**题目4：统计Linux系统缺页的次数**

1. **设计目的与要求**

本设计旨在通过在Linux内核中添加新的系统调用，深入理解操作系统内核的工作机制和系统调用的实现方法。此外，通过实际编程练习，增强对Linux内核编程和环境配置的掌握。

1. **设计内容**

通过在Linux内核中自建变量，并利用/proc文件系统作为中介的方法，统计系统缺页的次数。 要求：

1. 在内核中实现缺页次数统计；
2. 编译并安装新内核；
3. 新建内核模块，并加载到新内核，通过/proc实现用户态下查看缺页次数。
4. **设备与环境**

操作系统：Ubuntu 20.04

虚拟化软件：VMware Workstation 17 Pro

开发环境：Visual Studio Code

内核版本：5.6.8

1. **设计思想**

设计的主要思想是通过直接修改Linux内核源代码来添加自定义的系统调用，实现对核心功能的扩展。主要包括以下步骤：

1.获取当前Linux内核的源代码。

2.修改内核源代码以添加新的系统调用定义和实现。

3.修改系统调用表和相关头文件，注册新的系统调用。

4.重新编译和加载内核，对新增的系统调用进行测试。

1. **主要数据结构和流程**

1.下载内核源码并解压

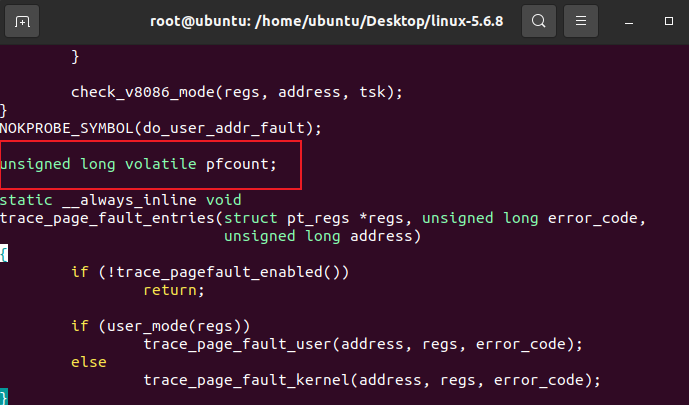
需要修改以下文件

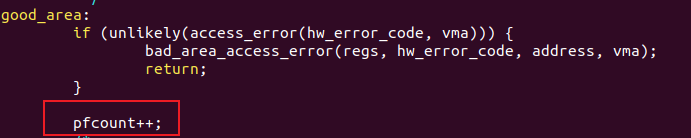
arch/x86/mm/fault.c

include/linux/mm.h

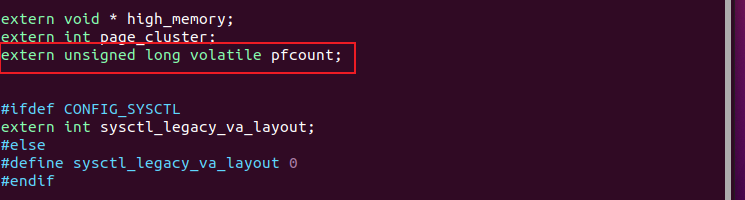
kernel/kallsyms.c

2.修改 arch/x86/mm/fault.c，定义 pfcount 变量

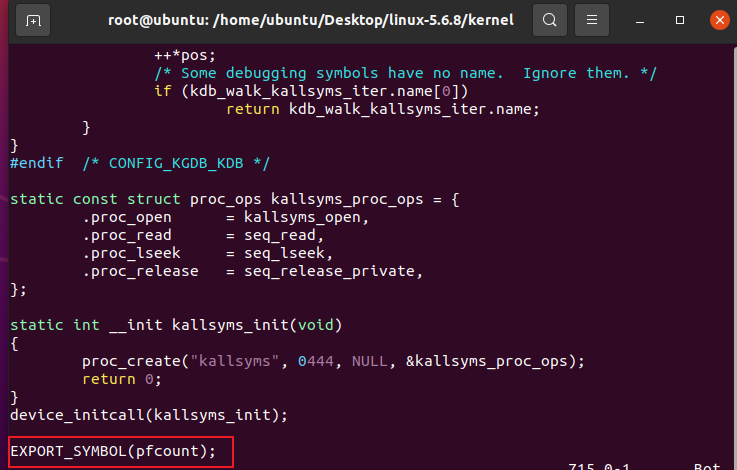




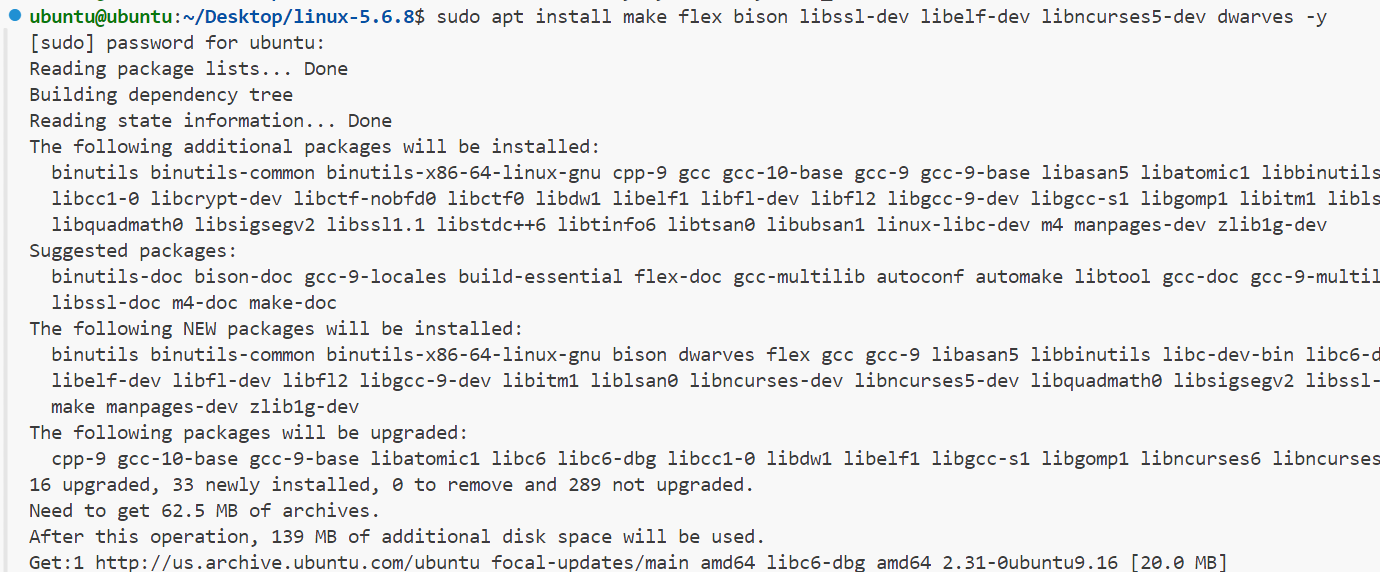
3.修改 include/linux/mm.h



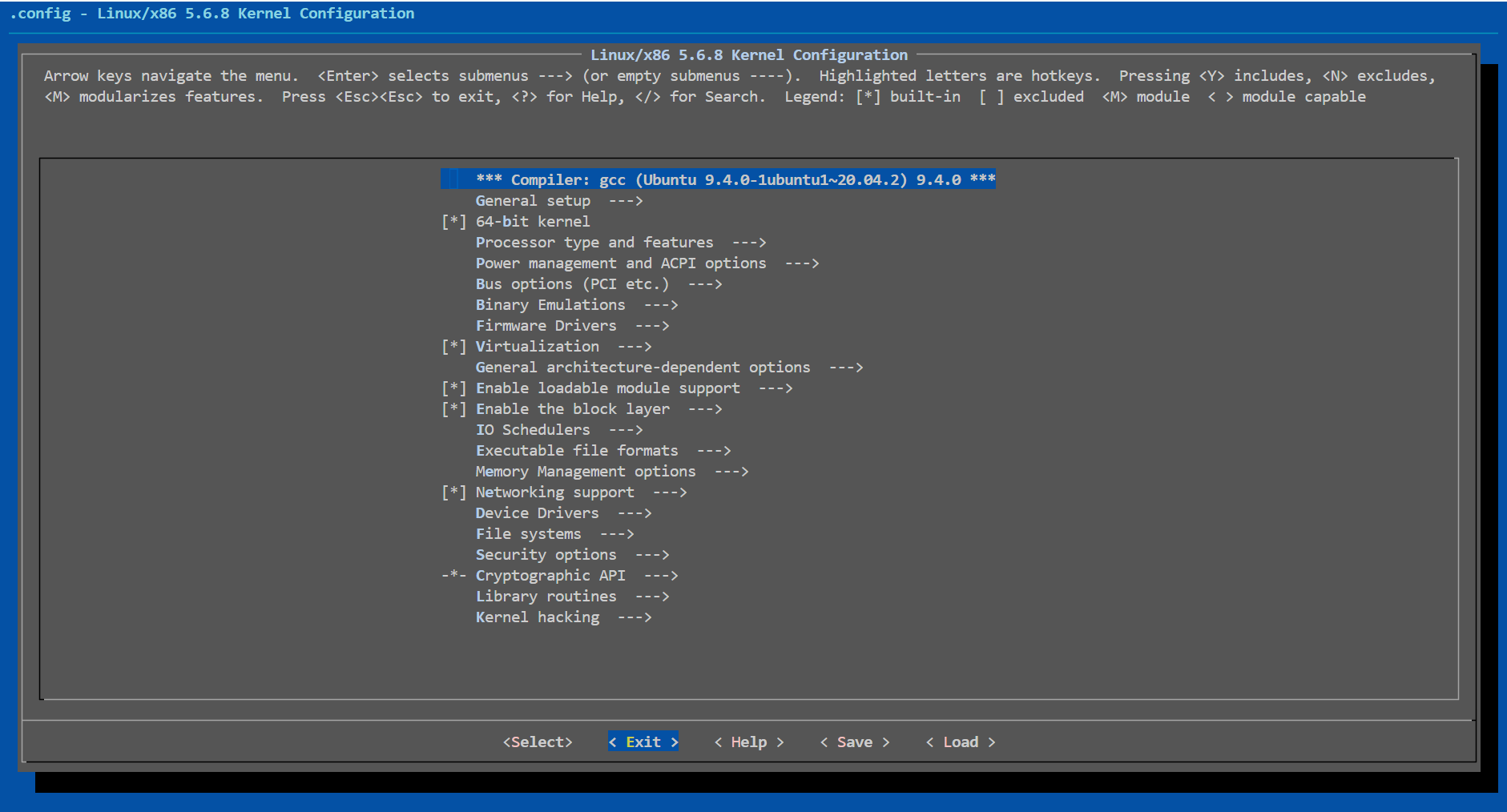
1. 修改 kernel/kallsyms.c，暴露pfcount变量。注意，修改内核文件需要root用户权限。



5.安装编译内核所需的依赖包



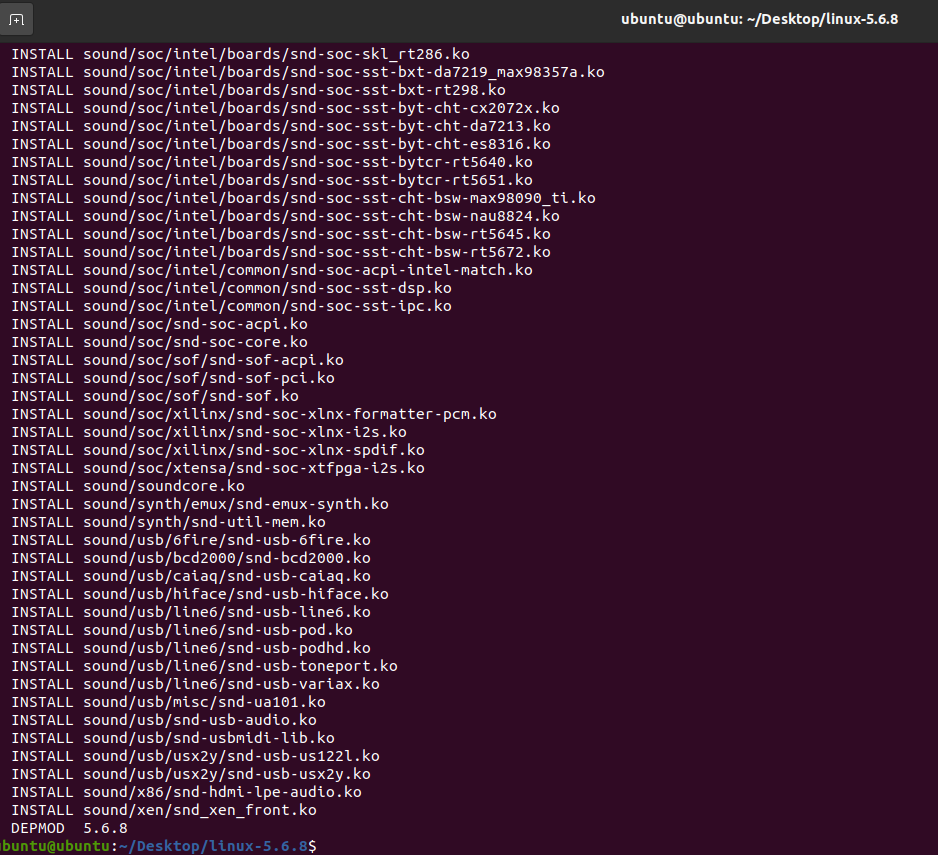
6. 配置内核，输入**make menuconfig**，在跳出的界面中进行配置



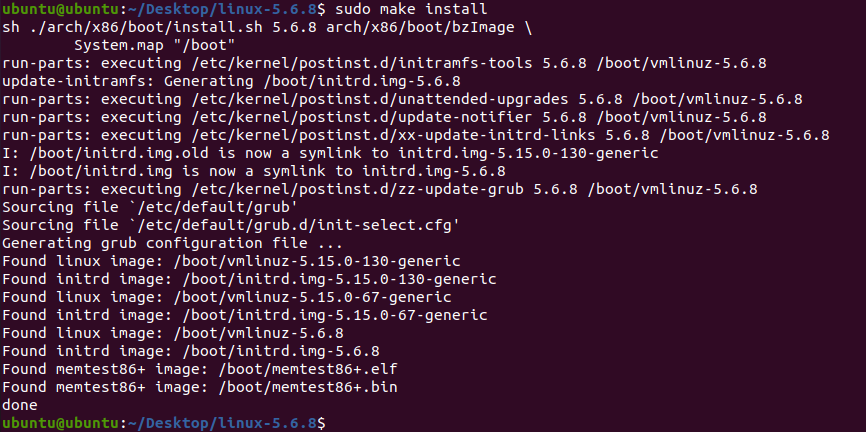
7.编译，查看自己机器处理器的核数，中键入 **make -jn**（n个线程开始编译）



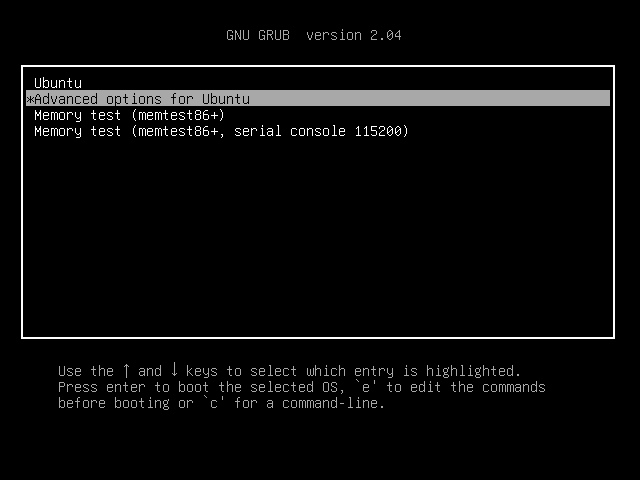
1. 安装编译好的模块，输入sudo make modules\_install 命令，完成模块的安装操作

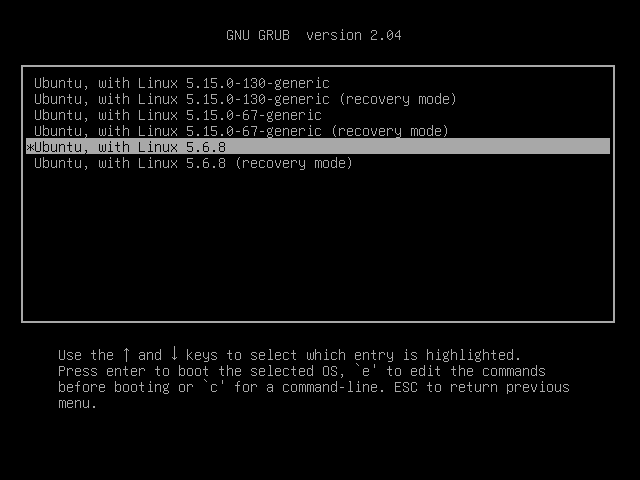


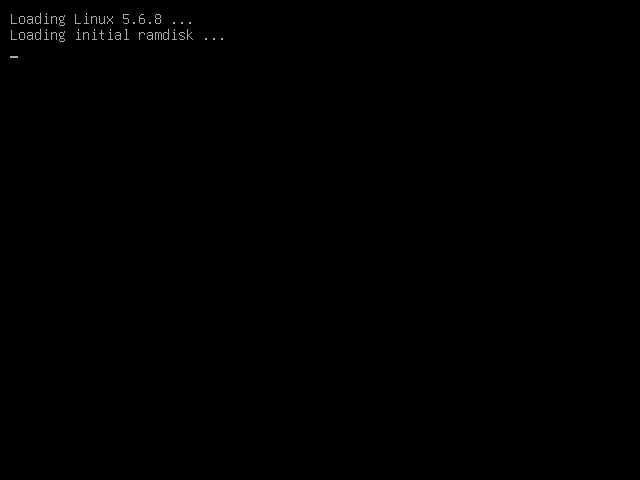
9. 输入 sudo make install安装内核

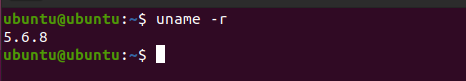


10.重启，内核切换成功









1. **实验测试结果及结果分析**

1、编写读取pfcount模块

#include <linux/module.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

#include <linux/proc\_fs.h>

#include <linux/fs.h>

#include <linux/mm.h>

#include <linux/seq\_file.h>

#include <linux/slab.h>

#include <linux/kernel.h>

#include <linux/timer.h>

#include <linux/jiffies.h>

#include <stdarg.h>

*// 引入外部全局变量 pfcount，用于记录页面缺页中断次数*

extern unsigned long volatile pfcount;

*// 定义全局变量*

static struct timer\_list test\_timer; *// 定时器结构体*

static unsigned long pfcount\_last; *// 上一次的页面缺页中断计数*

static unsigned long pfcount\_in\_2; *// 两秒内的页面缺页中断计数*

static int count = 0; *// 用于记录定时器触发的次数*

*// 模块声明*

MODULE\_LICENSE("GPL");

*/\**

*\* 5. show 函数*

*\* 用于将内核数据输出到用户空间。*

*\* 当访问对应的 /proc 文件时会调用该函数。*

*\*/*

static int my\_proc\_show(struct seq\_file \**m*, void \**v*)

{

*/\**

*\* 注意：在这里不能使用 printk 等函数。*

*\* 使用 seq\_file 系列函数向用户空间输出内容。*

*\*/*

    seq\_printf(m, "[latest] Number of page fault interrupts in 2 seconds: %ld !\n", pfcount\_in\_2);

    return 0;

}

*/\**

*\* 定时器的回调函数*

*\* 每两秒触发一次，计算页面缺页中断的数量，并打印日志。*

*\*/*

static void irq\_test\_timer\_function(struct timer\_list \**timer*)

{

*// 打印日志信息，记录两秒内页面缺页中断数量*

    printk("%d Number of page fault interrupts in 2 seconds: %ld\n", count, pfcount - pfcount\_last);

*// 计算两秒内的页面缺页中断数*

    pfcount\_in\_2 = pfcount - pfcount\_last;

*// 更新上一次的页面缺页中断计数*

    pfcount\_last = pfcount;

*// 重新设置定时器，延迟两秒后再次触发*

    mod\_timer(&test\_timer, jiffies + 2 \* HZ);

    count++;

}

*/\**

*\* 打开 proc 文件时调用的函数*

*\* 绑定 seq\_show 函数指针，用于显示数据*

*\*/*

static int my\_proc\_open(struct inode \**inode*, struct file \**file*)

{

*/\**

*\* 使用 single\_open 函数绑定 seq\_show 函数。*

*\* single\_open 是内核提供的封装函数。*

*\*/*

    return single\_open(file, my\_proc\_show, NULL);

}

*/\**

*\* 2. 定义 proc 文件操作结构体*

*\* 填充 proc\_create 函数中使用的 file\_operations 结构体。*

*\* my\_proc\_\* 为自定义函数，seq 和 single 为内核已实现的函数。*

*\* open 为必须实现的函数。*

*\*/*

static struct proc\_ops my\_fops = {

    .proc\_open = my\_proc\_open, *// 打开文件时的回调函数*

    .proc\_read = seq\_read, *// 读取文件内容*

    .proc\_lseek = seq\_lseek, *// 文件指针的偏移操作*

    .proc\_release = single\_release, *// 关闭文件*

};

*/\**

*\* 3. 模块加载函数*

*\* 初始化时创建 /proc 文件和定时器*

*\*/*

static int \_\_init my\_init(void)

{

    struct proc\_dir\_entry \*file; *// /proc 文件指针*

    struct proc\_dir\_entry \*parent; *// 父级目录指针*

*// 在 /proc 下创建目录，这里用学号作为目录名*

    parent = proc\_mkdir("3190608027", NULL);

*/\**

*\* 使用 proc\_create 创建 /proc 文件，并绑定 file\_operations*

*\* 参数说明：*

*\* 参数1：文件名*

*\* 参数2：文件权限*

*\* 参数3：父目录指针，如果为 NULL，则在 /proc 下创建*

*\* 参数4：绑定的 file\_operations 结构体*

*\*/*

    file = proc\_create("readpfcount", 0644, parent, &my\_fops);

    if (!file)

        return -ENOMEM;

*// 初始化定时器*

    pfcount\_last = pfcount; *// 初始化上一次的页面缺页计数*

    test\_timer.expires = jiffies + 2 \* HZ; *// 设置初始超时时间为 2 秒*

    timer\_setup(&test\_timer, irq\_test\_timer\_function, 0); *// 设置定时器回调函数*

*// 添加定时器*

    add\_timer(&test\_timer);

    printk(KERN\_INFO "Module initialized and timer added.\n");

    return 0;

}

*/\**

*\* 6. 模块卸载函数*

*\* 卸载时删除 /proc 文件和定时器*

*\*/*

static void \_\_exit my\_exit(void)

{

    printk(KERN\_INFO "Exiting module and removing timer.\n");

*// 删除定时器*

    del\_timer(&test\_timer);

*// 删除 /proc 文件和目录*

    remove\_proc\_entry("readpfcount", NULL); *// 删除文件*

    remove\_proc\_entry("3190608027", NULL); *// 删除目录*

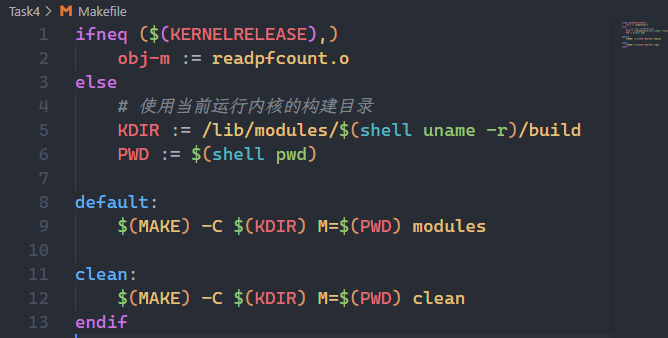
}

*// 注册模块加载和卸载函数*

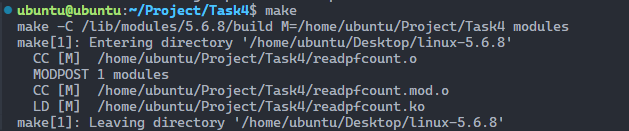
module\_init(my\_init);

module\_exit(my\_exit);

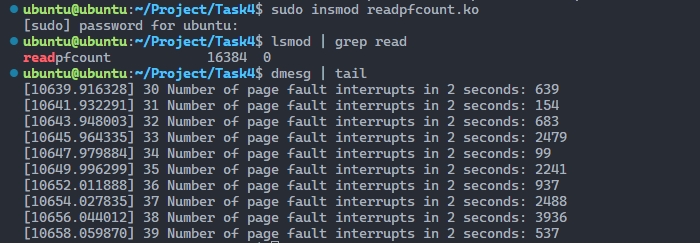
2、Makefile



1. Make



1. 安装模块



1. 使用 cat /proc/readpfcount查看缺员次数

