操作系统

实验报告

|  |  |
| --- | --- |
| 学生姓名 | 罗健刚 |
| 学 号 | 21312030 |
| 专业班级 | 网络空间安全1班 |
| 学 院 | 网络空间安全学院 |
| 完成时间 | 2023/10/27 |

目录

一、实验目的 ..3

二、实验内容 3

2.1 内容一…………………………………………………………………3

2.2 内容二…………………………………………………………………3

2.3 内容三…………………………………………………………………3

三、实验过程及结果…………………………………………………………………4

3.1 任务一…………………………………………………………………4

3.2 任务二…………………………………………………………………5

3.3 任务三…………………………………………………………………7

3.4 任务四…………………………………………………………………9

3.5 任务五…………………………………………………………………13

3.6 任务六…………………………………………………………………14

四、实验总结…………………………………………………………………………16

1. **实验目的**

(1) 理解 Linux 系统处理系统调用的流程；

(2) 掌握增加与调用系统调用的方法；

(3) 理解 Linux 的内核模块和编译方法；

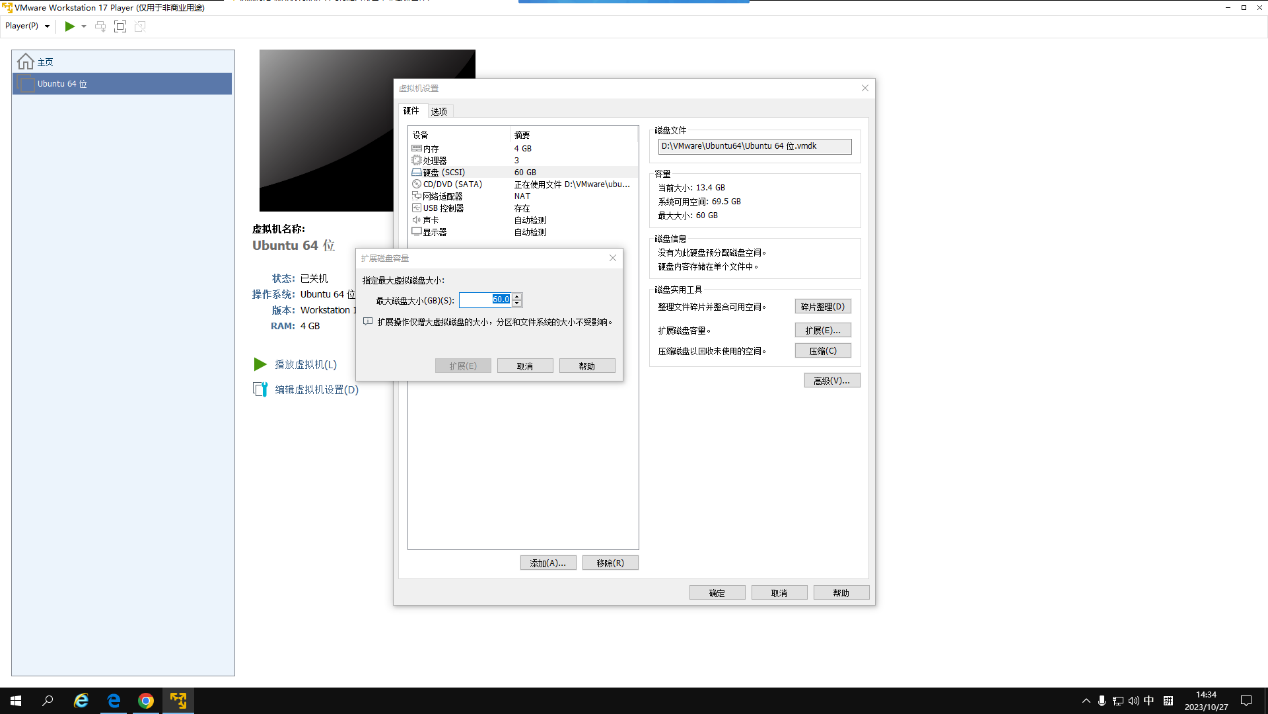
1. **实验内容**

(1) 向现有 Linux 内核加入一个新的系统调用，实现一个新的内核函数 mycall()，这里我们选择借助指针返回调用程序的nice值和优先级prio；

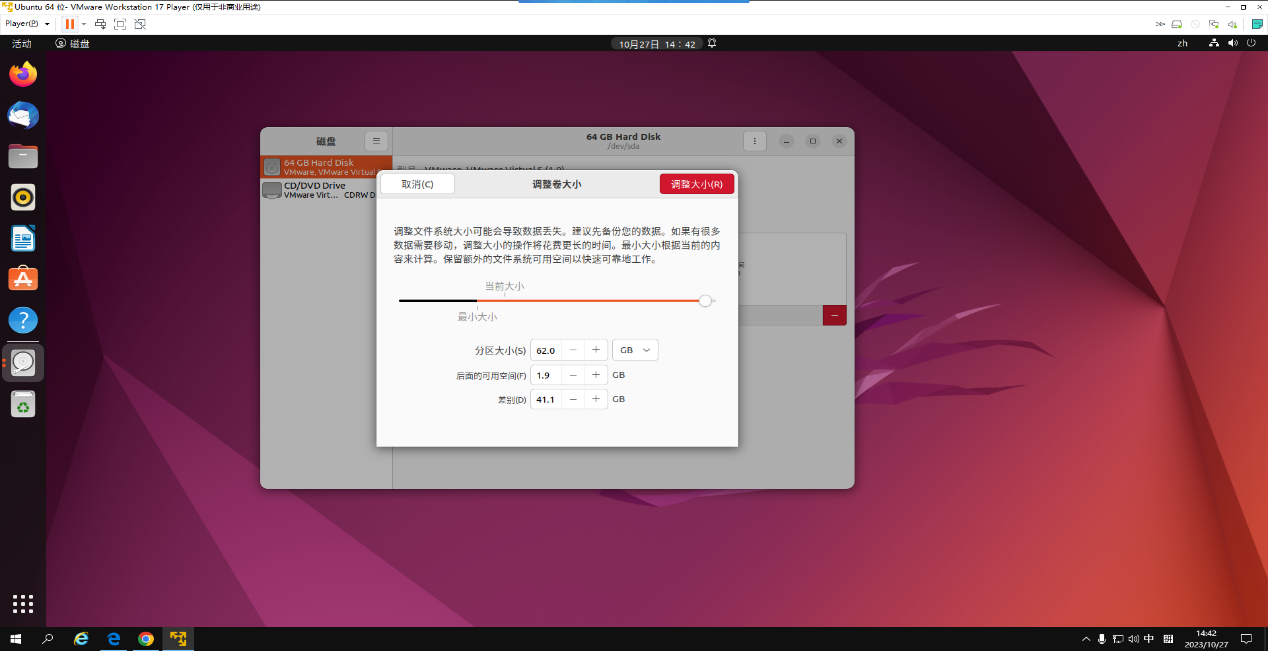
(2) 用编译内核的方法，将其加入内核源码并编译；

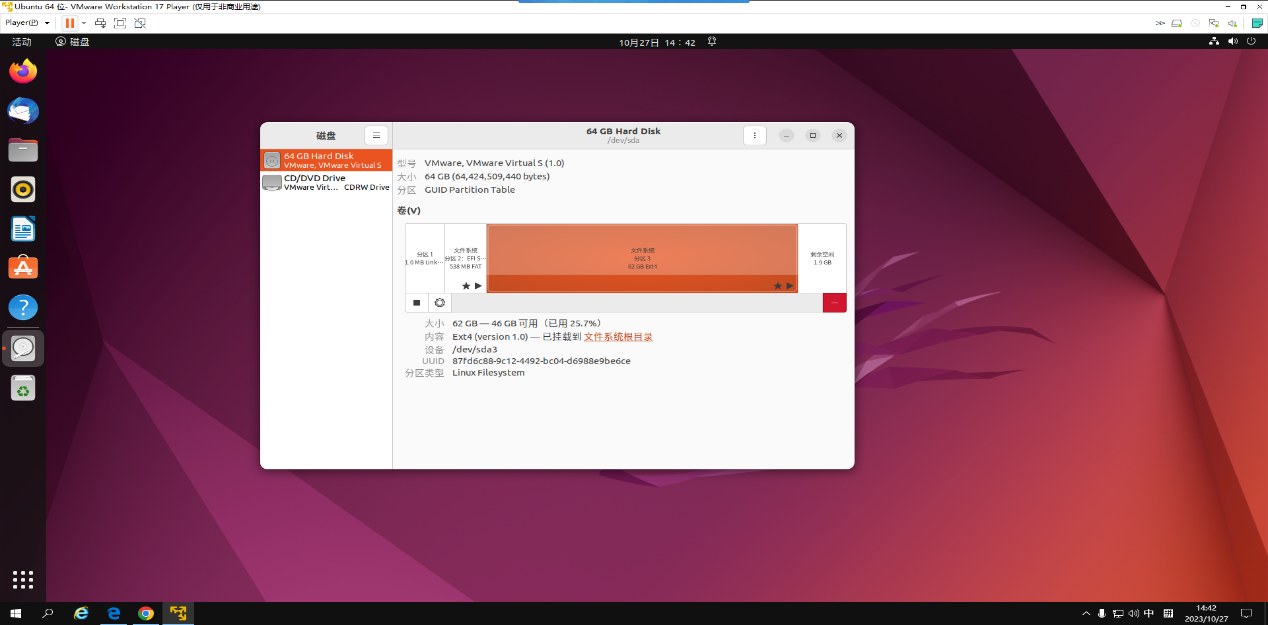
(3) 使用新的编译后的内核，编写测试程序测试该系统调用；

1. **实验过程与结果**(1) 虚拟机扩容
2. 调整磁盘大小至60G

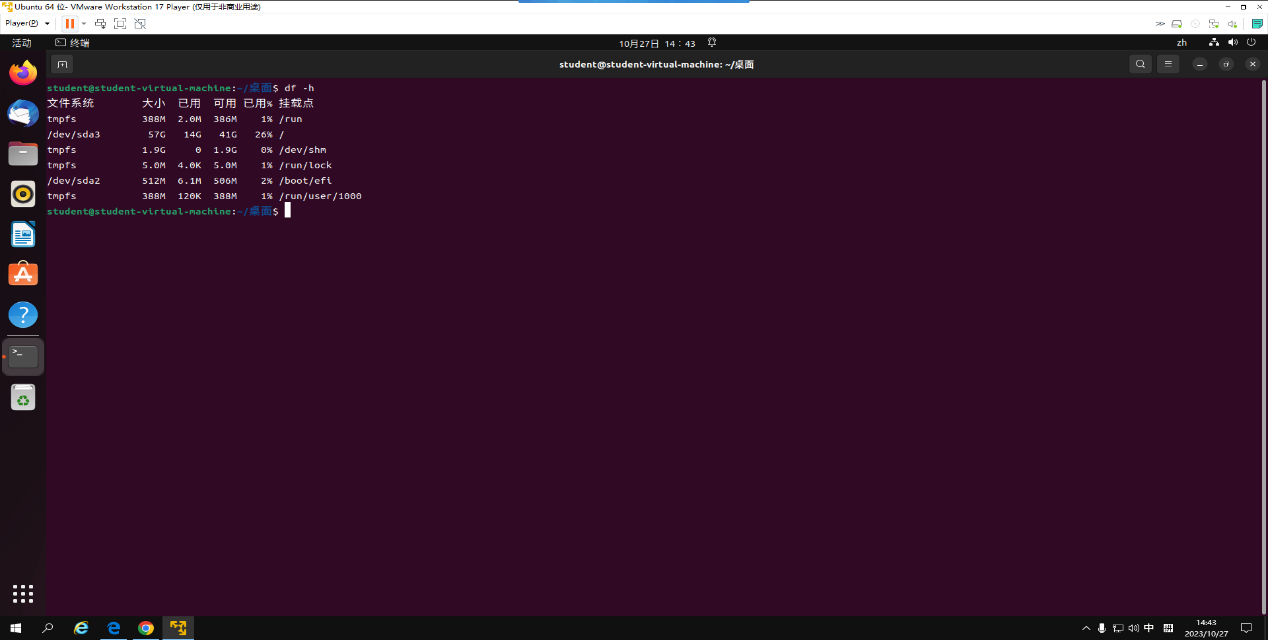


1. 在设置中调整磁盘大小

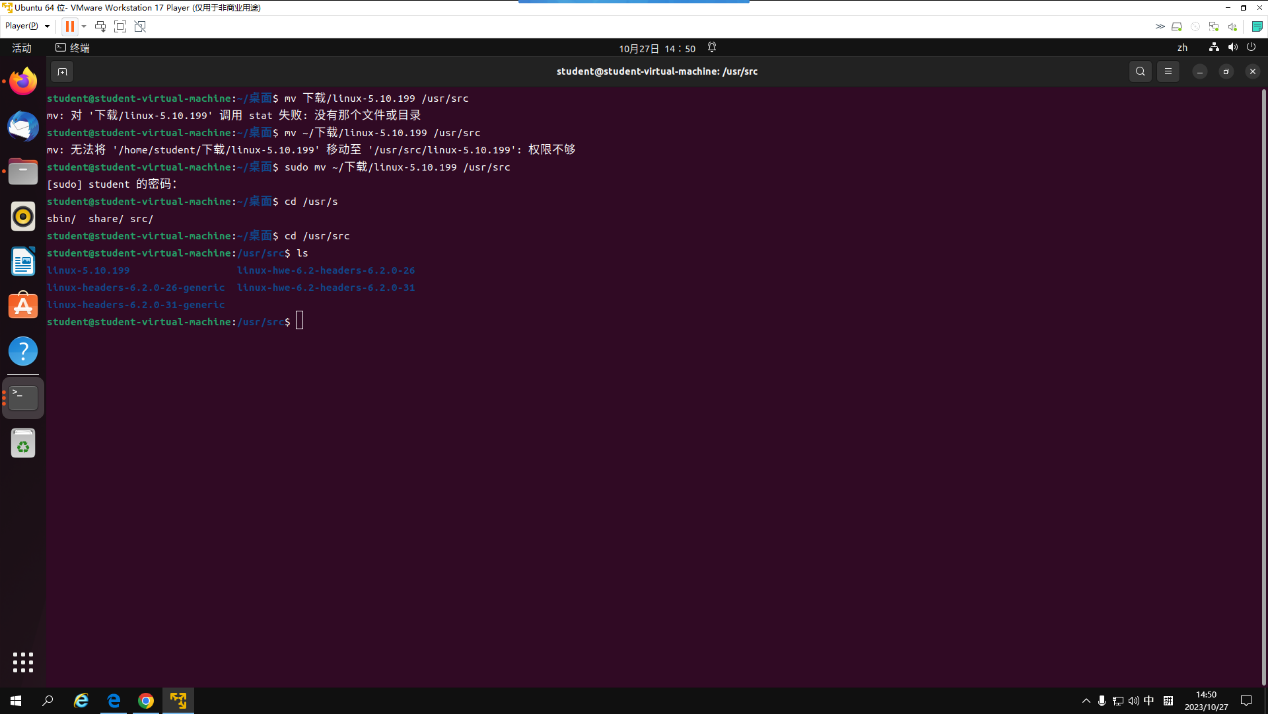




1. 使用df –h命令查看磁盘情况，发现硬盘成功扩容



（2）下载 Linux 内核文件，并移动到/usr/src/文件夹底下：



（3）编写系统调用服务程序

1.在kernel/sys.c中添加自定义函数：

SYSCALL\_DEFINE5(mycall, pid\_t, pid, int, flag, int, nicevalue, void \_\_user \*, prio, void \_\_user \*, nice)

{

/\*

根据进程号pid找到相应的进程控制块PCB；

根据PCB读取它的nice值和优先级prio；

根据PCB对相应进程的nice值进行修改；

将得到的nice值和优先级prio进行返回。

这个函数有五个参数，因此选择SYSCALL\_DEFINE5

\*/

int cur\_prio, cur\_nice;

struct pid \*ppid;

struct task\_struct \*pcb;

ppid = find\_get\_pid(pid);

pcb = pid\_task(ppid, PIDTYPE\_PID);

if (flag == 1)

{

set\_user\_nice(pcb, nicevalue);

}

else if (flag != 0)

{

return EFAULT;

}

cur\_prio = task\_prio(pcb);

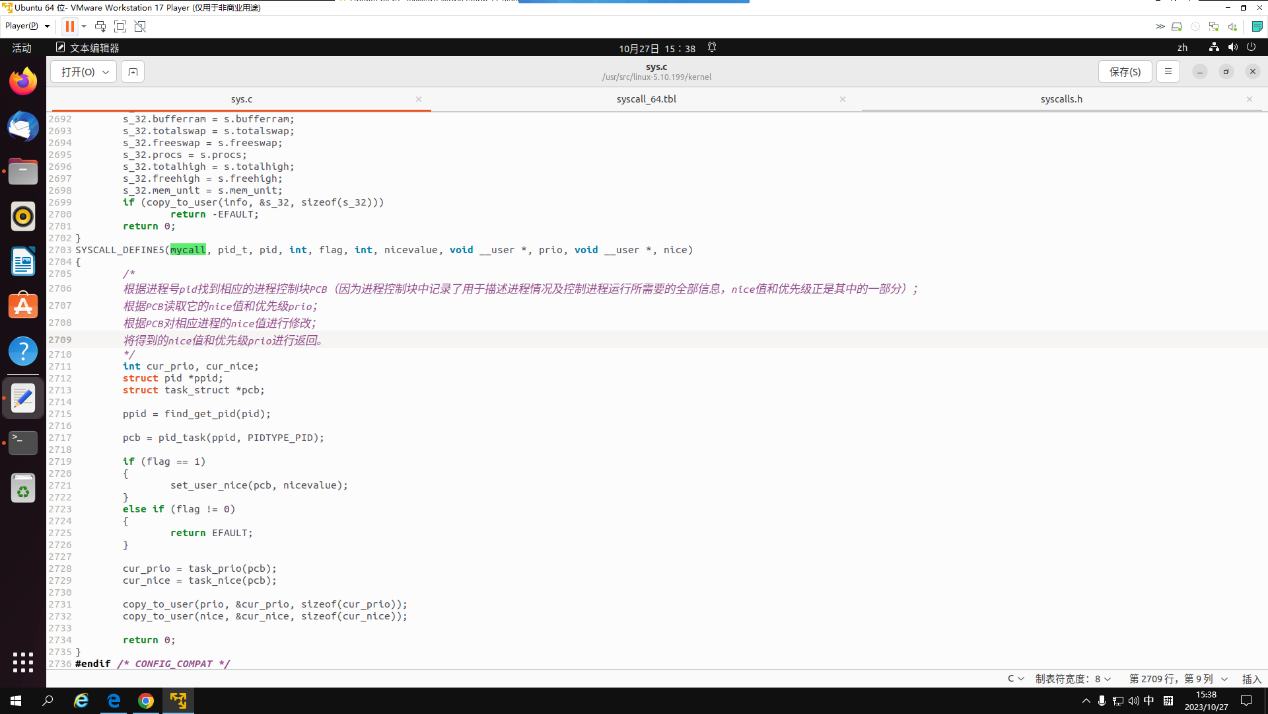
cur\_nice = task\_nice(pcb);

copy\_to\_user(prio, &cur\_prio, sizeof(cur\_prio));

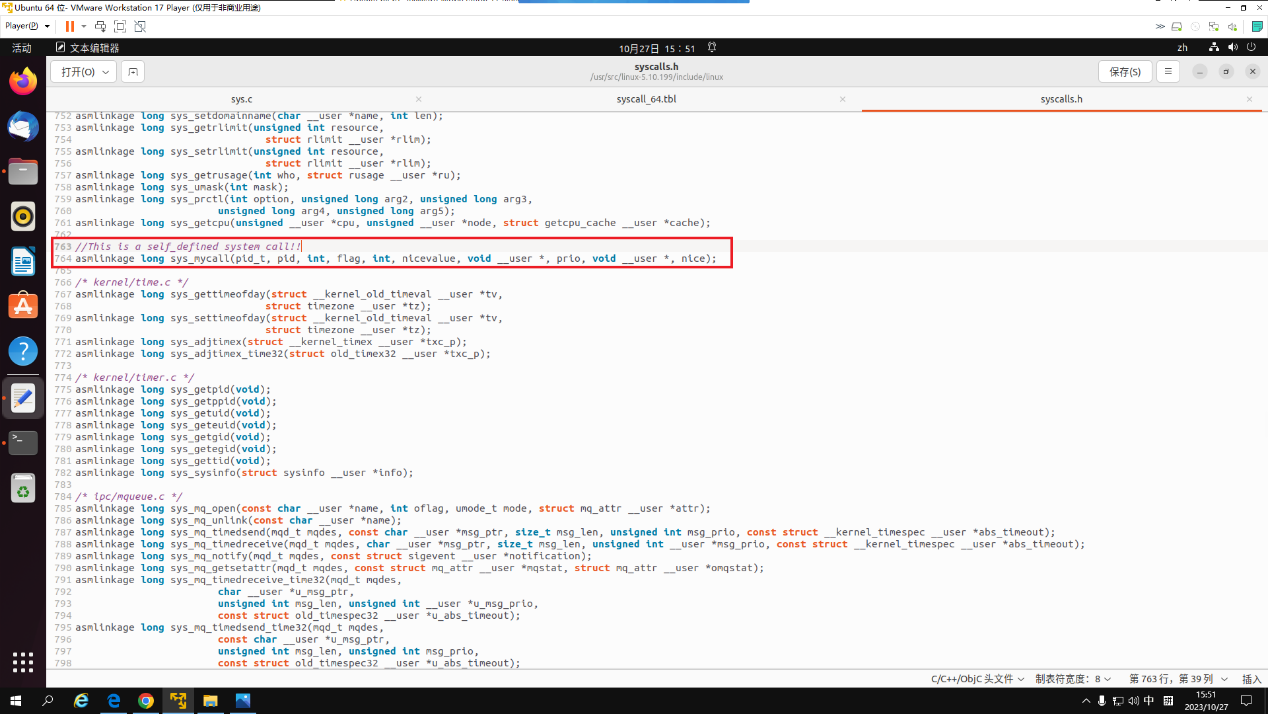
copy\_to\_user(nice, &cur\_nice, sizeof(cur\_nice));

return 0;

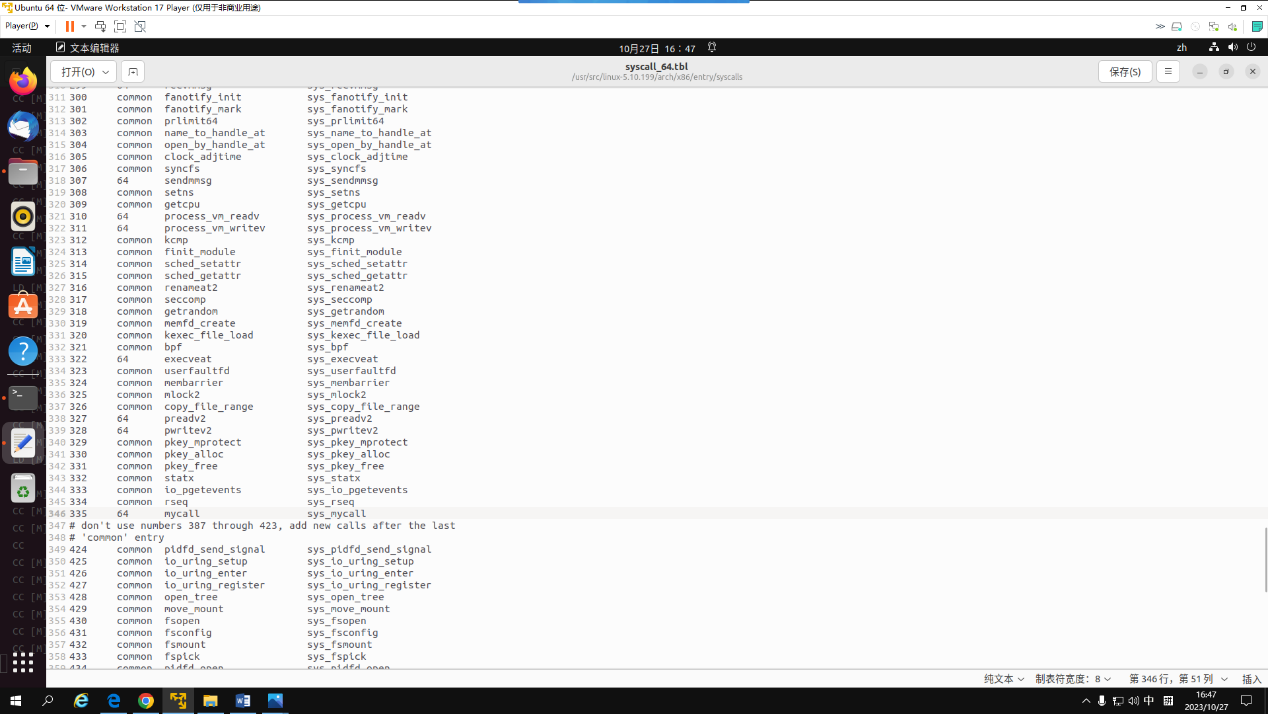
}



2.在include/linux/syscalls.h 头文件中添加函数声明

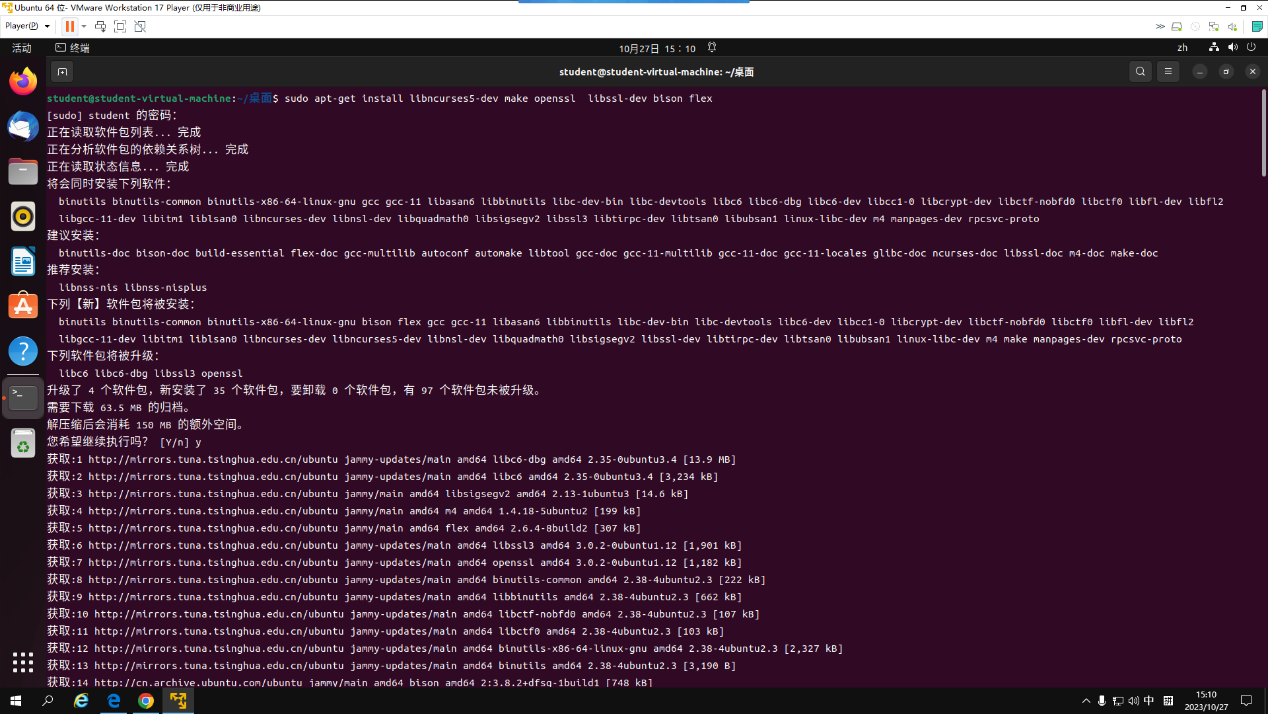


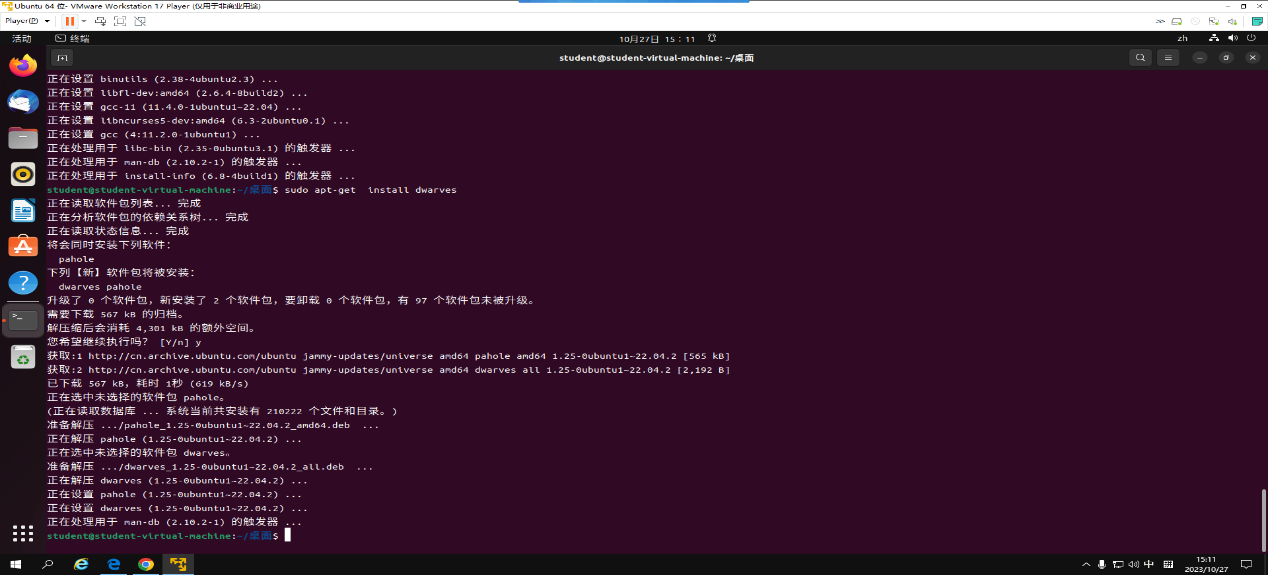
1. 在arch/x86/entry/syscalls/syscall\_64.tbl中修改系统调用表，添加系统调用号（335）



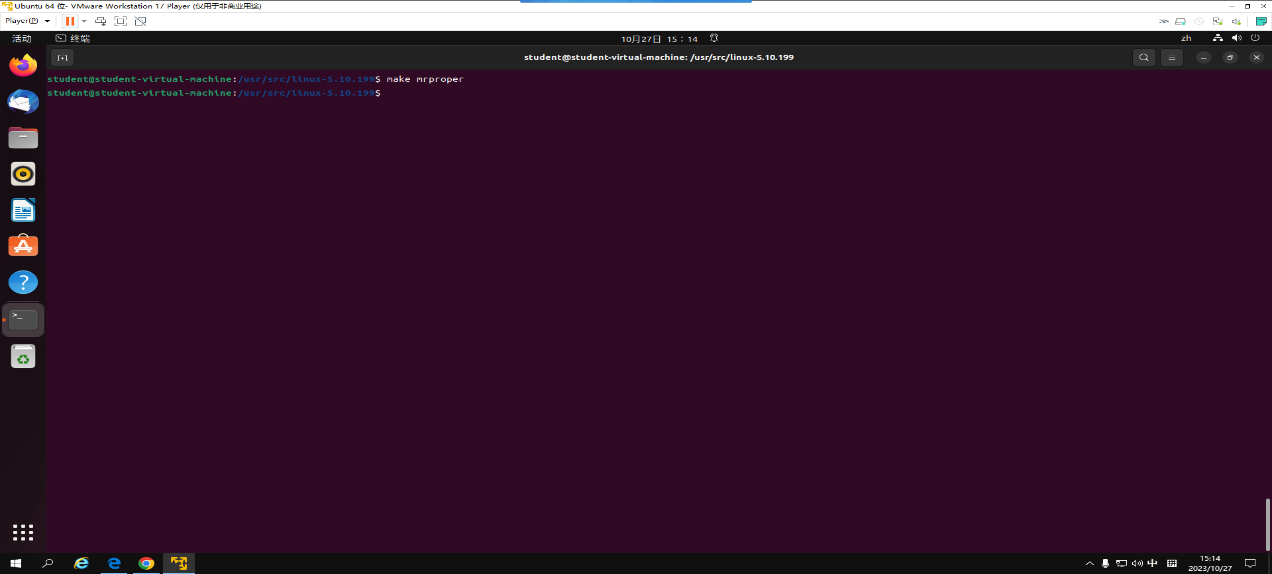
(4) 编译内核

1.安装依赖包

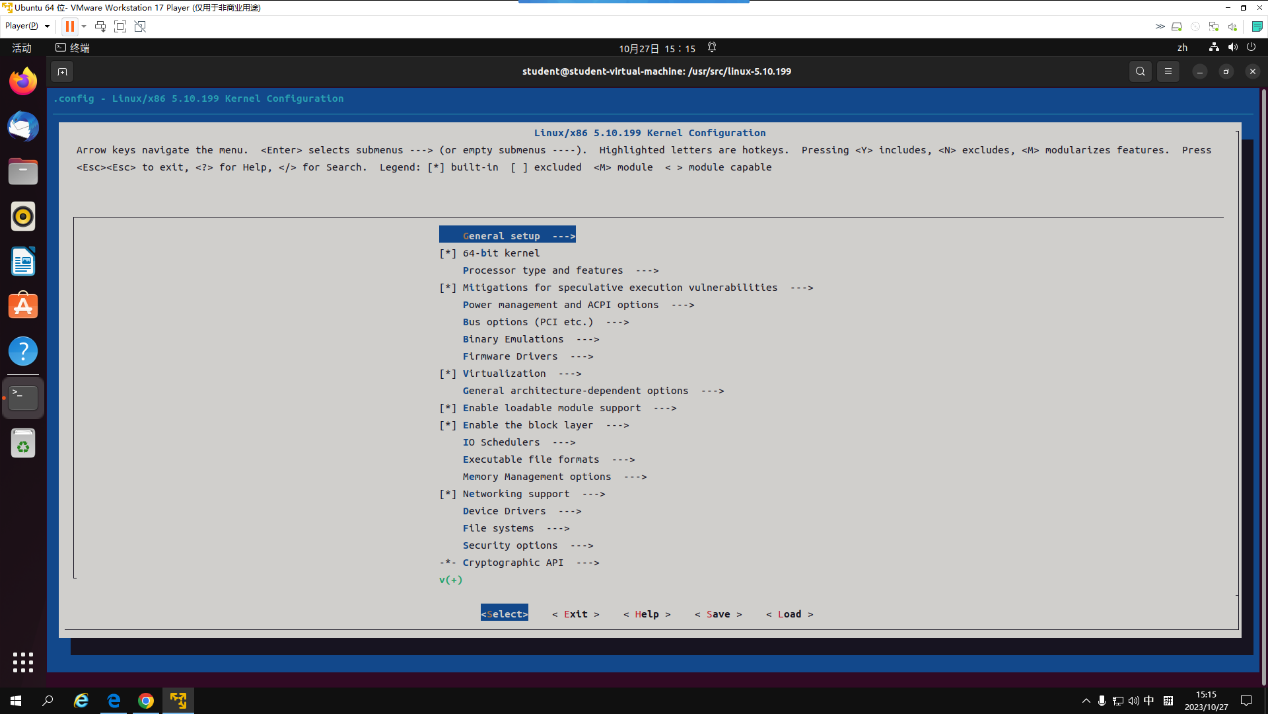


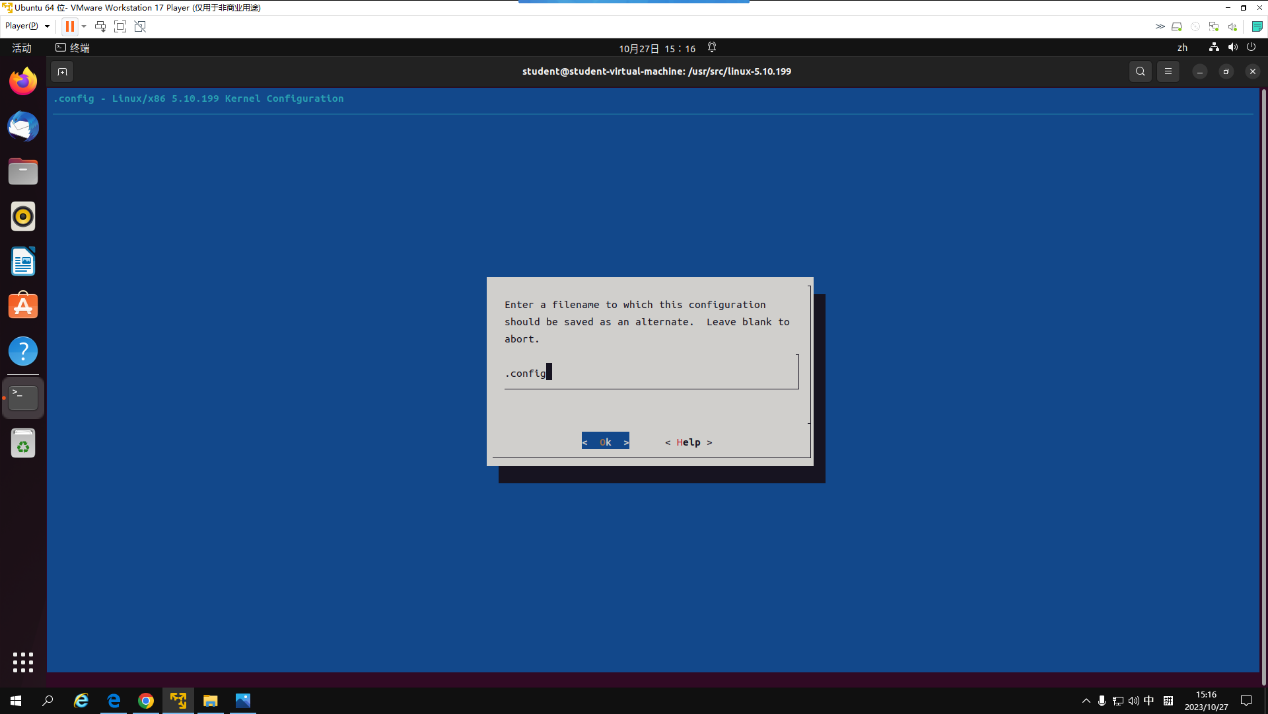


2.使用make mrproper删除之前的编译信息

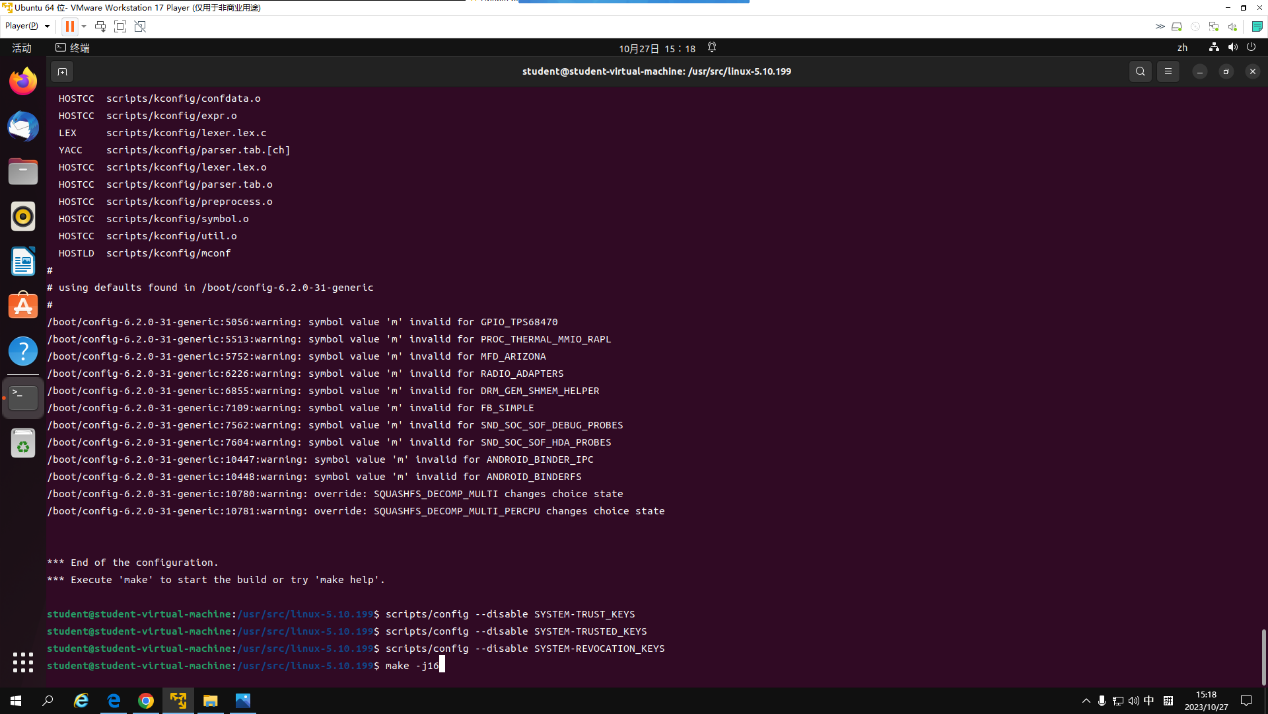


3.使用make menu config生成配置清单

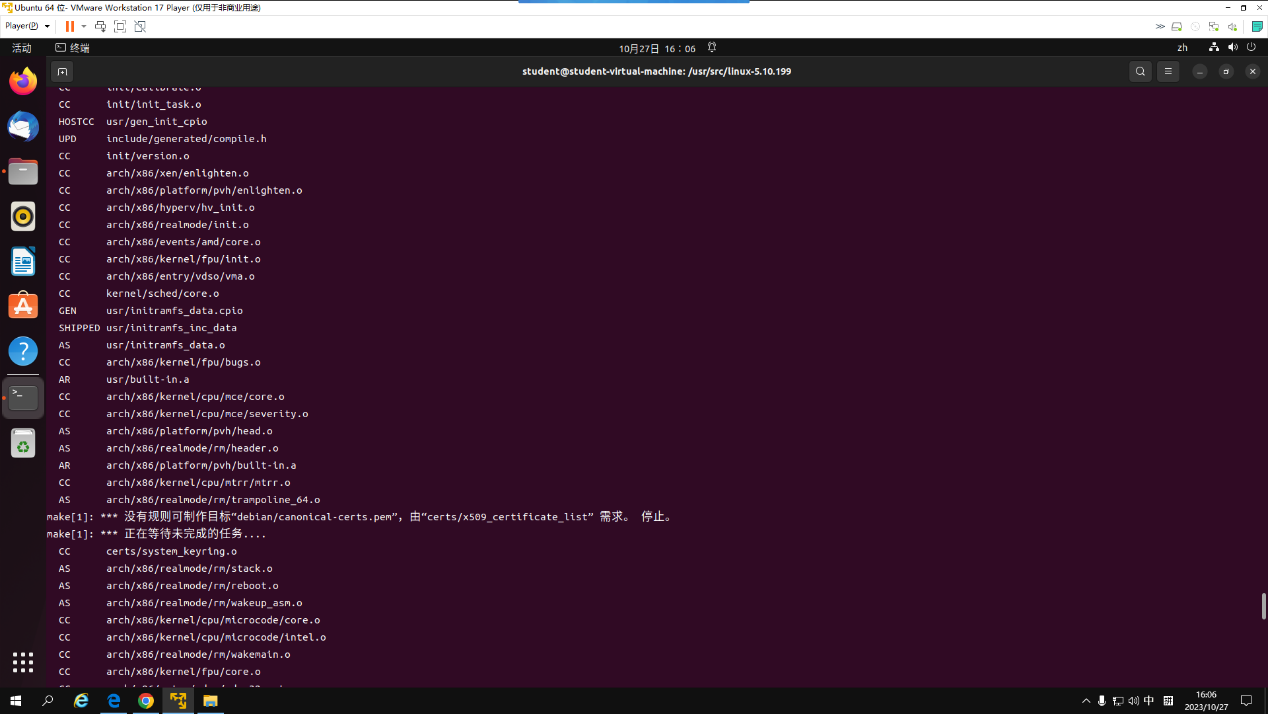




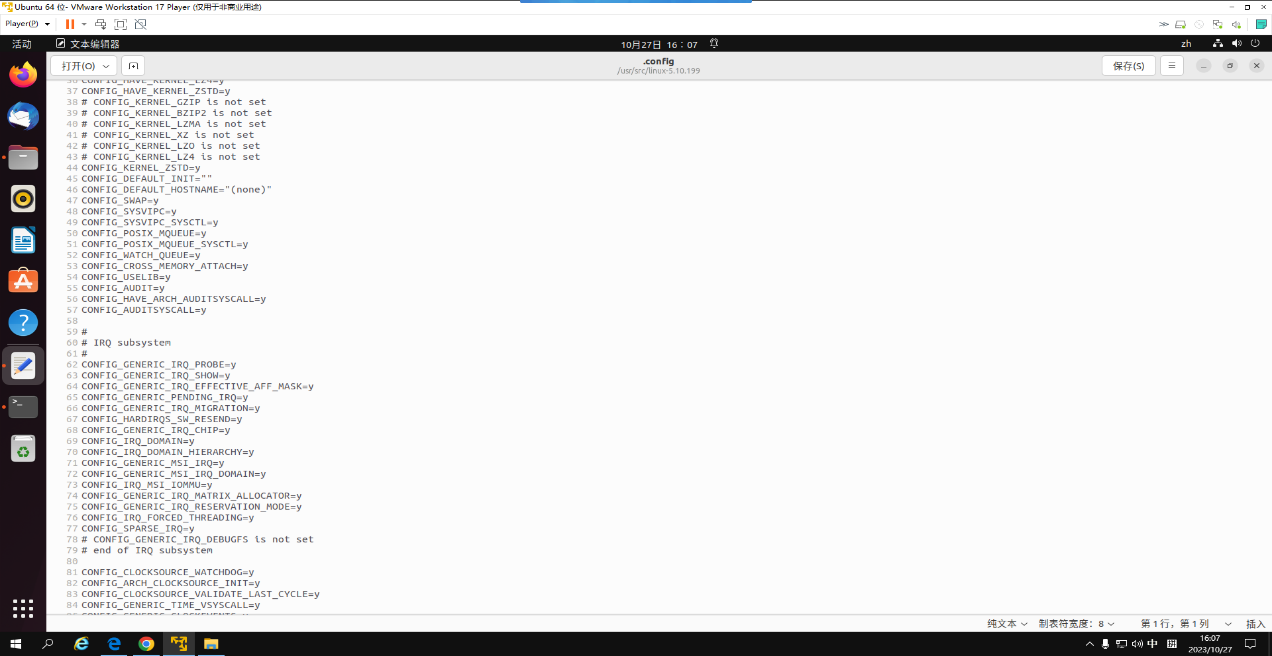
4.使用make j16命令开始编译



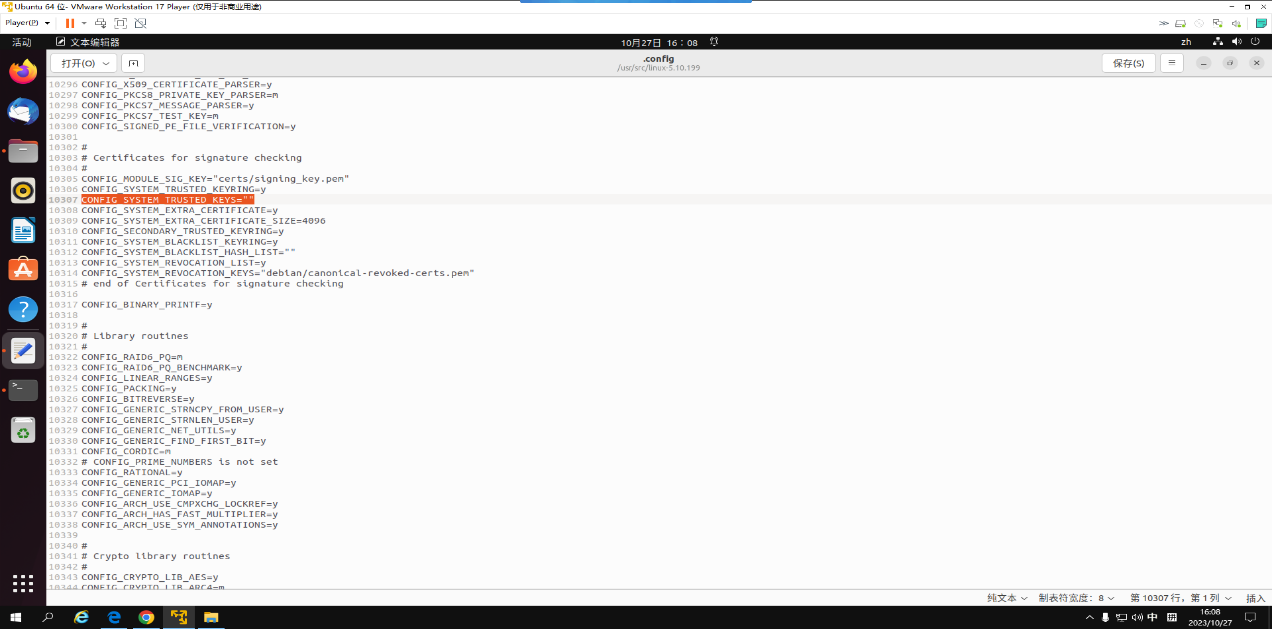
出现错误：



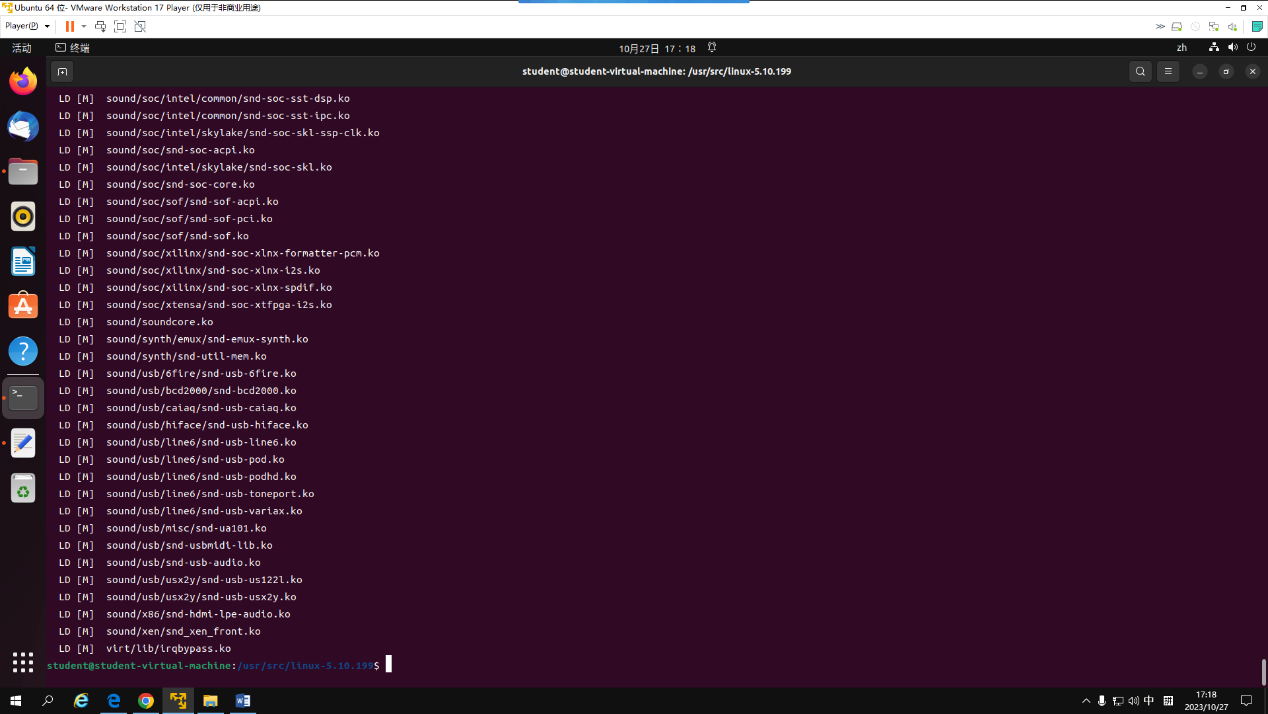
发现可能是配置清单文件设置不正确，通过grep .config命令打开配置文件：



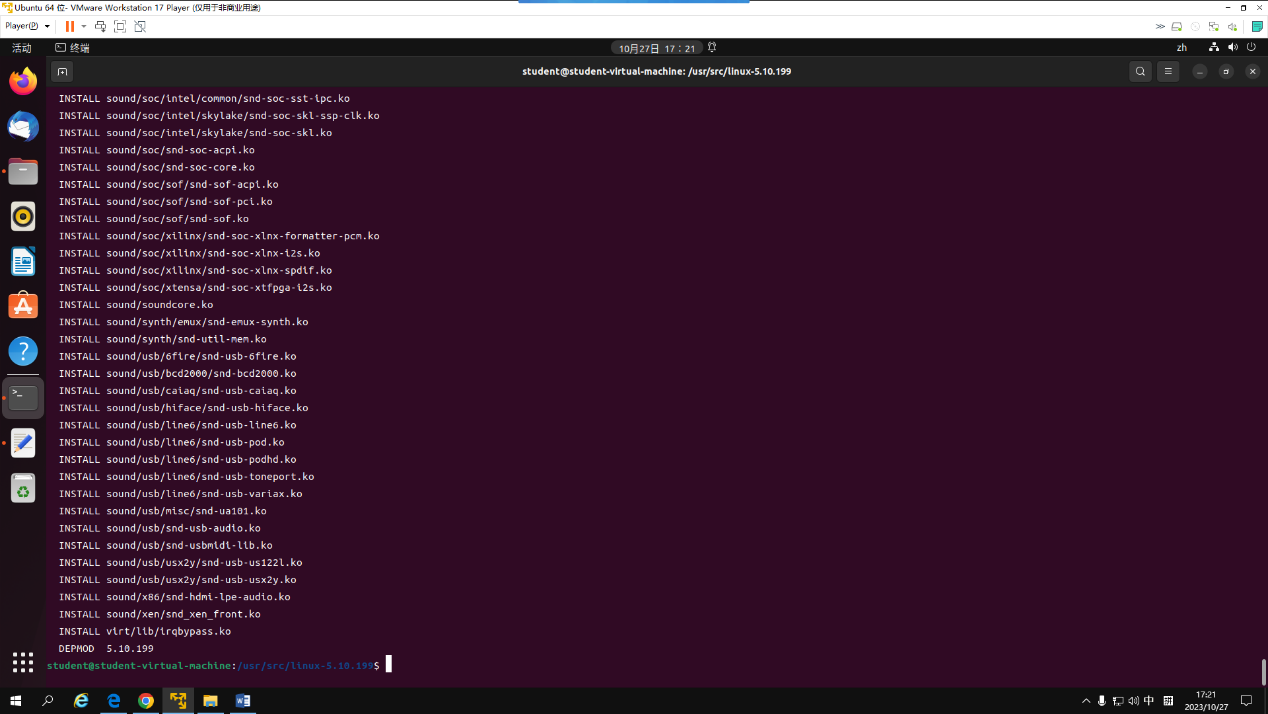
将CONFIG\_SYSTEM\_TRUSTED\_KEYS与CONFIG\_SYSTEM\_REVOCATION\_KEYS两项设置为空：



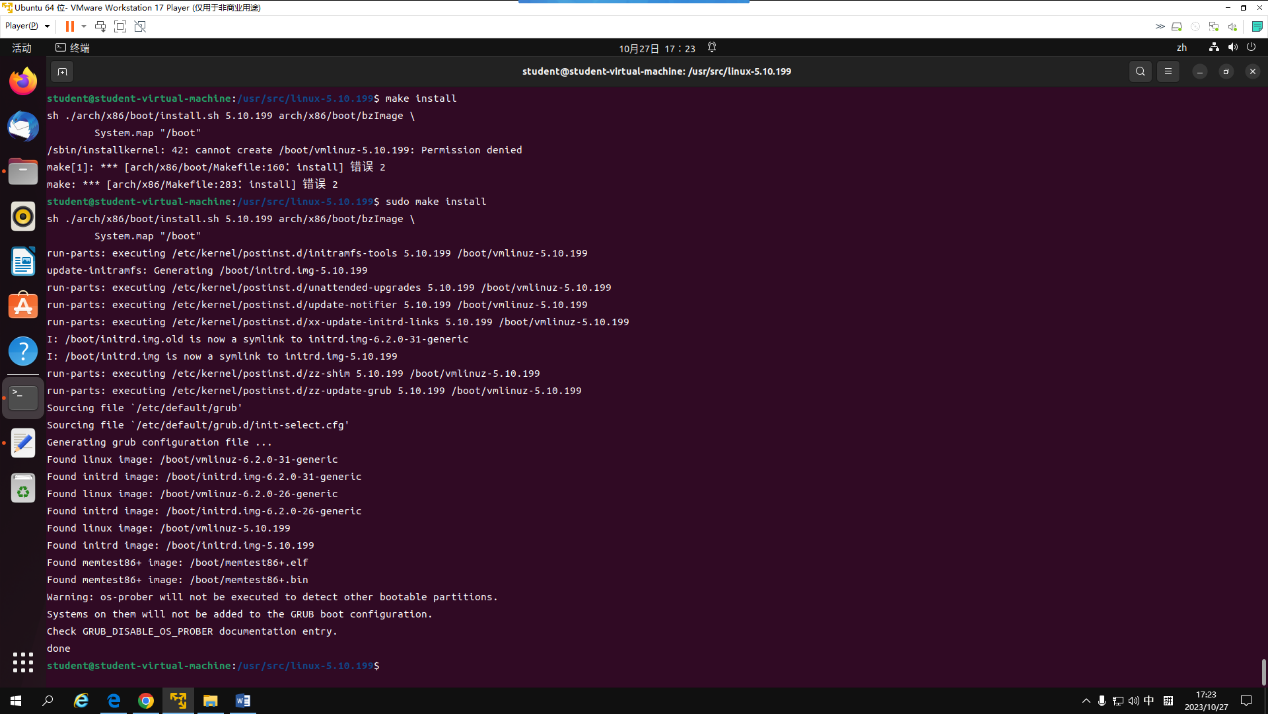
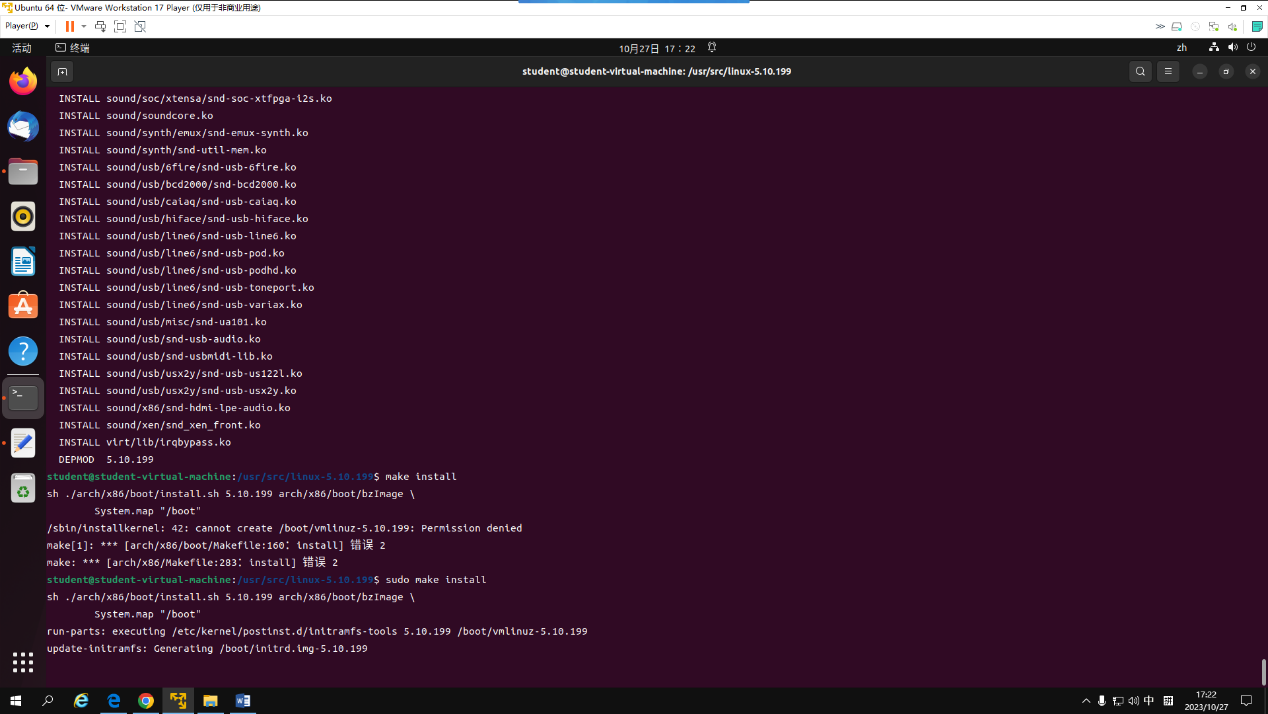
经过1个多小时的编译，成功生成：



使用sudo make modules\_install 模块命令：

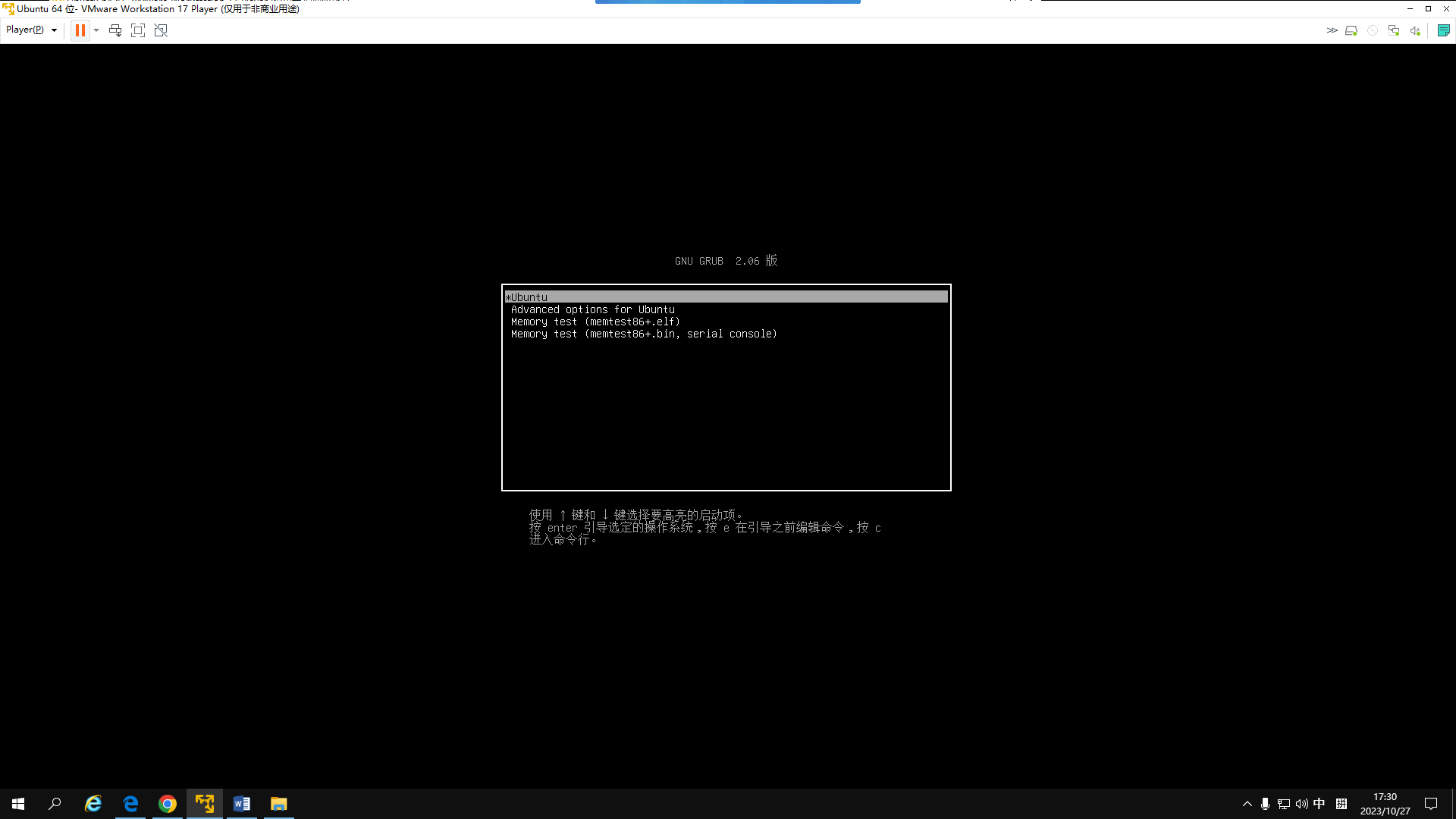


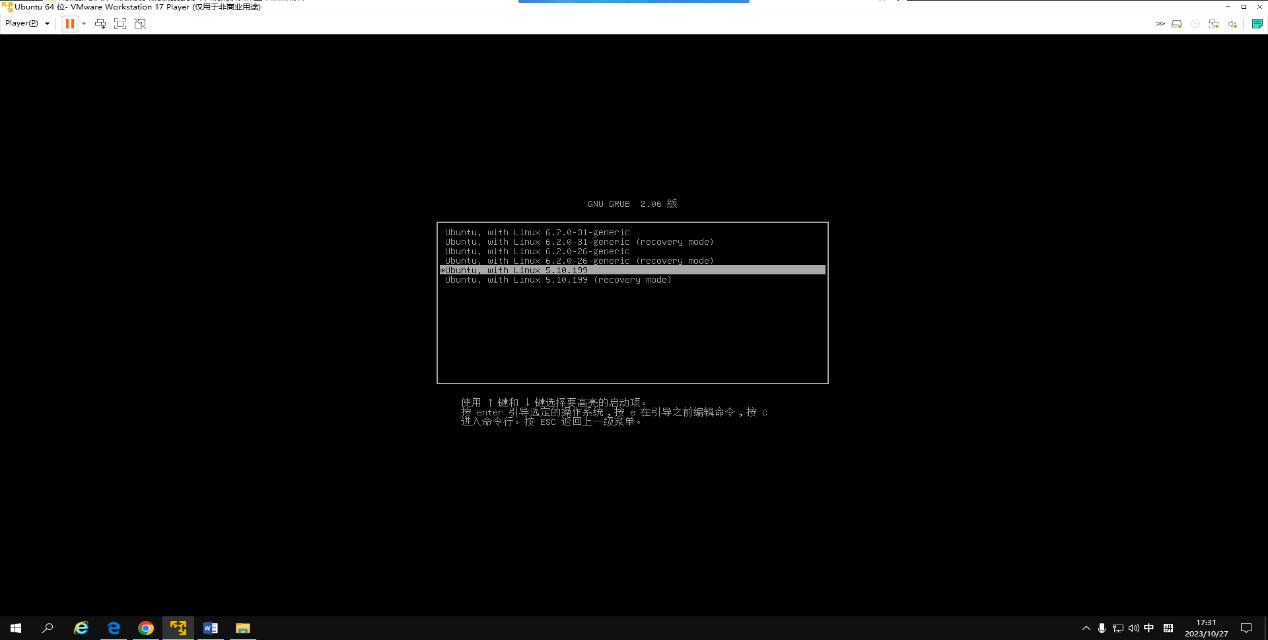
使用sudo make install 安装内核：



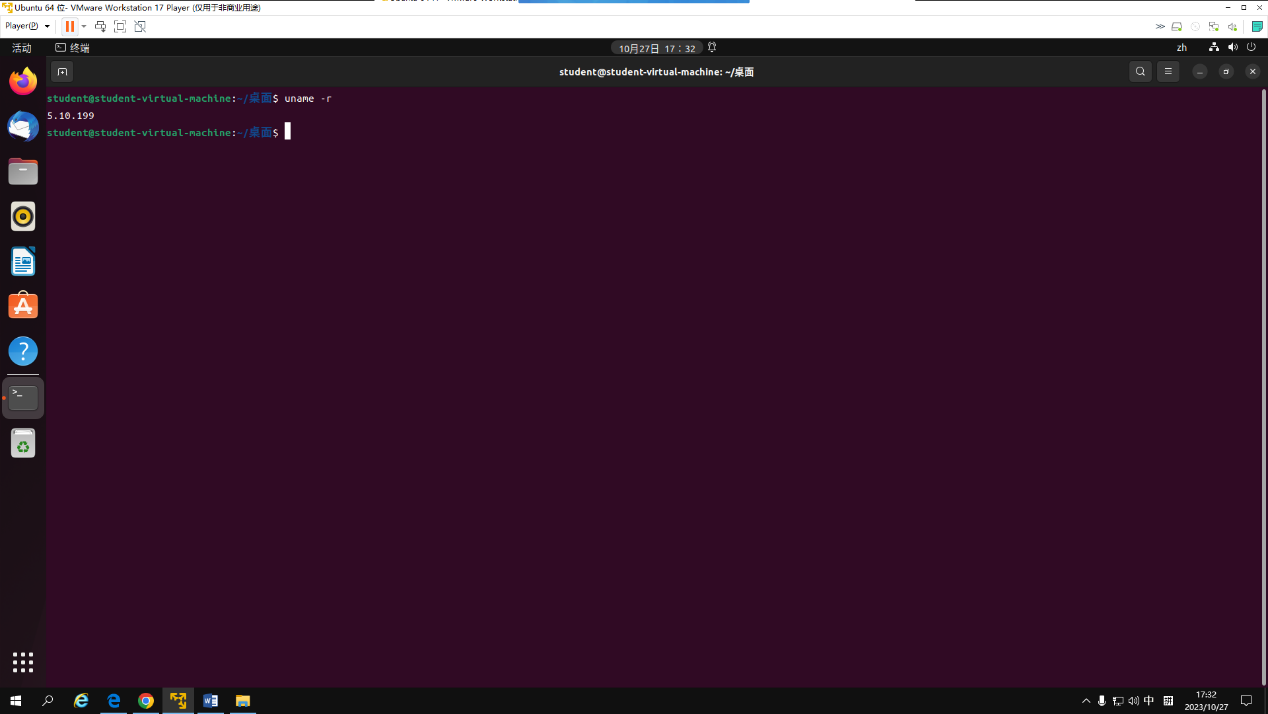
(5) 切换到新的 Linux 内核

1.切换到启动界面，选择Advanced Options -> 5.10 版本





2.查看版本号，发现版本号正确，内核装载成功



(6) 编写程序测试系统调用

编写的程序如下：

#include <syscall.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/types.h>

#define MY\_CALL\_NUMBER 335

int main(){

    int pid, flag, nicevalue;

        int prev\_prio, prev\_nice, cur\_prio, cur\_nice;

        int result;

        printf("Please input variable(pid, flag, nicevalue): ");

        scanf("%d%d%d", &pid, &flag, &nicevalue);

        result = syscall(MY\_CALL\_NUMBER, pid, 0, nicevalue, &prev\_prio,

                     &prev\_nice);

        if (result == 14)

        {

            printf("ERROR!");

            return 1;

        }

        if (flag == 1)

        {

            syscall(MY\_CALL\_NUMBER, pid, 1, nicevalue, &cur\_prio, &cur\_nice);

            printf("Original priority is: [%d], original nice is [%d]\n", prev\_prio,prev\_nice);

            printf("Current priority is : [%d], current nice is [%d]\n", cur\_prio,cur\_nice);

        }

        else if (flag == 0)

        {

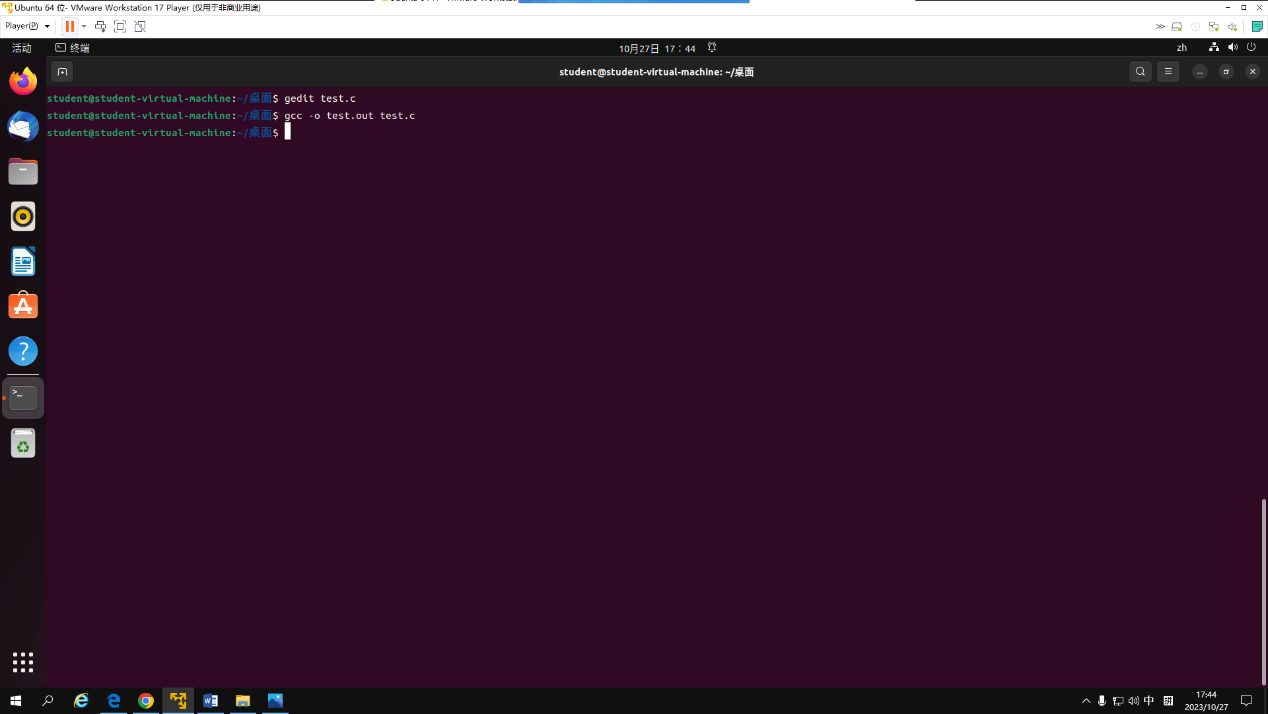
            printf("Current priority is : [%d], current nice is [%d]\n", prev\_prio,prev\_nice);

        }

    return 0;

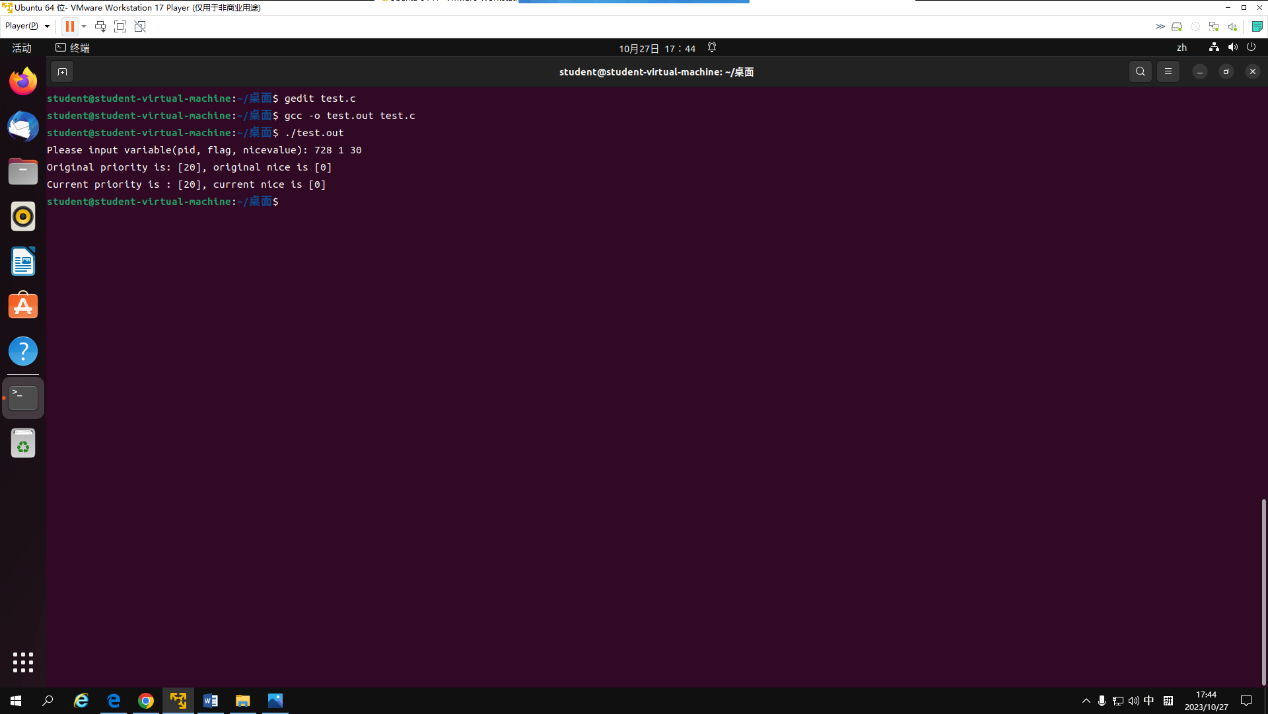
}

使用gcc编译器编译，没有错误：



运行程序，发现成功启动系统调用

（输入pid = 728 ， flag = 30 ， nice = 20）



1. **实验总结**
2. Linux系统中新增并实现系统调用的基本步骤：

1.编写系统调用函数: 首先需要编写系统调用的实际功能代码，定义一个新的函数，该函数将执行作为系统调用提供的操作；

2．编写系统调用的包装函数: 在内核中，系统调用被标识为一个特定的系统调用号。为了在用户空间中使用系统调用，需要编写一个包装函数，该函数将调用内核中的系统调用。包装函数将把参数传递给内核，并处理返回值；

3.更新系统调用表: Linux内核使用一个系统调用表（System Call Table）来映射系统调用号和相应的内核函数。应该向这个表中添加新的系统调用号，以便将新的系统调用号和包装函数关联起来；

4.重新编译和安装内核: 在修改了内核源代码后，需要重新编译内核并安装新的内核映像；

5.重新启动系统: 完成内核的重新编译和安装后，需要重新启动系统以加载新的内核，新系统调用就可以在用户空间中使用了；

6.在用户空间中使用系统调用: 在系统启动后，在用户空间的应用程序中使用新的系统调用，通过调用包装函数来访问内核中实现的新功能；