现过程，并附1000字以内的大作业小结（严禁抄袭） |

《Linux操作系统期末大作业》

请注意必须自己重新安装，不要克隆或使用他人已安装好的操作系统，否则可能判为抄袭！

要求将计算机名改为名字的拼音全称，以便截图时能看到个人信息！

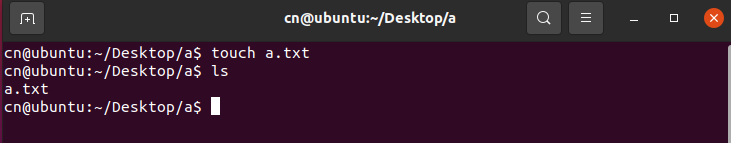
**（一） 系统概要**

请在虚拟机上安装Ubuntu操作系统（请务必重新安装，不要使用已安装的操作系统，避免被误判为抄袭），获取（1）查看内核和操作系统版本，（2）查看MAC地址，硬件地址（3）查看系统的后台任务（4）测试当前主机是否能与百度网站进行联通（5）如何查看系统中已注册的中断及访问频次（1秒更新一次）。（6）使用什么命令查看磁盘使用空间？空闲空间呢?（7）查看当前谁在使用该主机用什么命令?查找自己所在的终端信息用什么命令?（8）如何让系统启动时自动开启防火墙，并且允许IP段192.168.0.0/24访问端口20。要求逐项列出查询命令及结果截图；如需安装新工具，请列出安装命令和结果截图。

**截图示例**

**（1）新增文件**

**命令：touch a.txt**

**截图：**

**（二） 基本操作**

（1）创建一个名为"examusers"的用户组。添加三个用户到这个用户组：user1、user2、user3。为这三个用户设置密码，并确保用户可以使用密码登录。设置用户1（user1）仅具有读取一个特定目录的权限，用户2（user2）可以读取和写入该目录，而用户3（user3）则拥有完全权限。要求逐项列出查询命令及结果截图；如需安装新工具，请列出安装命令和结果截图。

（2）gcc命令的常用选项：-Wall、-l、-c、-o、-g、，使用下面提供的代码并用这些选项编译程序，再通过gdb调试程序，列出程序清单、运行程序、设置断点、清楚断点、显示程序变量值、单步执行。要求逐项列出查询命令及结果截图；如需安装新工具，请列出安装命令和结果截图。

#include <stdio.h>

int main(){

int i =1, j =0;

j = i++

printf("i = %d , j = %d\n",i , j);

}

**（三）进阶操作**

（1）伙伴算法（buddy算法），是为了核心内存管理能够快速响应请求，尽可能地在提高内存利用率的同时减少内存碎片的一种算法。当使用伙伴分配算法进行内存分配时，假设初始时有一个大小为 64KB 的内存空间，然后有以下的内存分配请求序列到达：

进程 A 请求分配 8KB 的内存

进程 B 请求分配 16KB 的内存

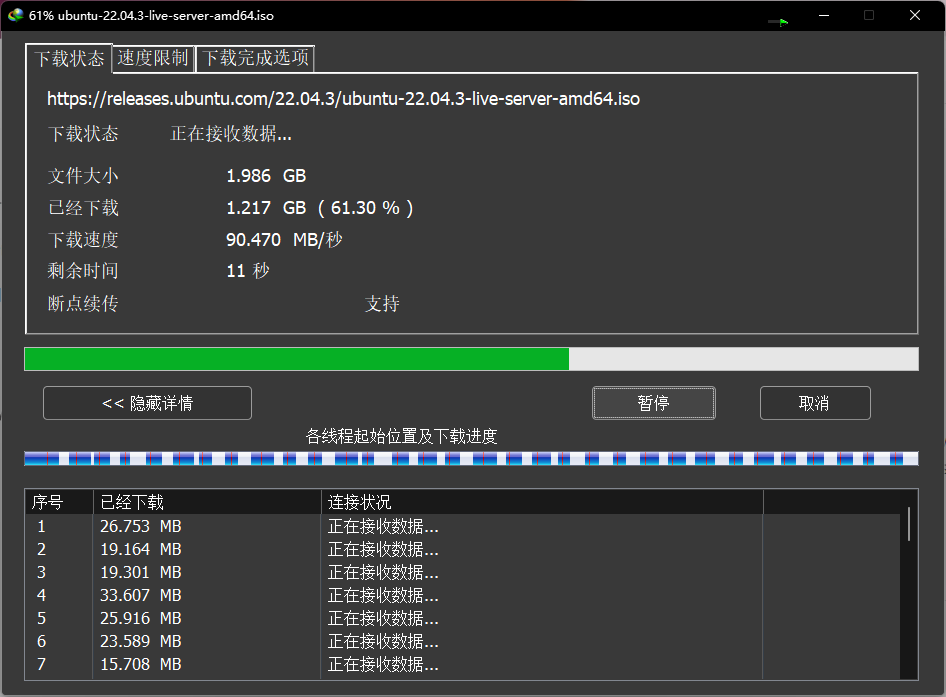
进程 C 请求分配 32KB 的内存

请问，这些请求可以如何被满足？请给出每个请求的分配情况，并在每次分配后展示剩余的空闲内存块情况。要求写出完整的设计思路和实现代码（含注释），以及运行结果截图。

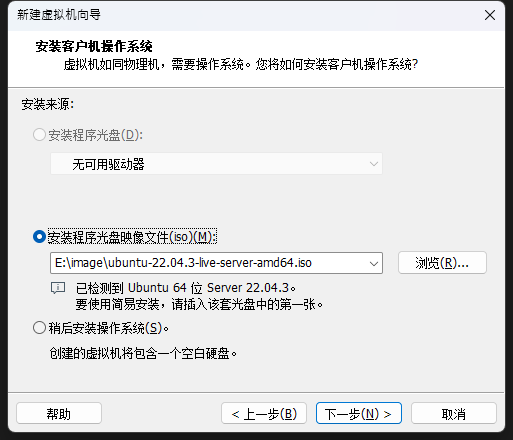
（2）Ext4作为大多数Linux发行版中的默认文件系统，Ext4文件系统中，每个文件都有且仅有一个inode，inode记录了文件的权限、属性以及数据所在的块block号码。请创建一个大小为10MB的文件，根据文件的inode号结合Ext4分区信息，通过系列计算定位该文件的inode Table，并找到存储文件的数据块。请给出计算依据和详细过程，并截图。

（一） 系统概要

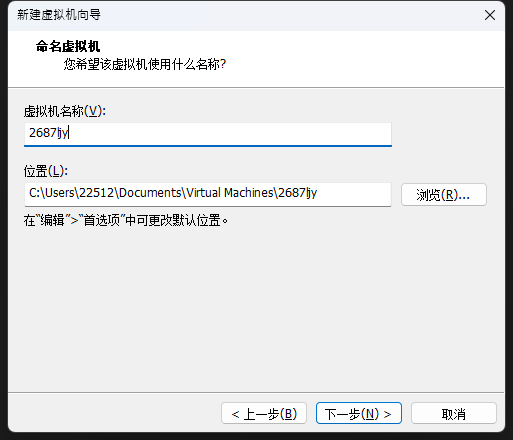
安装系统：下载最新版ubuntu server镜像



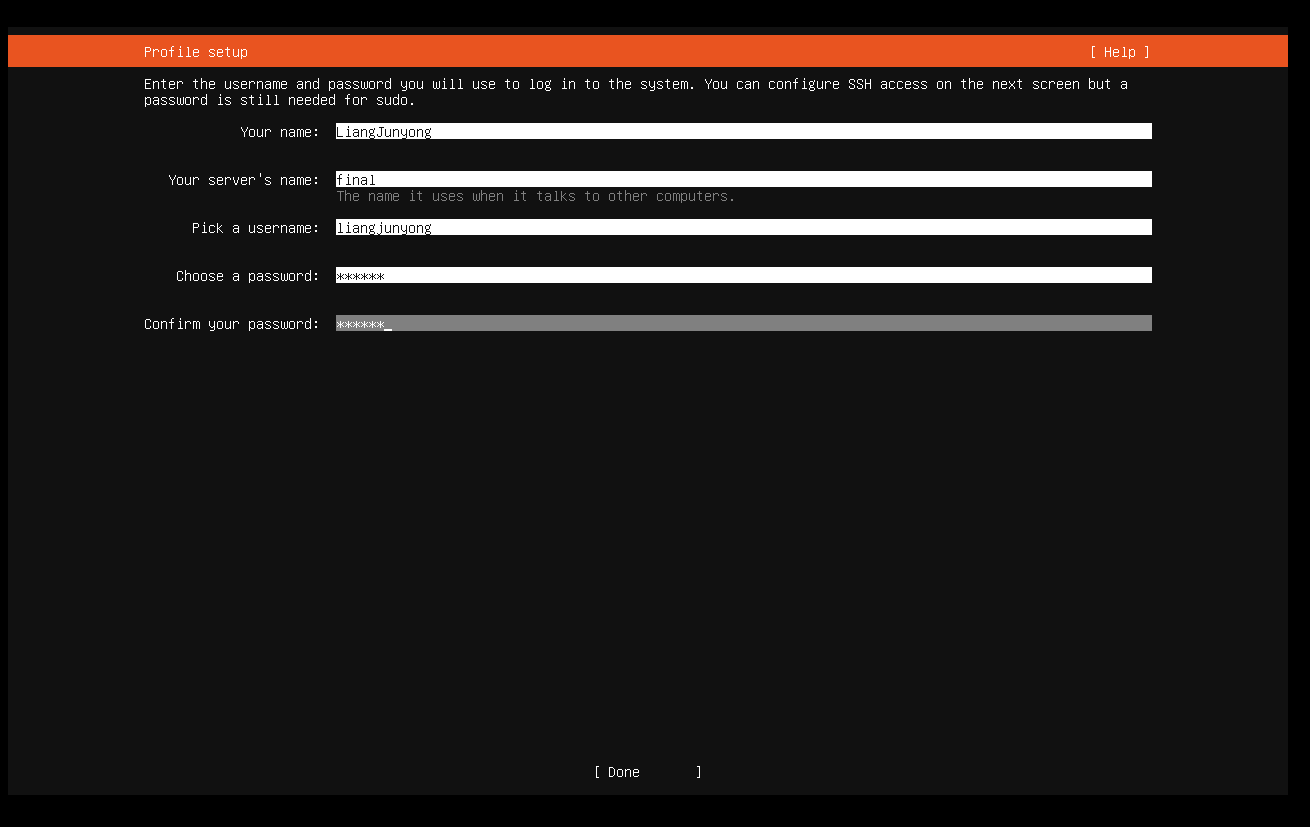
使用vmware创建虚拟机



用学号+姓名缩写命名



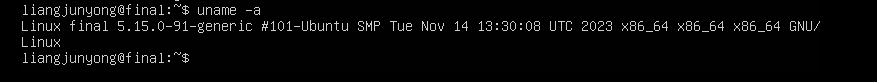
用名字拼音作为用户名



1. 查看内核和操作系统版本

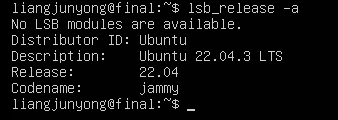
命令：uname -a 可以查看内核版本

截图：



命令：lsb\_release -a 可以查看发行版发行版本

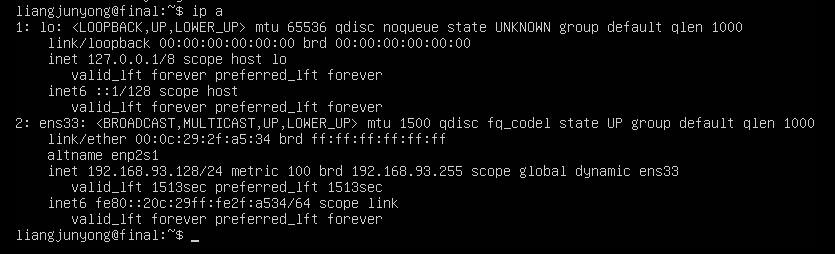
截图：



1. 查看MAC地址，硬件地址

命令：ip a

截图：



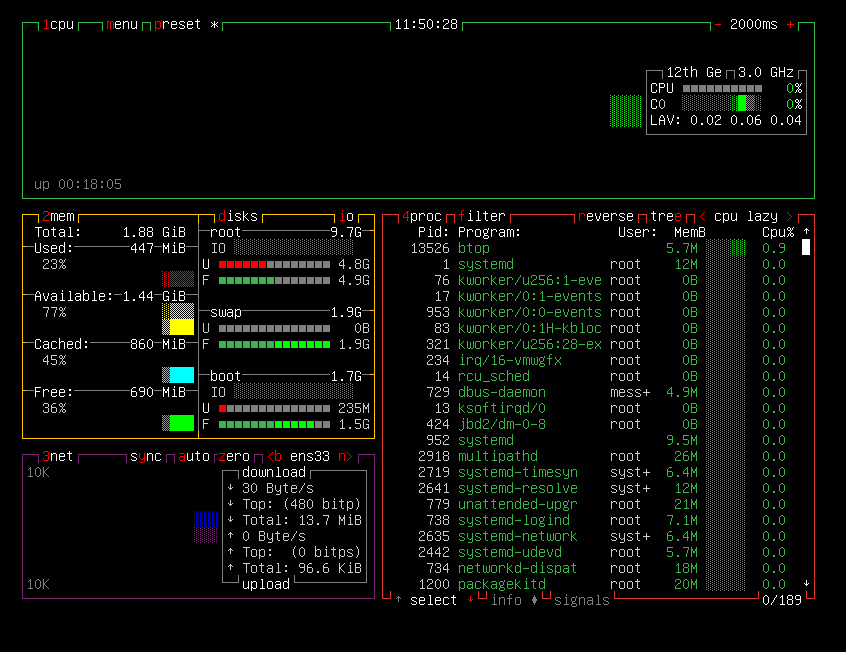
1. 查看系统的后台任务

命令：btop

使用btop进行查看

默认没有安装btop,因此需要使用sudo apt install btop进行安装

截图：

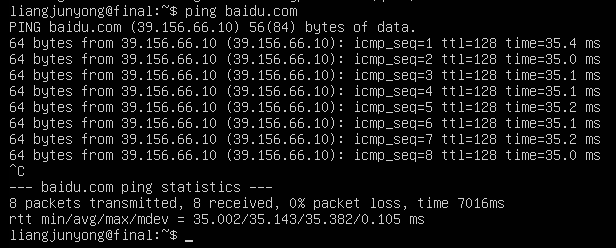


1. 测试当前主机是否能与百度网站进行联通

命令：ping baidu.com

默认没有ping工具, 使用sudo apt install iputils-ping进行安装

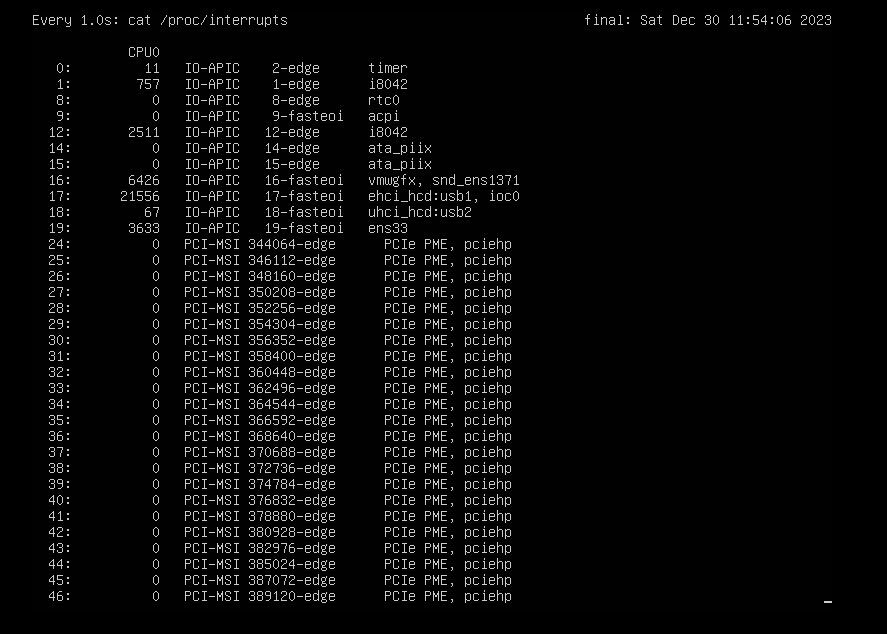
截图：



1. 如何查看系统中已注册的中断及访问频次（1秒更新一次）。

命令：watch -n 1 cat /proc/interrupts

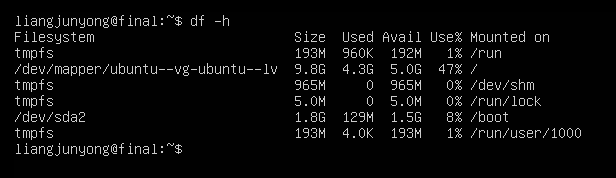
截图：



1. 使用什么命令查看磁盘使用空间？空闲空间呢?

命令：df -h可以查看使用空间和空闲空间

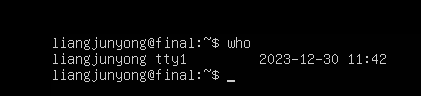
截图：



1. 查看当前谁在使用该主机用什么命令?查找自己所在的终端信息用什么命令?

命令：who命令可以查看所有登录用户,登录的终端,登录的时间

截图：



命令：tty命令可以看到目前终端的设备信息

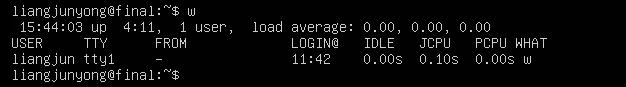
截图：



命令：w

也可以用w命令

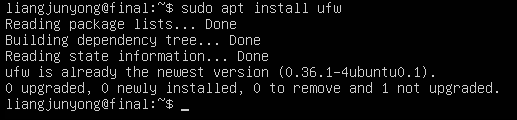
截图：



1. 如何让系统启动时自动开启防火墙，并且允许IP段192.168.0.0/24访问端口20。

命令：sudo apt install ufw

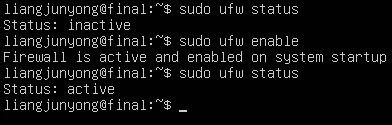
准备使用ufw作为防火墙, 先进行安装



命令：sudo ufw enable 可以启用防火墙并在开机时自己启动

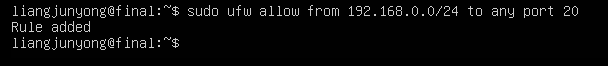
命令：sudo ufw status 可以查看防火墙当前状态

截图：



命令：sudo ufw allow from 192.168.0.0/24 to any port 20

截图：

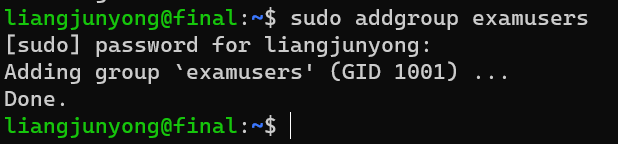


（二） 基本操作

(1)

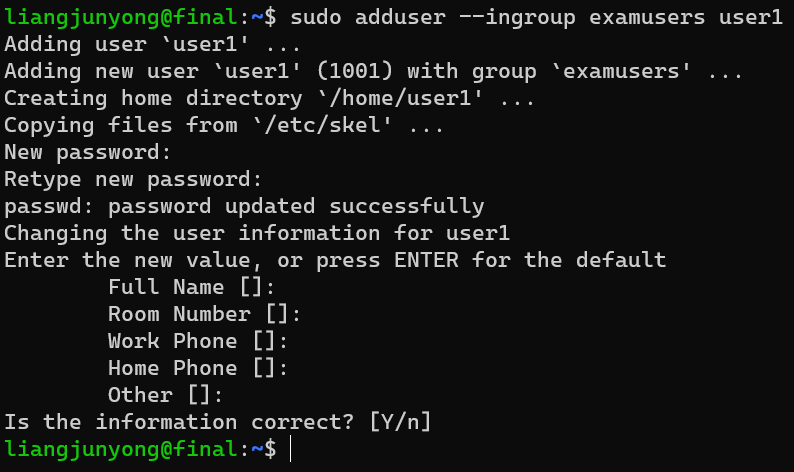
命令：sudo addgroup examusers 创建用户组

截图：



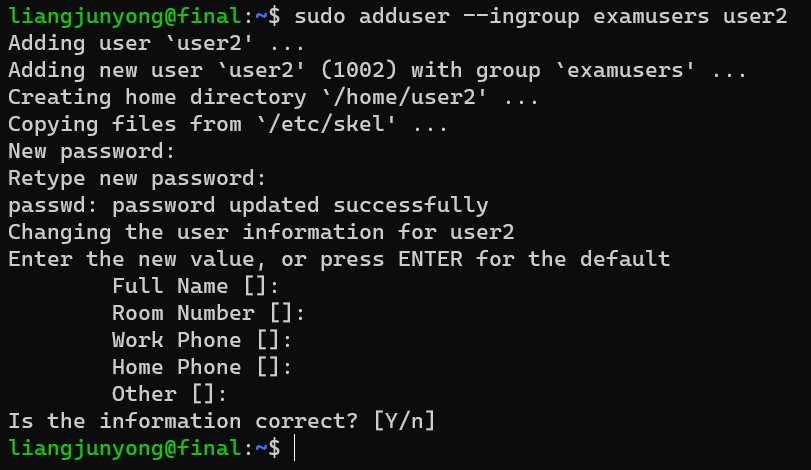
命令：sudo adduser --ingroup examusers user1

截图：



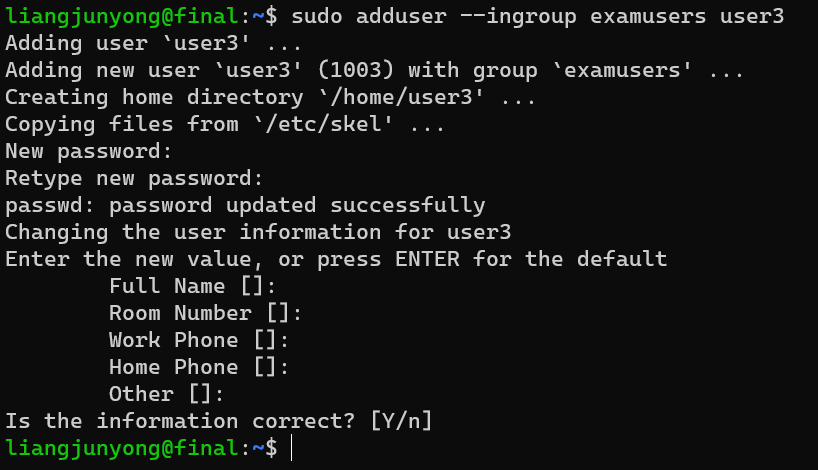
命令：sudo adduser --ingroup examusers user2

截图：



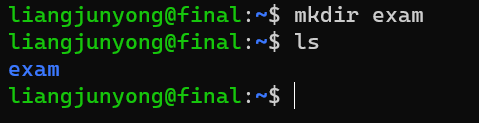
命令：sudo adduser --ingroup examusers user3

截图：



命令：mkdir exam

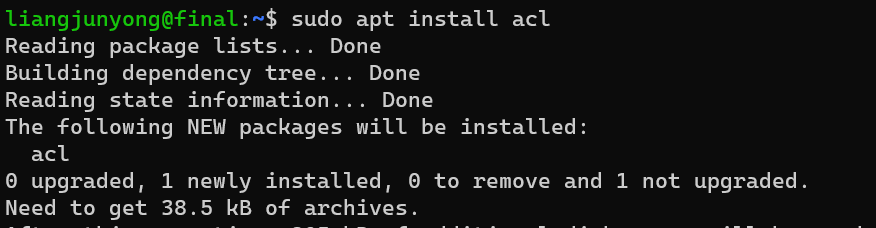
截图：



使用acl对权限进行管理先安装

命令：sudo apt install acl

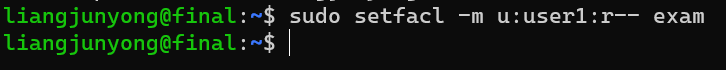
截图：



user1设置只读

命令：sudo setfacl -m u:user1:r-- exam

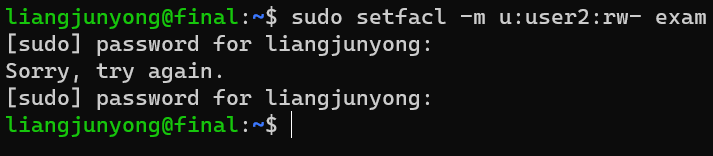
截图：



user2设置读写

命令：sudo setfacl -m u:user2:rw- exam

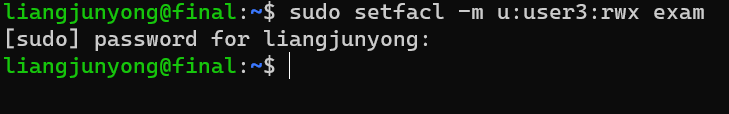
截图：



user3 设置完全权限

命令：sudo setfacl -m u:user3:rwx exam

截图：

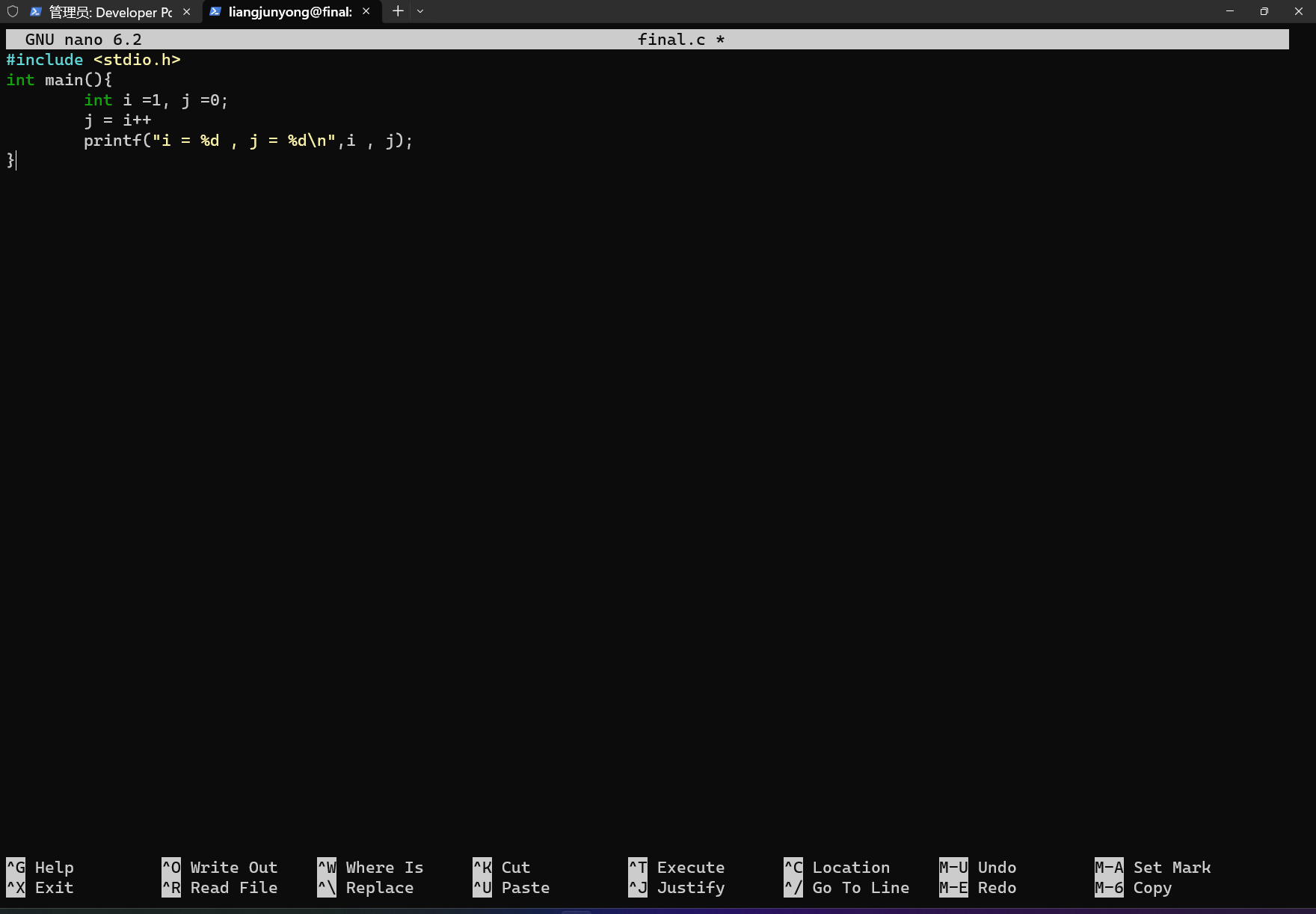


(2)

创建一个叫final.c的源码文件

命令：nano final.c

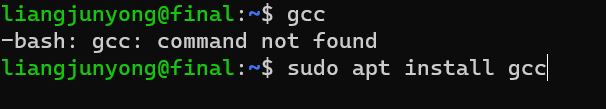
截图：



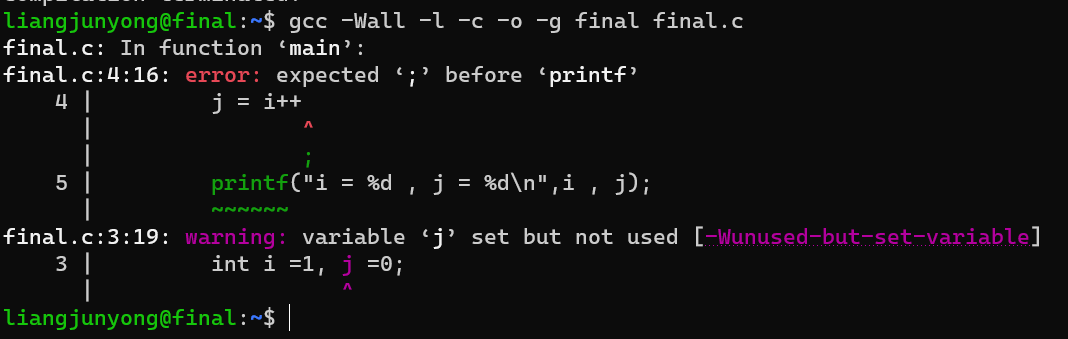
安装gcc

命令：sudo apt install gcc

截图：

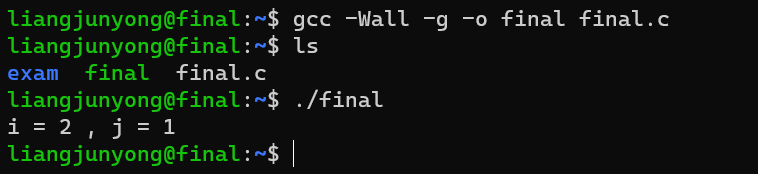


使用提供的参数进行编译，发现存在语法错误，修正一下



成功编译后运行

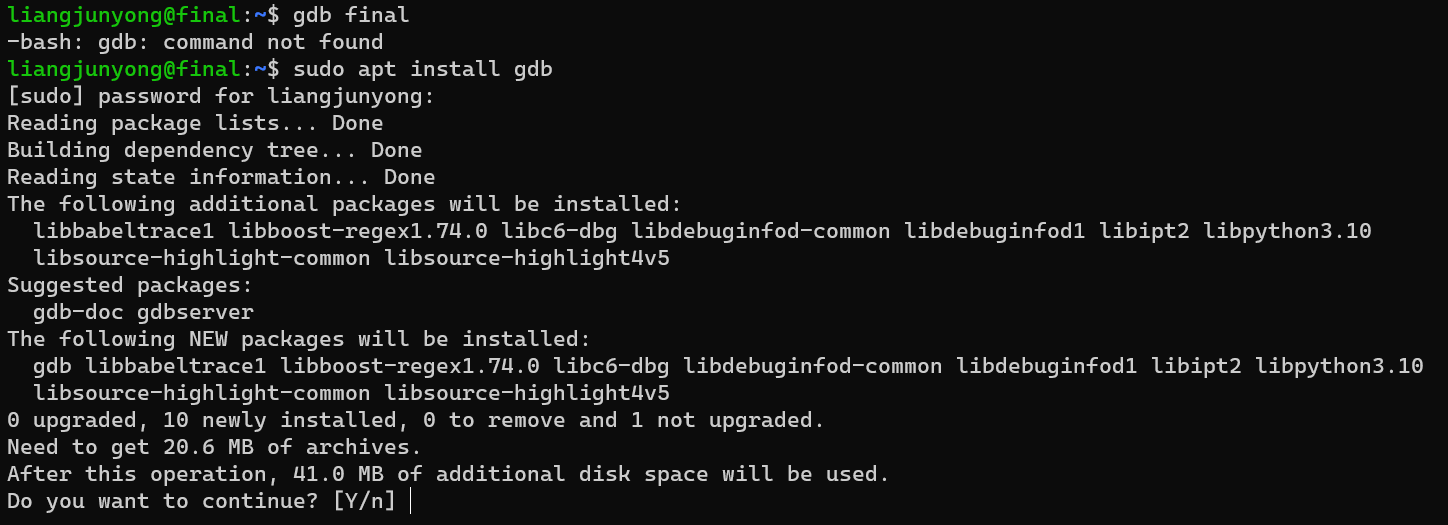
命令：gcc -Wall -g -o final final.c



安装并启动gdb

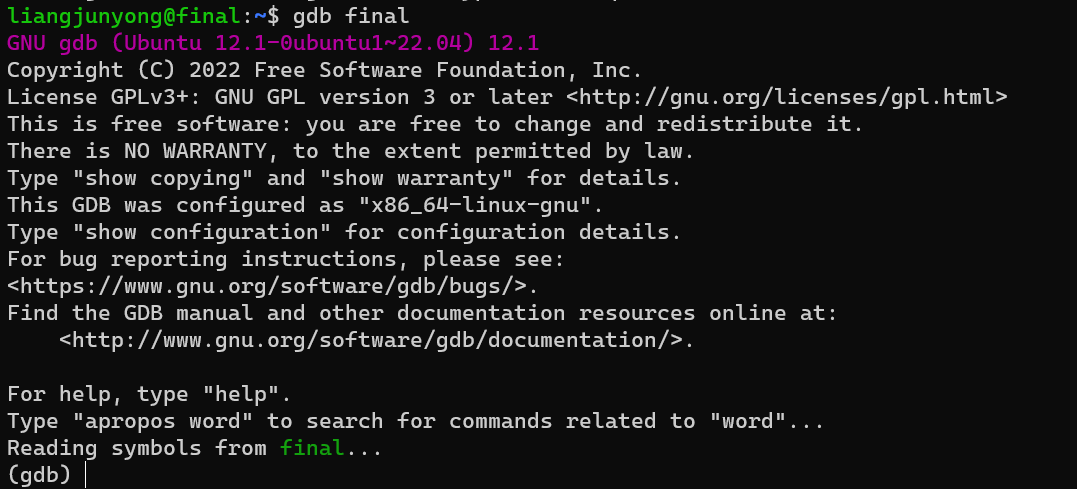
命令： sudo apt install gdb

截图：



命令： gdb final

截图：



设置断点

在main函数入口设置断点

命令： b main

截图



在第4行设置断点

命令： b 4

截图：

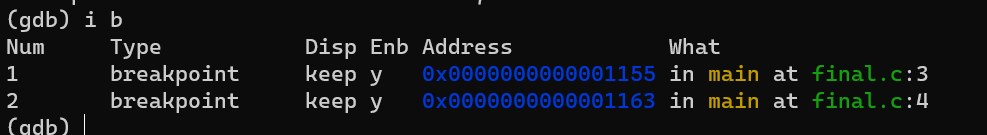


清除断点

先查看所有断点

命令： i b

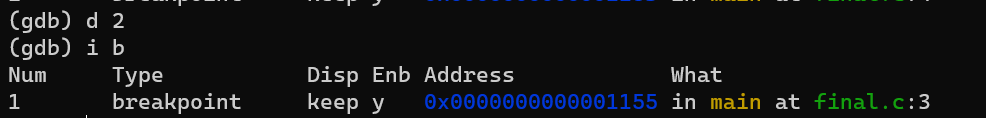
截图



去除第二个断点

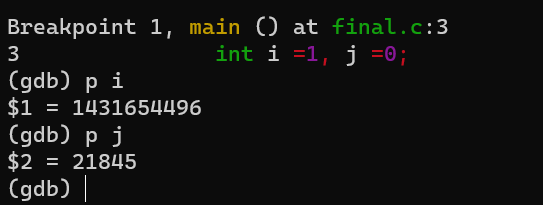
命令：d 2

截图：

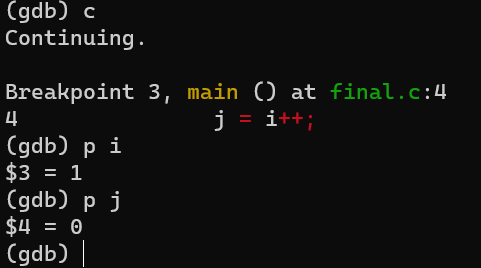


显示程序变量值

断点在main入口处，因此两个变量还未初始化

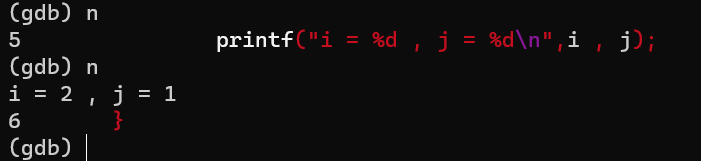


继续执行，在第四行断点，此时两个变量已经被赋值



单步执行。

利用单步执行，执行print函数，输出了j=i++之后的结果



（三）进阶操作

（1）

算法思路:

请求内存

当系统收到一个内存分配请求时，Buddy算法按以下步骤操作：

1.计算块大小：首先，算法计算出满足请求所需的最小块大小，这个大小是2的幂次方（例如，请求为35字节，最小块大小将是64字节，即2的6次方）。

2.寻找合适的块：接着，算法在内存的空闲块列表中查找第一个足够大的块。如果找到的块大小恰好等于计算出的最小块大小，则直接分配这个块。

3.分割块：如果找到的块比需要的大，则将这个块分割成两个更小的块（每个块大小是原来的一半），然后重复这个过程，直到得到一个大小合适的块。

4.标记和返回地址：分配适当大小的块后，标记这个块为已分配，并返回其起始地址。

释放内存

释放内存时，Buddy算法执行以下步骤：

1.寻找伙伴块：然后，算法寻找刚释放块的“伙伴”块。两个伙伴块大小相同，地址相邻，并且在二进制表示中只有一位不同。

2.合并伙伴块：如果伙伴块也是空闲的，则将这两个块合并成一个更大的块。然后，算法继续检查合并后的块是否有空闲的伙伴块，如有则继续合并。这个过程一直进行，直到没有可合并的伙伴块。

3.标记块为可用：将要释放的内存块标记为可用。

4.更新空闲块列表：最后，更新空闲块列表，反映合并后的空闲块的状态。

代码

class BuddyAllocator:

    def \_\_init\_\_(self, size):

        if not self.is\_power\_of\_two(size):

            raise ValueError("Size must be power of two")

        self.size = self.get\_power\_of\_two(size)

        self.free\_list = {self.size: [0]}  # 使用2的幂作为键

        self.allocated\_list = {}  # 记录分配的地址

    def is\_power\_of\_two(self, n):

        if n <= 0:

            return False

        return (n & (n - 1)) == 0

    def get\_power\_of\_two(self, size):

        # 找到不小于size的最小2的幂

        power = 1

        while power < size:

            power \*= 2

        return power

    def allocate(self, size):

        # 分配内存函数

        size = self.get\_power\_of\_two(size)  # 调整size为2的幂

        # 从小到大寻找合适的block size

        for block\_size in sorted(self.free\_list):

            if block\_size >= size:

                # 取出第一个符合大小的block的首地址

                address = self.free\_list[block\_size].pop(0)

                if len(self.free\_list[block\_size]) == 0:

                    del self.free\_list[block\_size]  # 移除空列表

                # 确定了首地址和size, 对剩余内存进行拆分

                self.split(block\_size, address, size)

                self.allocated\_list[address] = size

                self.show\_mem()

                return address

        raise ValueError("Can't allocate")

    def split(self, block\_size, address, size):

        # 递归拆分

        while block\_size > size:

            block\_size //= 2

            # 计算出buddy的首地址

            buddy\_address = address + block\_size

            self.free\_list.setdefault(block\_size, []).append(buddy\_address)

    def free(self, address, size):

        # 释放内存

        size = self.get\_power\_of\_two(size)  # 调整size为2的幂

        # Do double free check

        if address not in self.allocated\_list:

            raise ValueError("Can't double free")

        buddy\_address = self.find\_buddy(address, size)

        # 如果buddy空闲,则合并

        if size in self.free\_list and buddy\_address in self.free\_list[size]:

            self.free\_list[size].remove(buddy\_address)

            if len(self.free\_list[size]) == 0:

                del self.free\_list[size]  # 移除空列表

            # 向左或向右合并buddy

            merged\_address = min(address, buddy\_address)

            # 尝试递归合并

            self.free(merged\_address, size \* 2)

        else:

            # buddy不空闲则把目前的释放

            self.free\_list.setdefault(size, []).append(address)

            del self.allocated\_list[address]  # 从记录中移除

            self.show\_mem()

    def find\_buddy(self, address, size):

        # 因为二分得到的buddy在地址上只有一位的差别,

        # 因此用异或操作可以直接计算出buddy的首地址

        return address ^ size

    def show\_mem(self):

        mem\_visual = ["-" for \_ in range(self.size)]  # 初始化全部为可用

        # 标记已分配的内存

        for address, size in self.allocated\_list.items():

            for i in range(address, address + size):

                mem\_visual[i] = "X"

        # 将内存可视化为字符串

        mem\_str = "".join(mem\_visual)

        # 打印可视化的内存

        print("Memory Visualization:")

        print(mem\_str)

if \_\_name\_\_ == "\_\_main\_\_":

    # 创建分配器

    buddy = BuddyAllocator(64)

    proc\_a = buddy.allocate(8)

    print(f"Allocated at {proc\_a}")

    print()

    proc\_b = buddy.allocate(16)

    print(f"Allocated at {proc\_b}")

    print()

    proc\_c = buddy.allocate(32)

    print(f"Allocated at {proc\_c}")

    print()

    buddy.free(proc\_b, 16)

    print(f"freed {proc\_b}")

    print()

    proc\_d = buddy.allocate(8)

    print(f"Allocated at {proc\_d}")

print()

测试代码

import unittest

from main import \*

class TestBuddyAllocator(unittest.TestCase):

    def test\_initialization(self):

        size = 1024

        buddy = BuddyAllocator(size)

        self.assertEqual(buddy.size, size)

    def test\_allocate\_and\_free(self):

        buddy = BuddyAllocator(1024)

        addr = buddy.allocate(100)

        self.assertIsNotNone(addr)

        buddy.free(addr, 100)

        self.assertIn(addr, buddy.free\_list[1024])

        addr = buddy.allocate(200)

        self.assertIsNotNone(addr)

        buddy.free(addr, 200)

        self.assertIn(addr, buddy.free\_list[1024])

    def test\_invalid\_initialization(self):

        with self.assertRaises(ValueError):

            BuddyAllocator(1000)  # 非2的幂次方

    def test\_allocation\_failure(self):

        buddy = BuddyAllocator(1024)

        with self.assertRaises(ValueError):

            buddy.allocate(2048)  # 请求更多内存

    def test\_double\_free(self):

        buddy = BuddyAllocator(1024)

        addr = buddy.allocate(100)

        buddy.free(addr, 100)

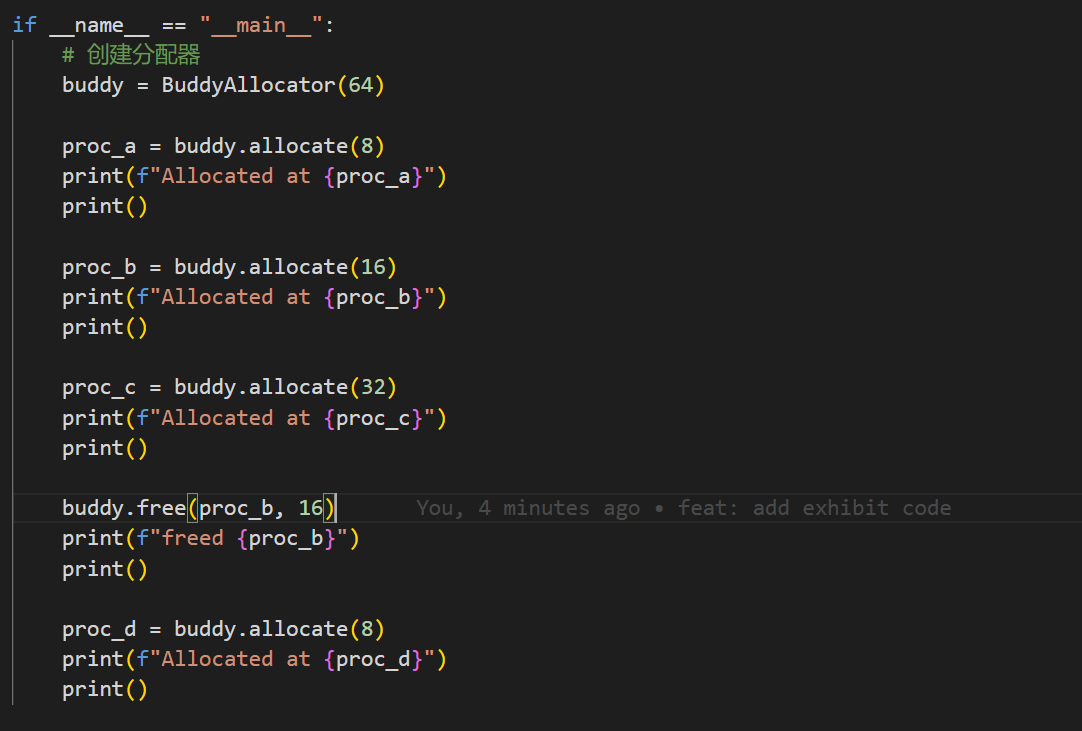
        with self.assertRaises(ValueError):

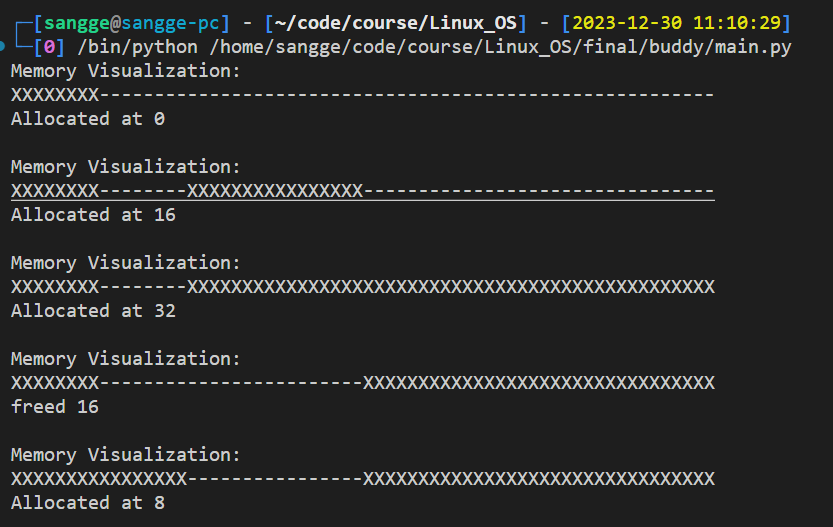
            buddy.free(addr, 100)  # 二次释放同一个地址

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

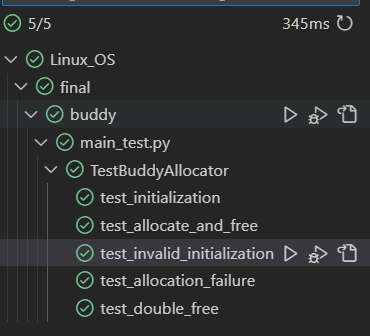
运行截图:





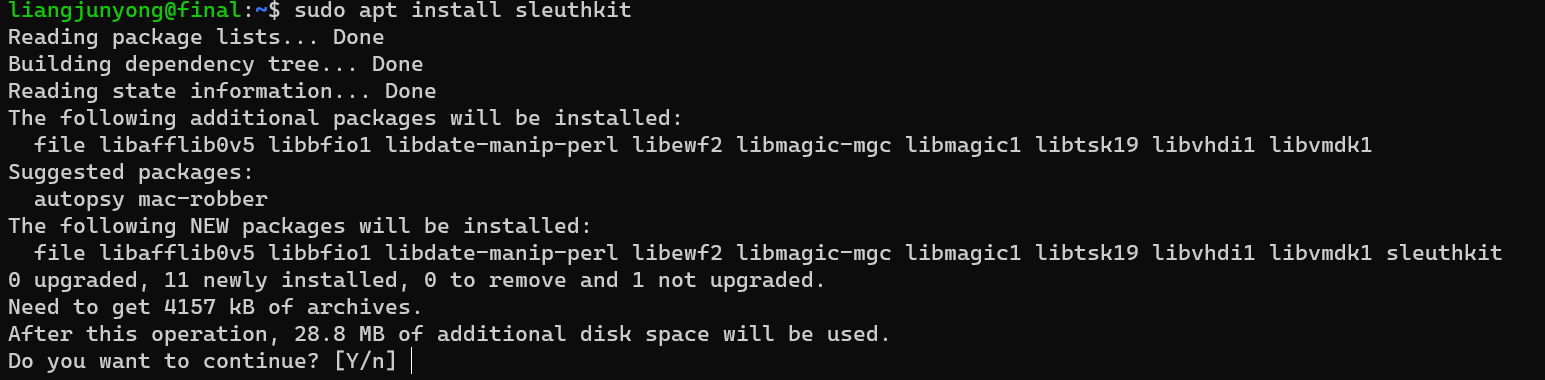
其中x代表已经分配的内存,-表示空闲的内存

单元测试全部通过

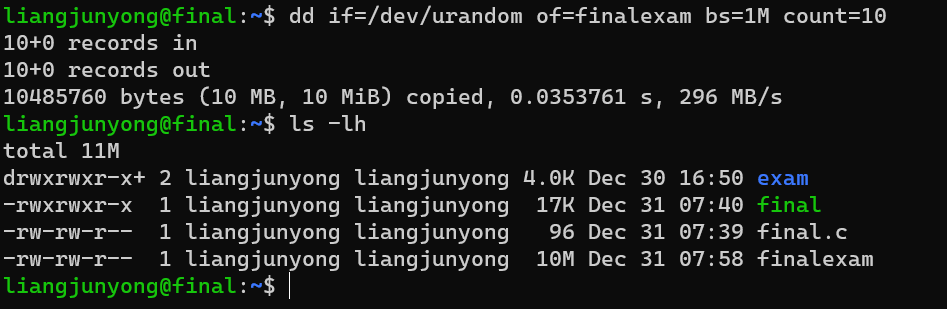


（2）

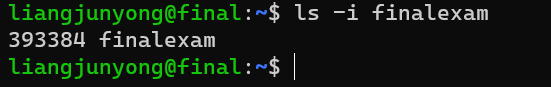
需要先安装sleuthkit，不然fsstat和blkcat命令用不了。



使用dd创建一个10MB的文件,文件内容就从随机生成器里面读取

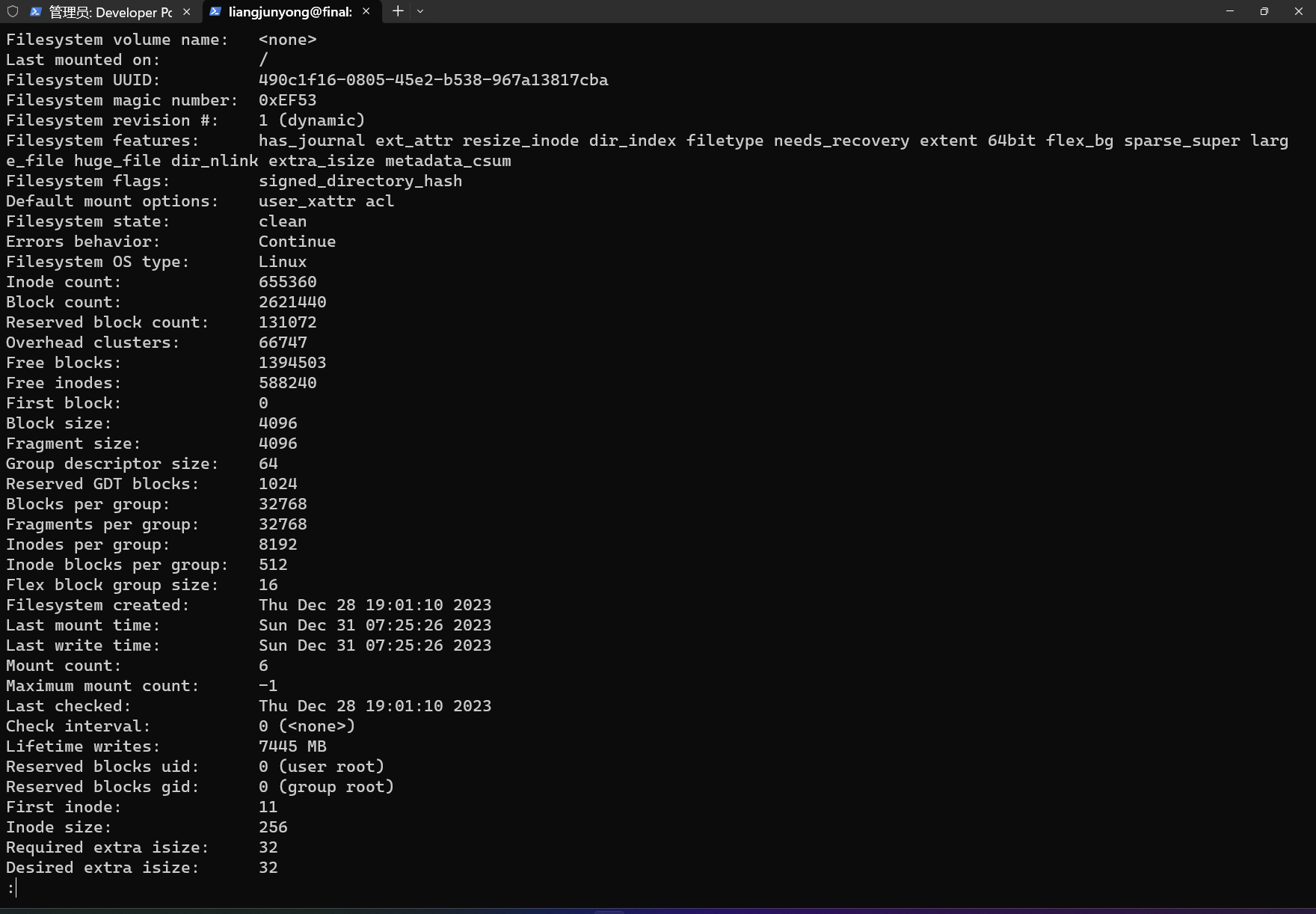


获取文件的inode号,393384



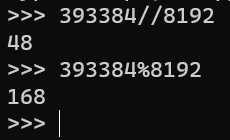
要定位 inode table，需要知道 ext4 文件系统的一些参数，如每个块的大小、每组的 inode 数等。可以使用 dumpe2fs 命令来获取这些信息



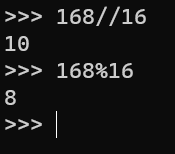


在图中可以看到inodes per group 是8192， inode block per group 是512

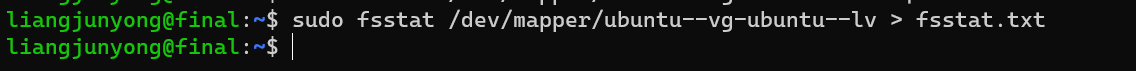
使用inode号计算出块组号和块组内的偏移量



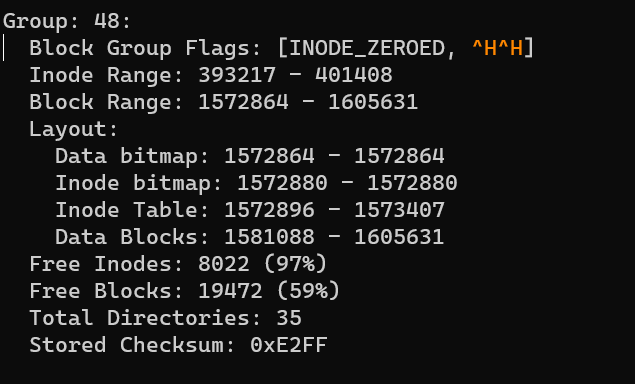
根据块组内的偏移量计算出本地 inode 号和在本地 inode 表中的偏移量



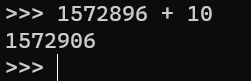
通过fsstat查看inode table



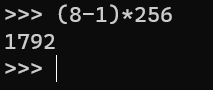
查看group 48



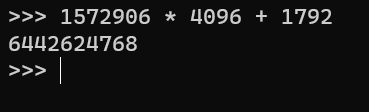
找到块组的开始块号，并加上本地 inode 号，来得到 inode table 的确切块号



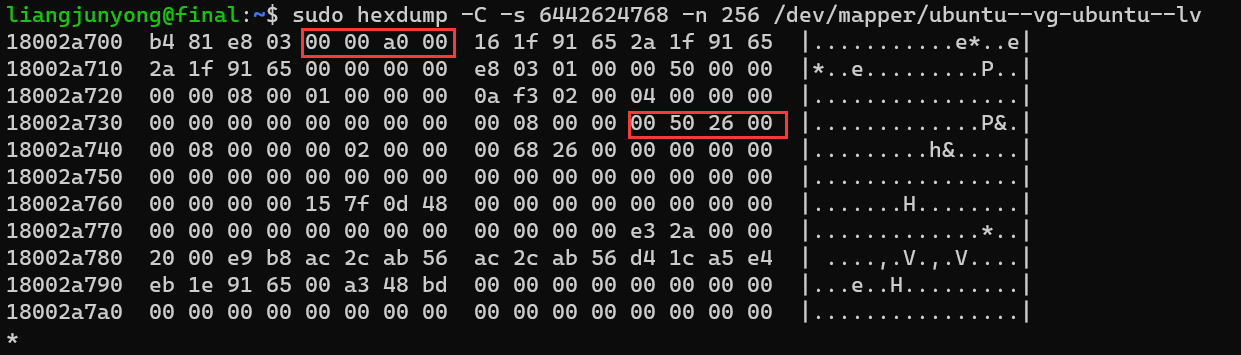
使用 inode 在 inode table 中的偏移量乘以 inode 的大小，得到在 inode table 块内的字节偏移量



然后计算文件的 inode 在磁盘上的确切位置，将 inode table 的块号乘以块的大小，然后加上字节偏移量。



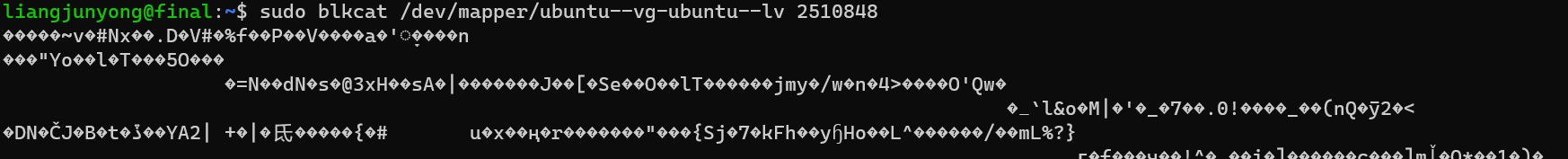
使用 hexdump 从计算出的位置读取 inode 的内容。-s 选项用于输入偏移量，-n 用于读取特定数量的字节

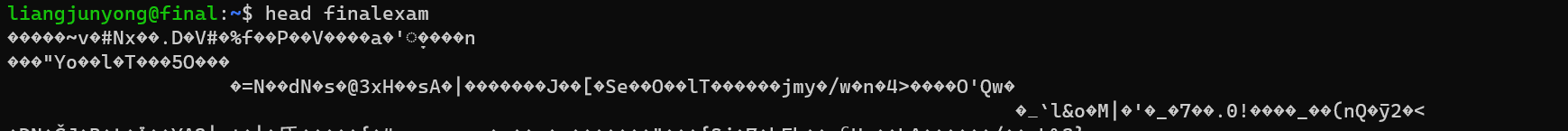


00a00000为文件的大小，00265000为数据块的起始块号









大作业小结

在本次《Linux操作系统》课程的期末大作业中，我得以深入实践和探索Linux的核心功能和特性。首先，我从基础开始，通过手动执行各种Linux命令，加深了对操作系统日常管理和维护工作的理解。这一部分不仅巩固了我的基础知识，还提高了我在实际环境中应用这些命令的能力。

进一步地，我使用gcc工具编译并运行了C语言程序。这个过程不仅让我体验了从源代码到可执行文件的整个过程，还加深了我对编译器如何将高级语言转换成机器代码的理解。接着，我利用gdb对C语言程序进行调试。这一步骤是学习编程中极为重要的一环，它不仅帮助我发现并修复程序中的错误，还让我对程序运行的内部逻辑有了更深的认识。

此外，我探索了内存分配中的伙伴系统（buddy system）算法。为了更好地理解这一高效的内存管理方法，我使用Python编写了一个模拟程序。编写这个程序的过程中，我遇到了多个bug。为了有效解决这些问题，我开始学习并应用单元测试的概念。通过编写和运行针对程序各个小部分的测试用例，我能够确保每个功能模块的正确性。这不仅减少了整体调试的时间，也提高了代码的可靠性。

最后，我重新审视了Linux的存储结构，特别是文件系统和inode的概念。我学习了通过inode号计算文件存储实际地址的方法，这对于理解文件系统的工作原理和数据存储方式是至关重要的。通过这次大作业，我不仅加深了对Linux操作系统的理解，还提升了我的编程技能和问题解决能力。这些学习经历无疑将对我的未来学习和职业生涯产生深远的影响。