《openEuler实验》

一、实验分工

姓名：项 枫 班级：2022211801学号：2022211570 分工：100%

二、实验环境

虚拟机Vmware和openEuler操作系统。

三、实验过程

（一）openEuler操作系统安装与内核编译

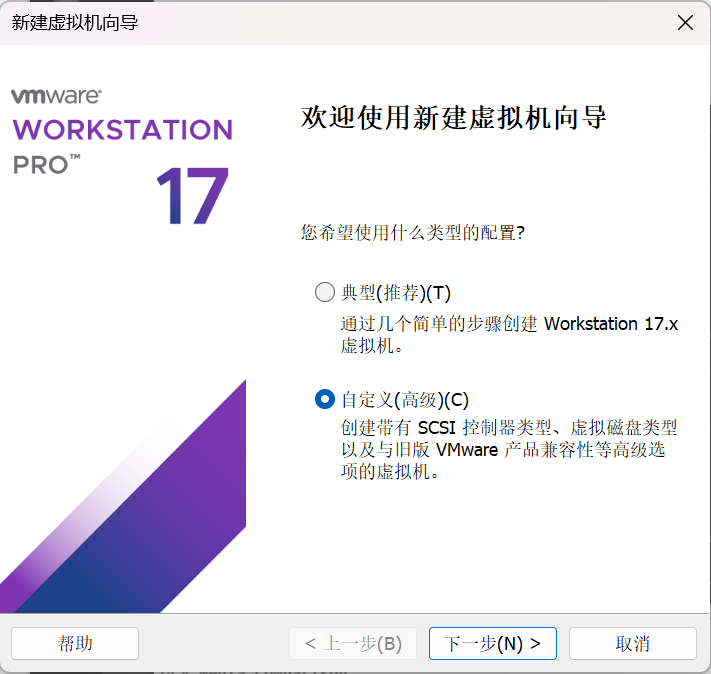
1、操作系统安装

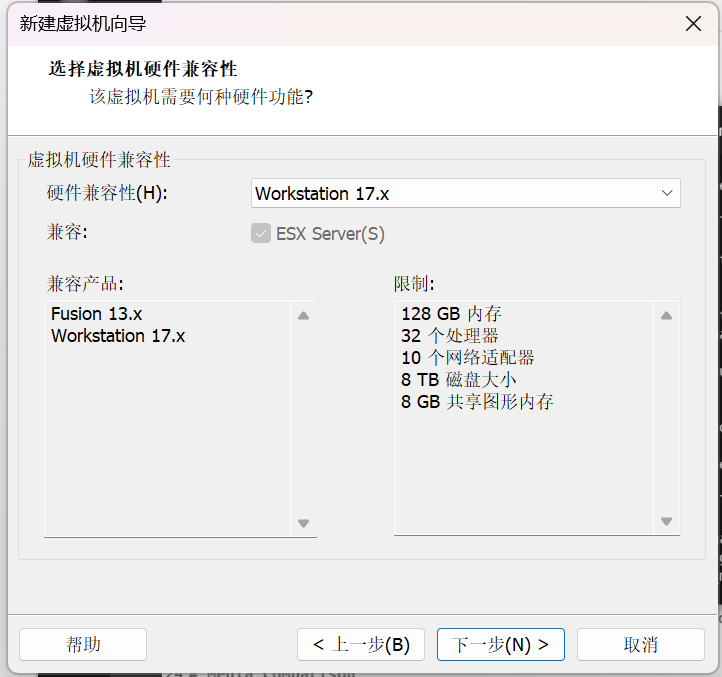
（1）下载OpenEular镜像

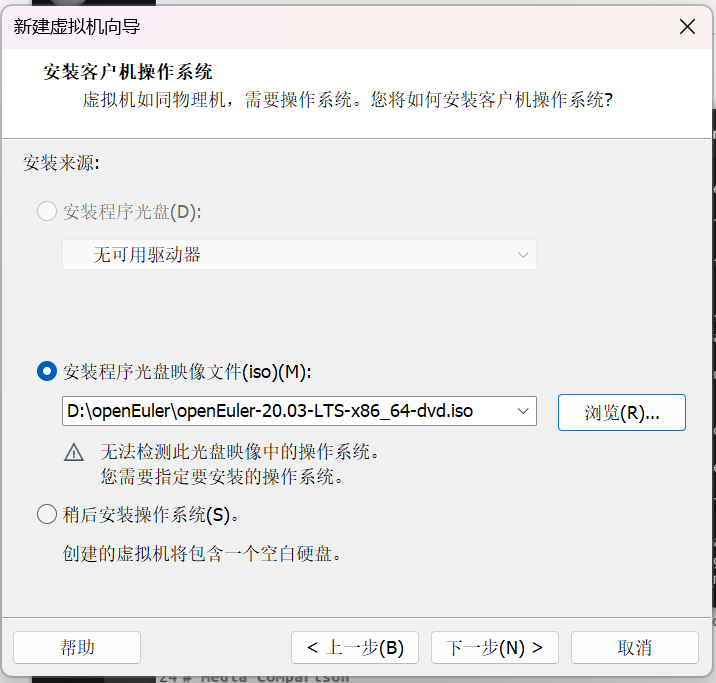


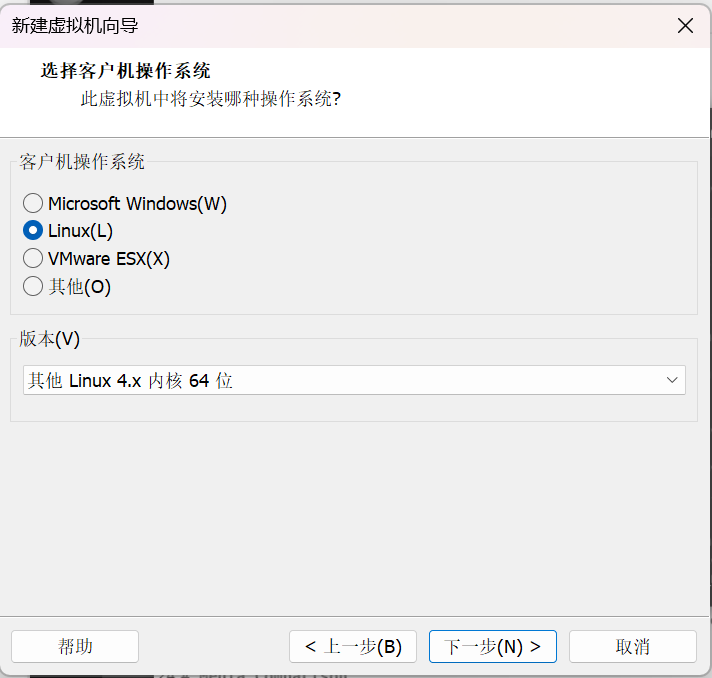
（2）安装到虚拟机

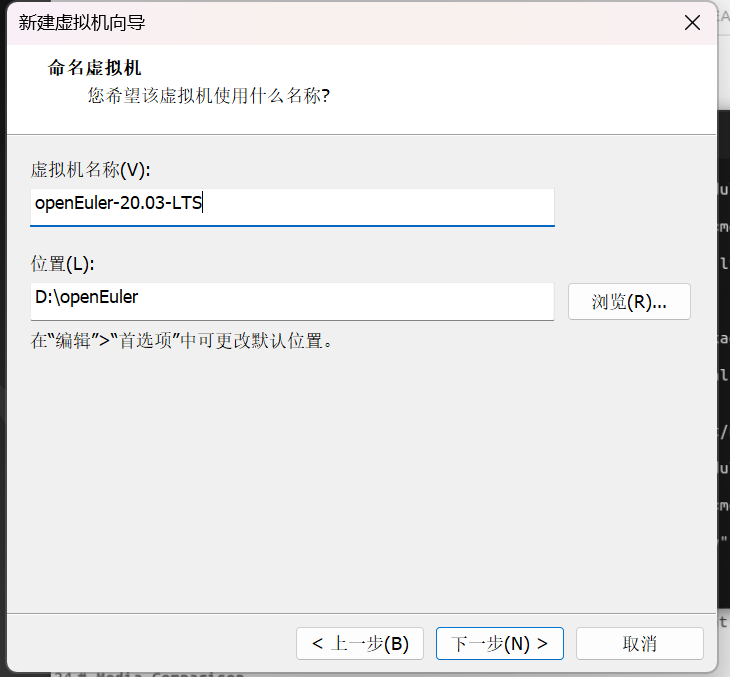
①新建虚拟机

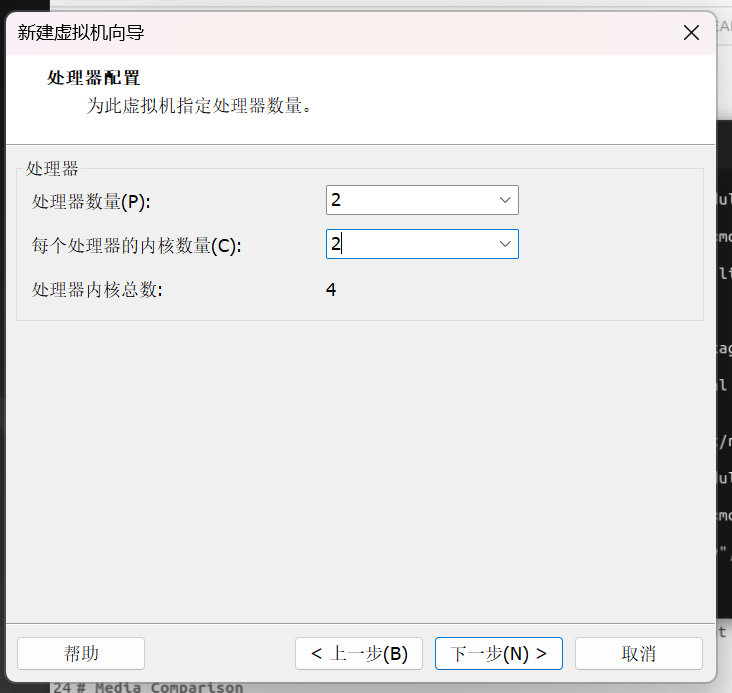








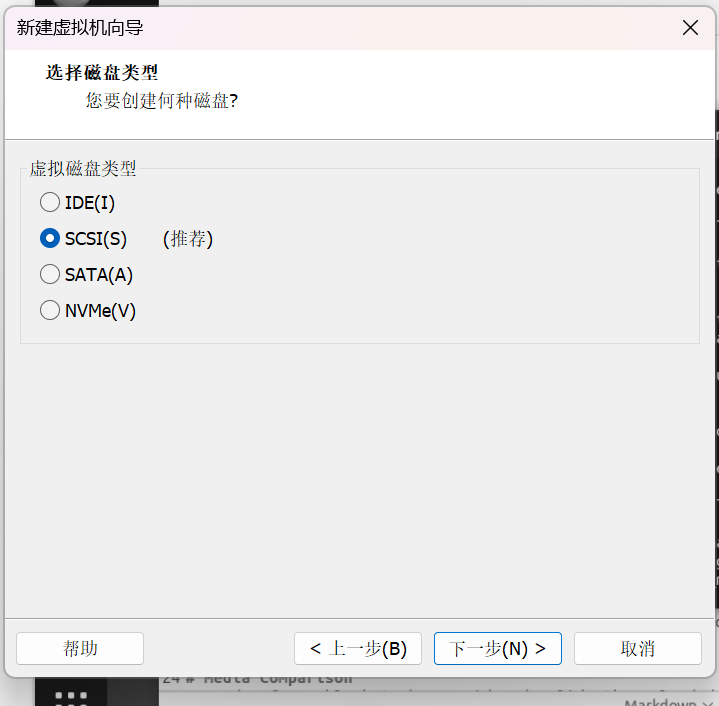


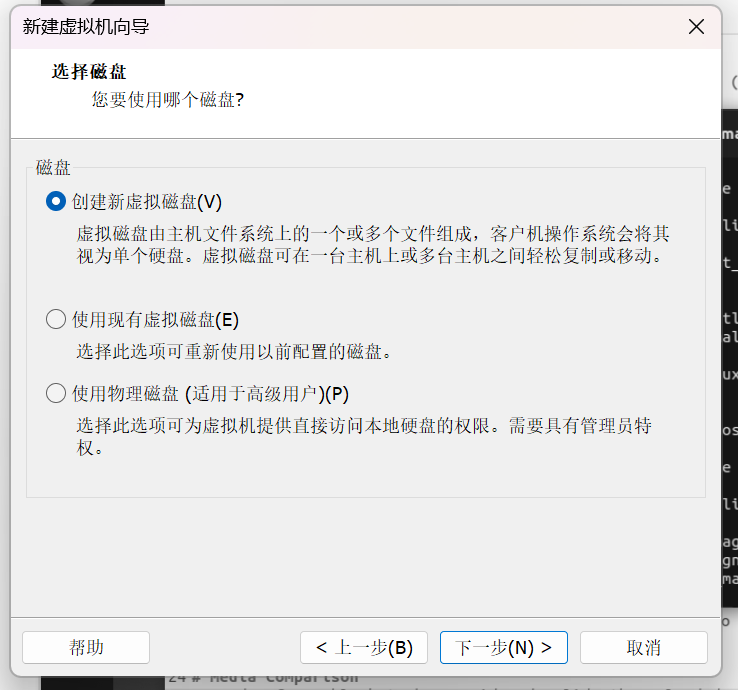


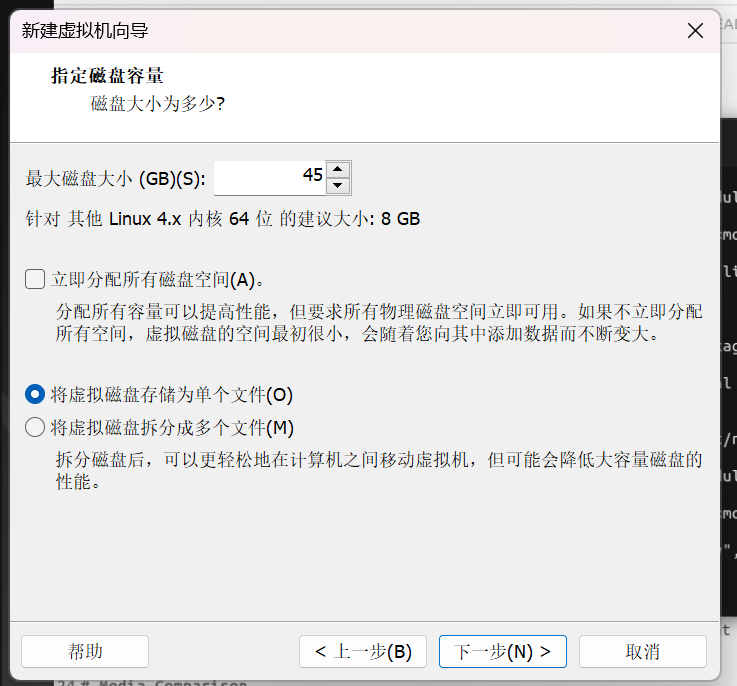


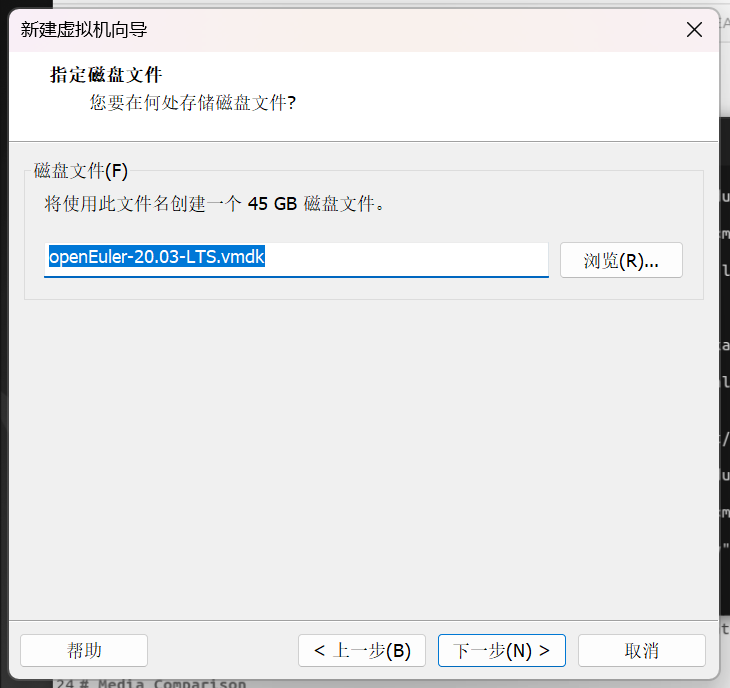


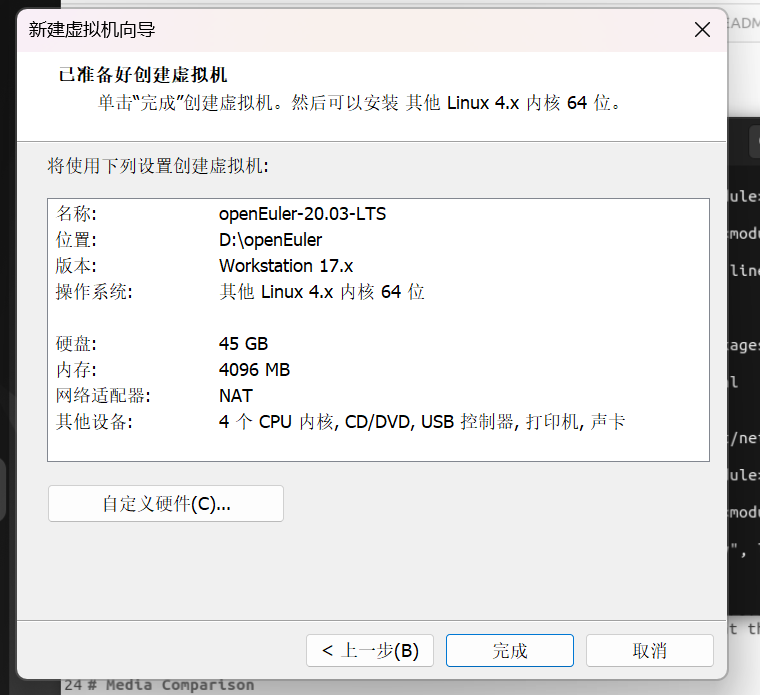




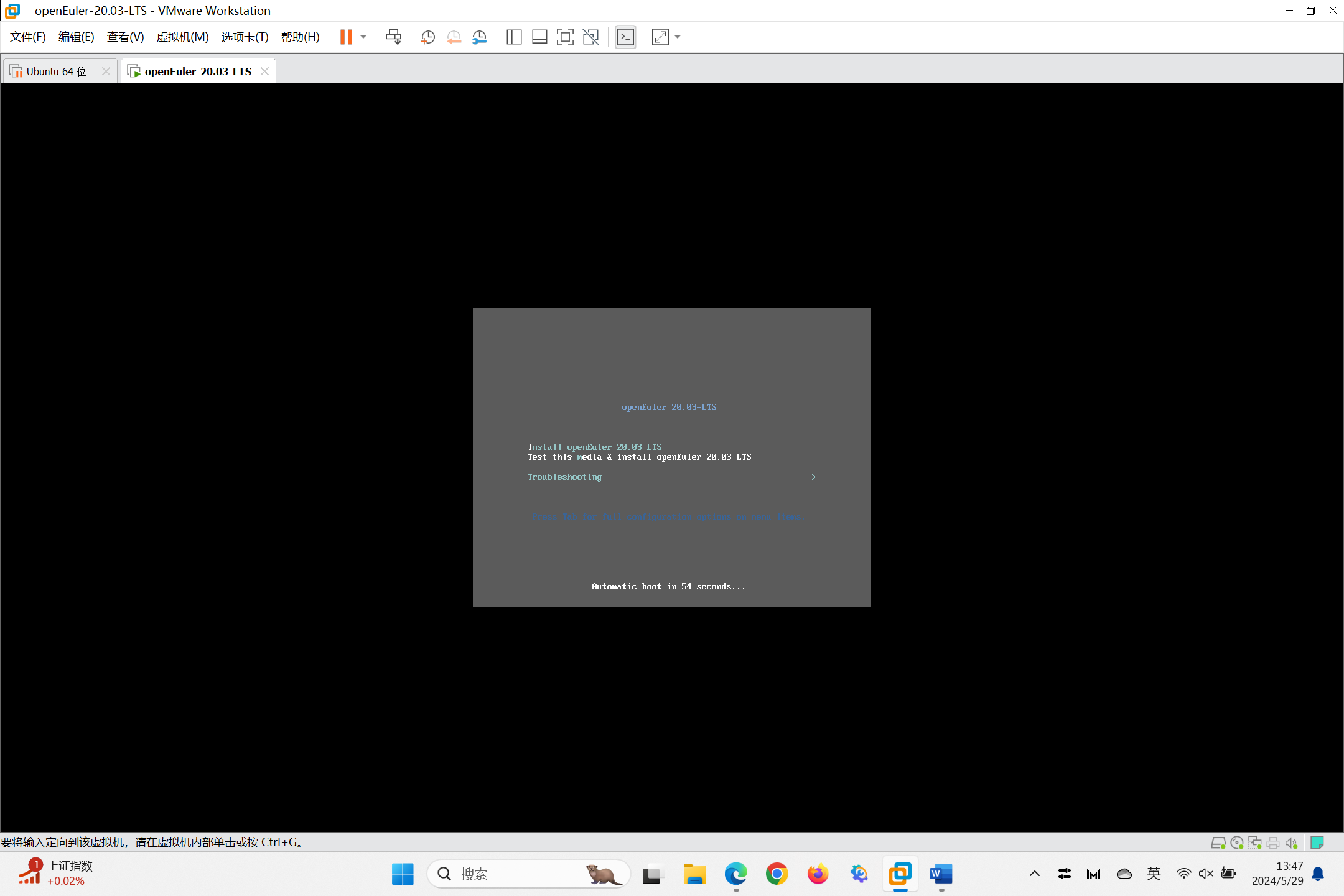


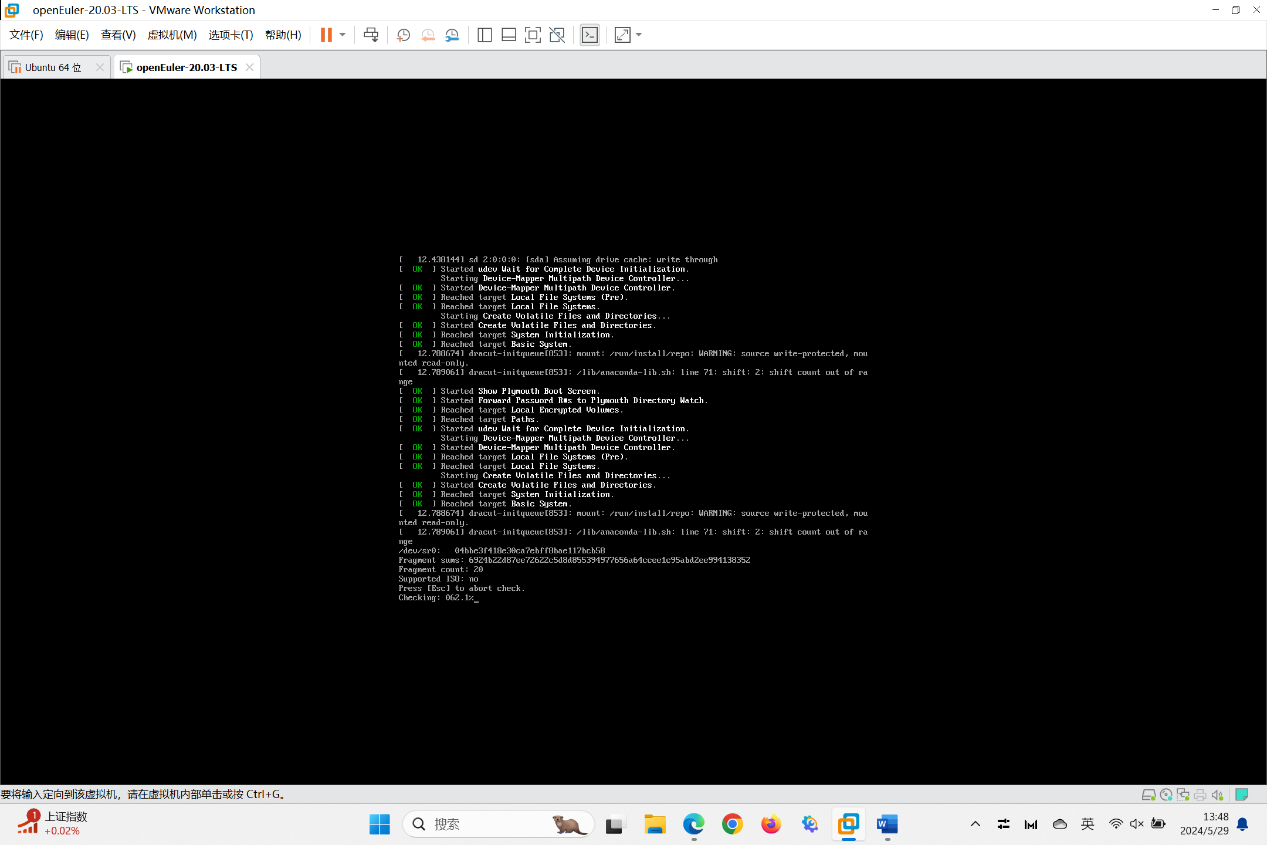


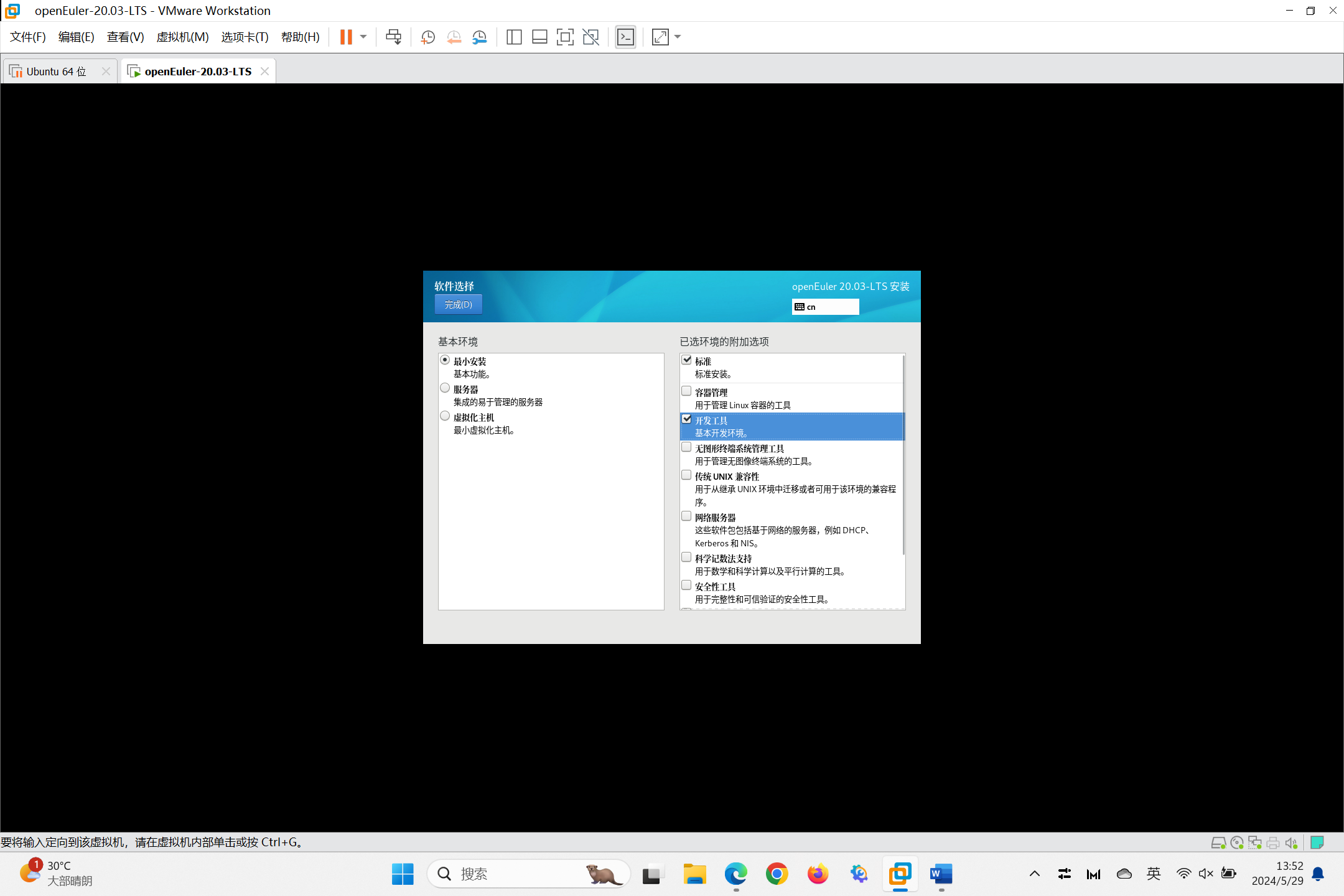


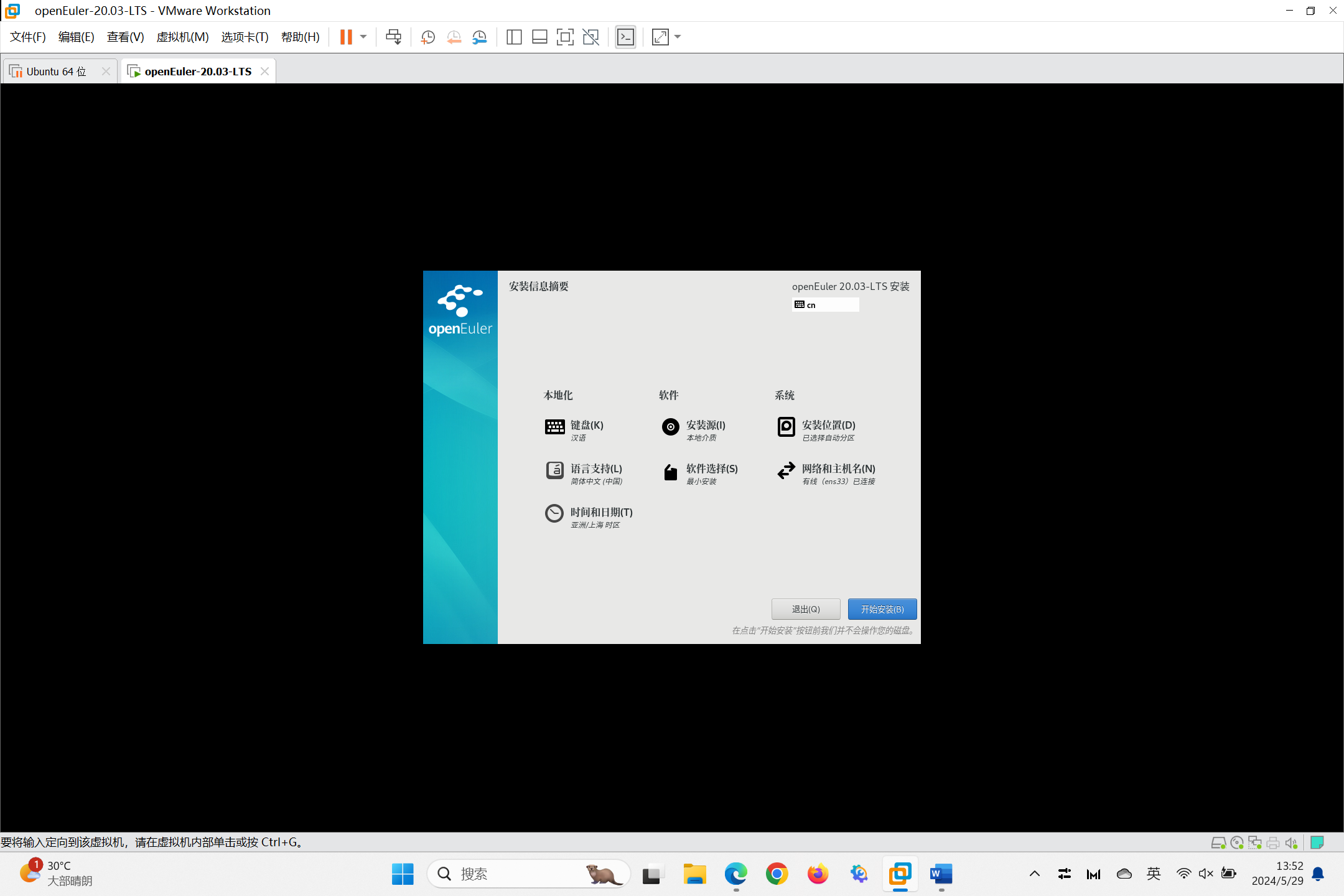


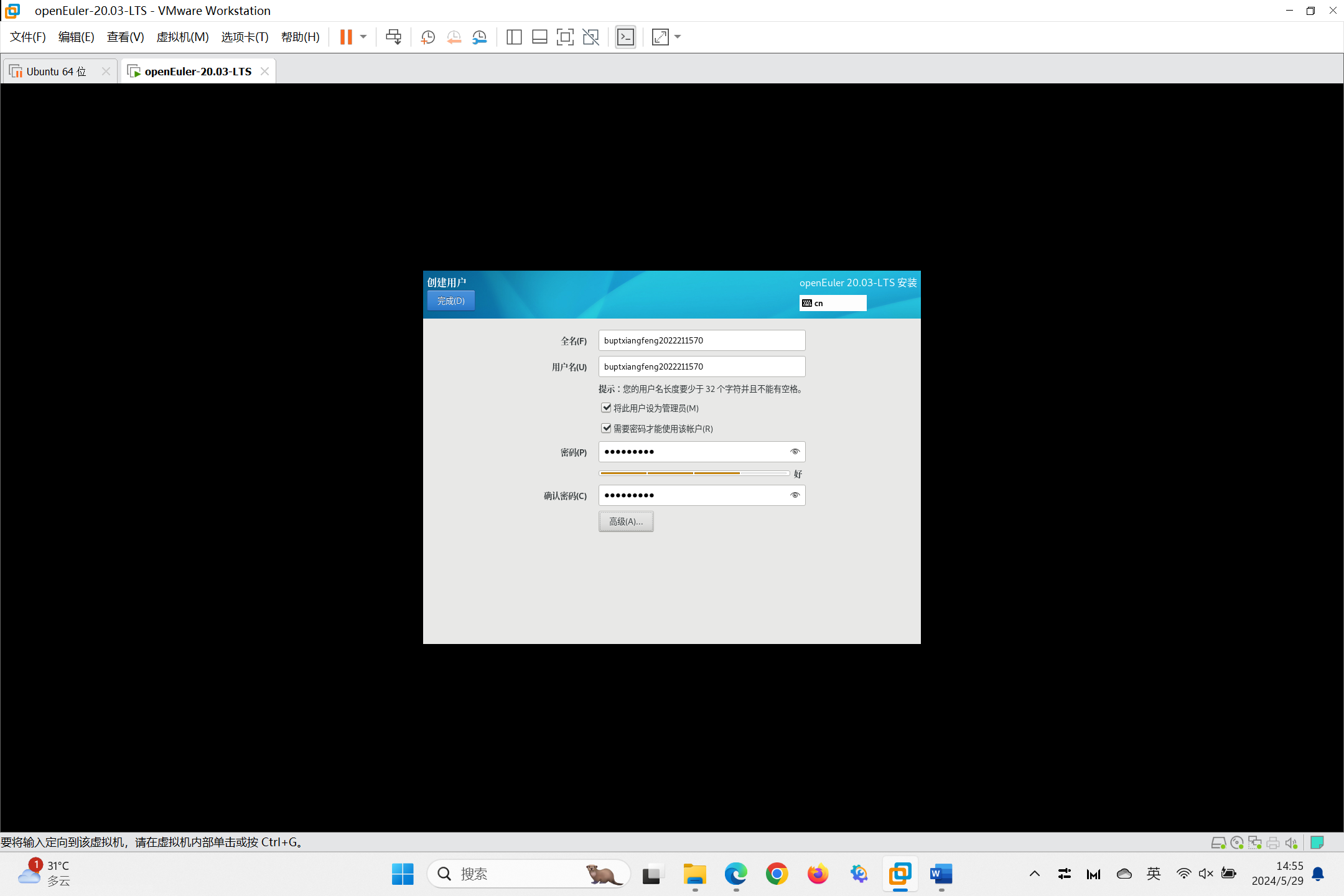
②进入虚拟机，开始安装

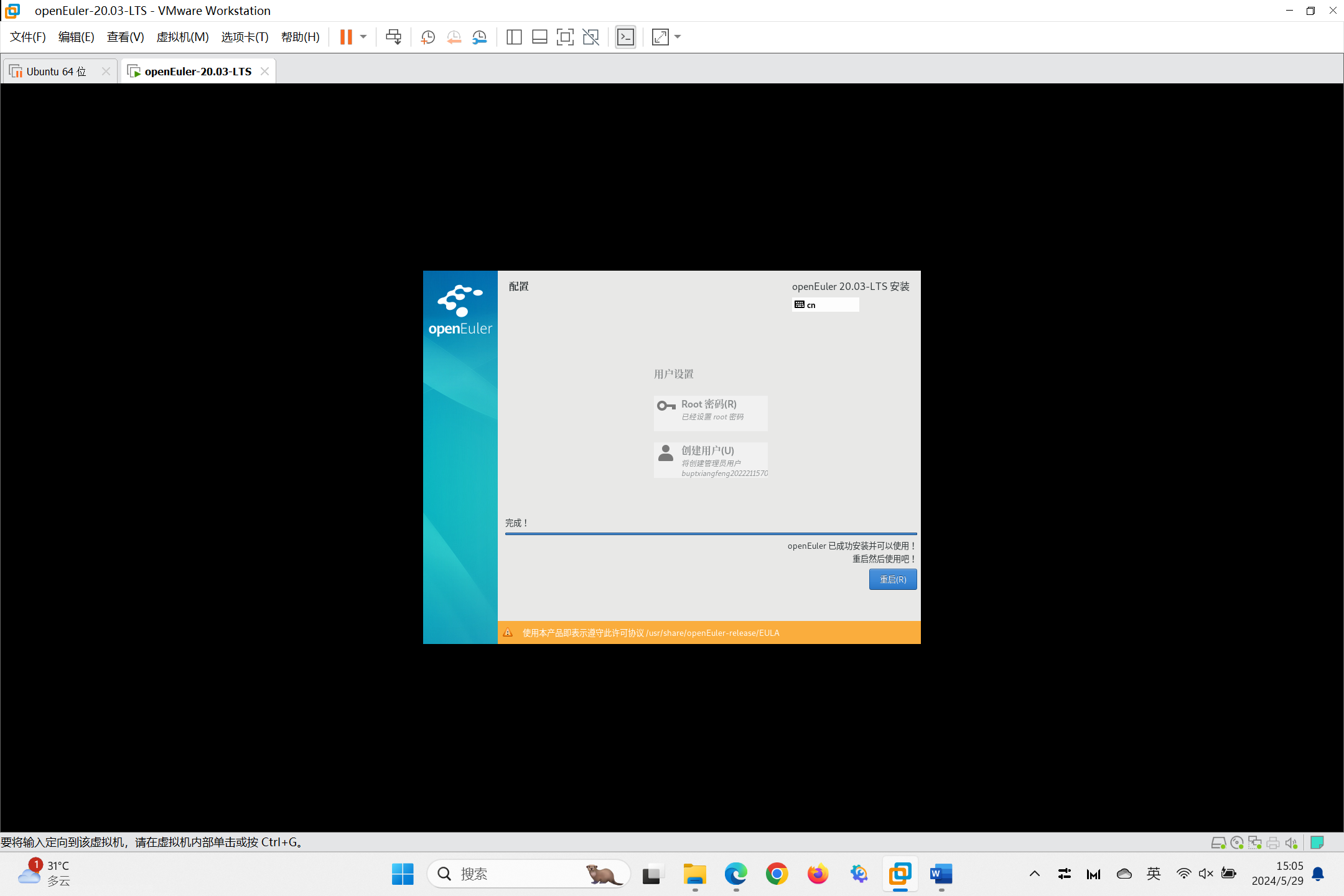












（3）安装桌面环境（gnome）以及terminal

①配置清华源

vim /etc/yum.repos.d/openEuler\_x86\_64.repo *#打开配置文件*

*# 添加如下内容*

[osrepo]

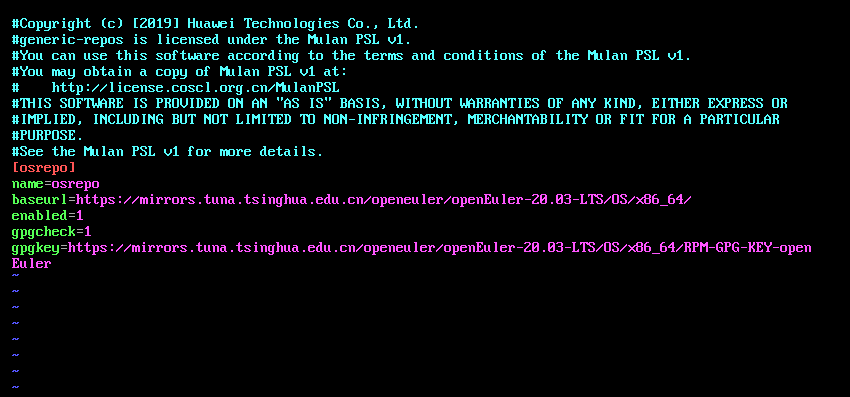
name=osrepo

baseurl=https://mirrors.tuna.tsinghua.edu.cn/openeuler/openEuler-20.03-LTS/OS/x86\_64/

enabled=1

gpgcheck=1

gpgkey=https://mirrors.tuna.tsinghua.edu.cn/openeuler/openEuler-20.03-LTS/OS/x86\_64/RPM-GPG-KEY-openEuler



②安装gnome、terminal

dnf install gnome-shell gdm gnome-session   #安装gnome及相关组件

dnf install gnome-terminal  #安装terminal

#设置开机自启动

systemctl enable gdm.service

systemctl set-default graphical.target

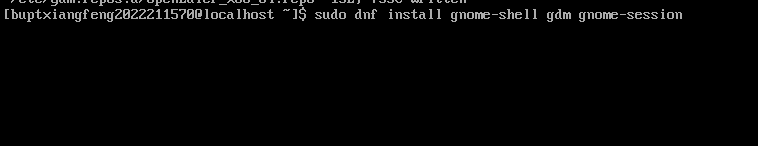
#补全丢失文件

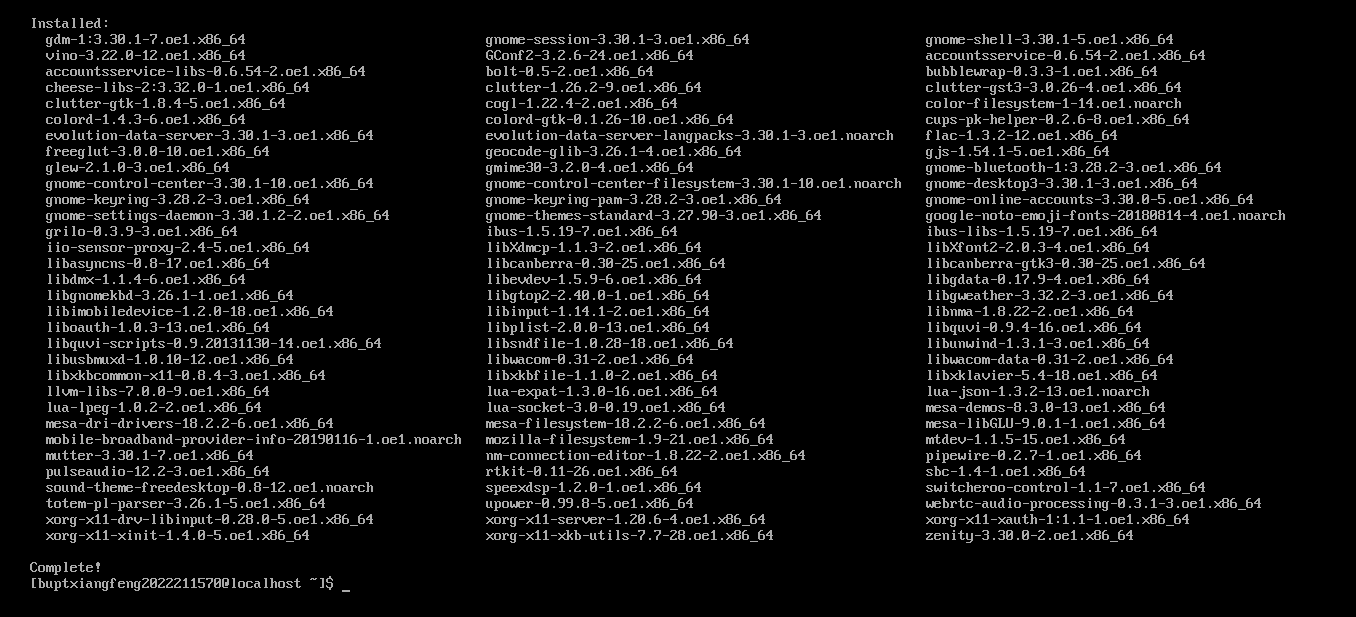
cd /tmp

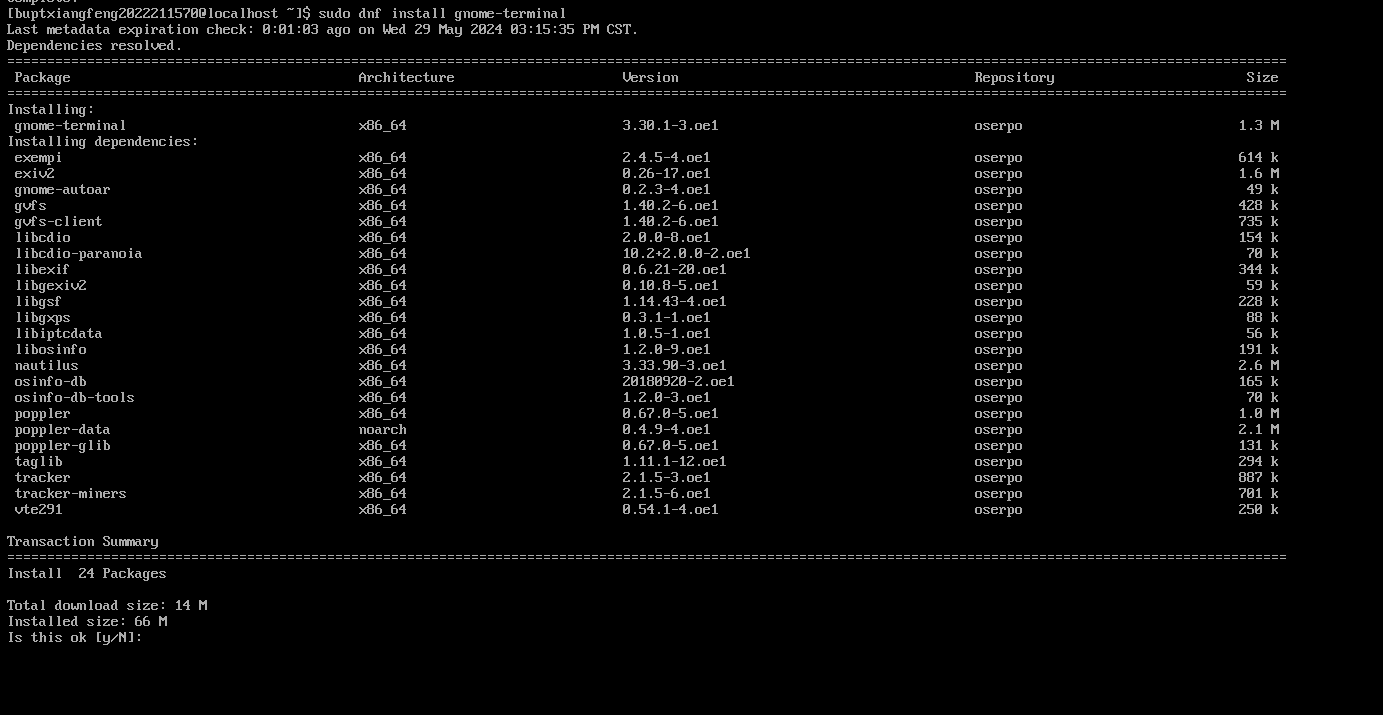
wget https://gitee.com/name1e5s/xsession/raw/master/Xsession

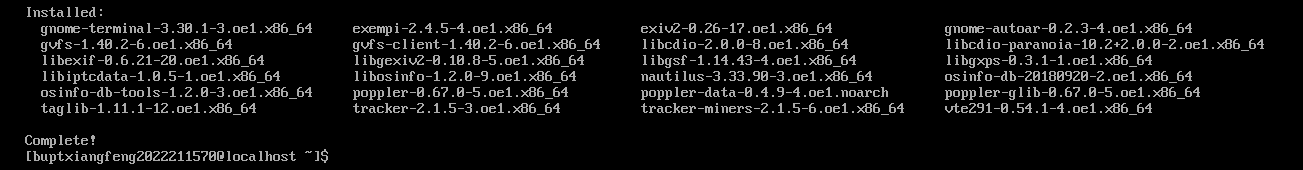
mv Xsession /etc/gdm/

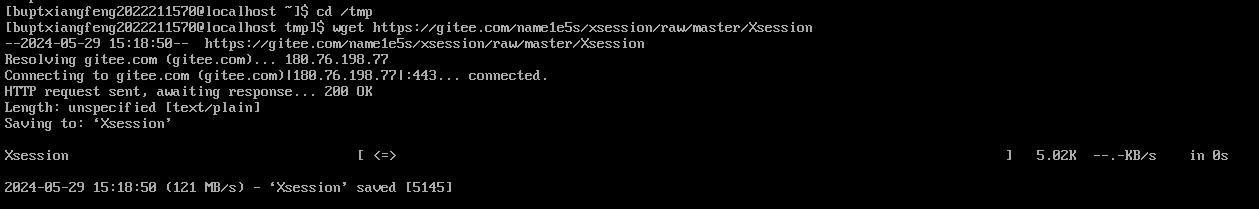
chmod 0777 /etc/gdm/Xsession











③gnome桌面安装成功



（4）执行uname -a指令、getconf PAGESIZE指令

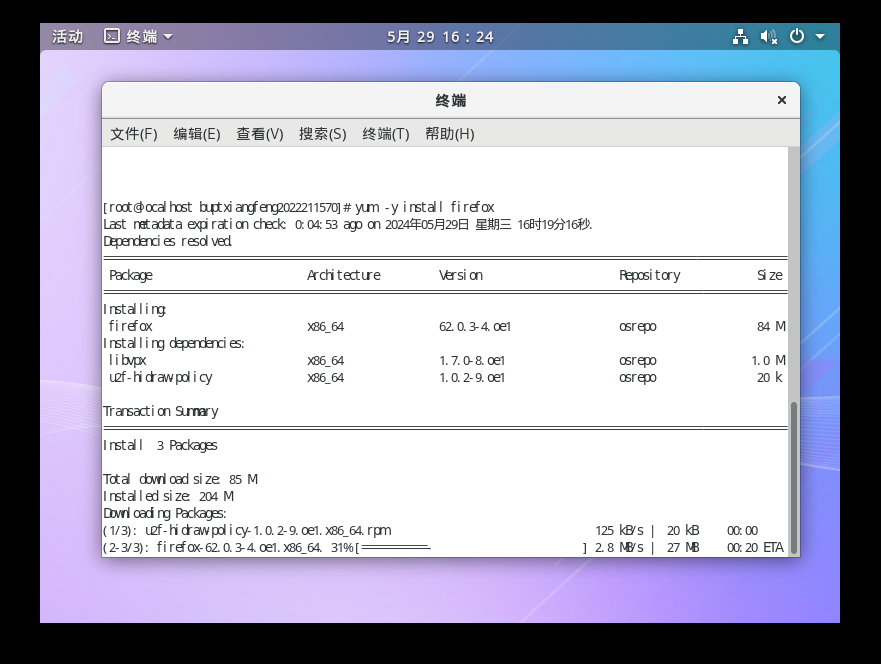
uname -a

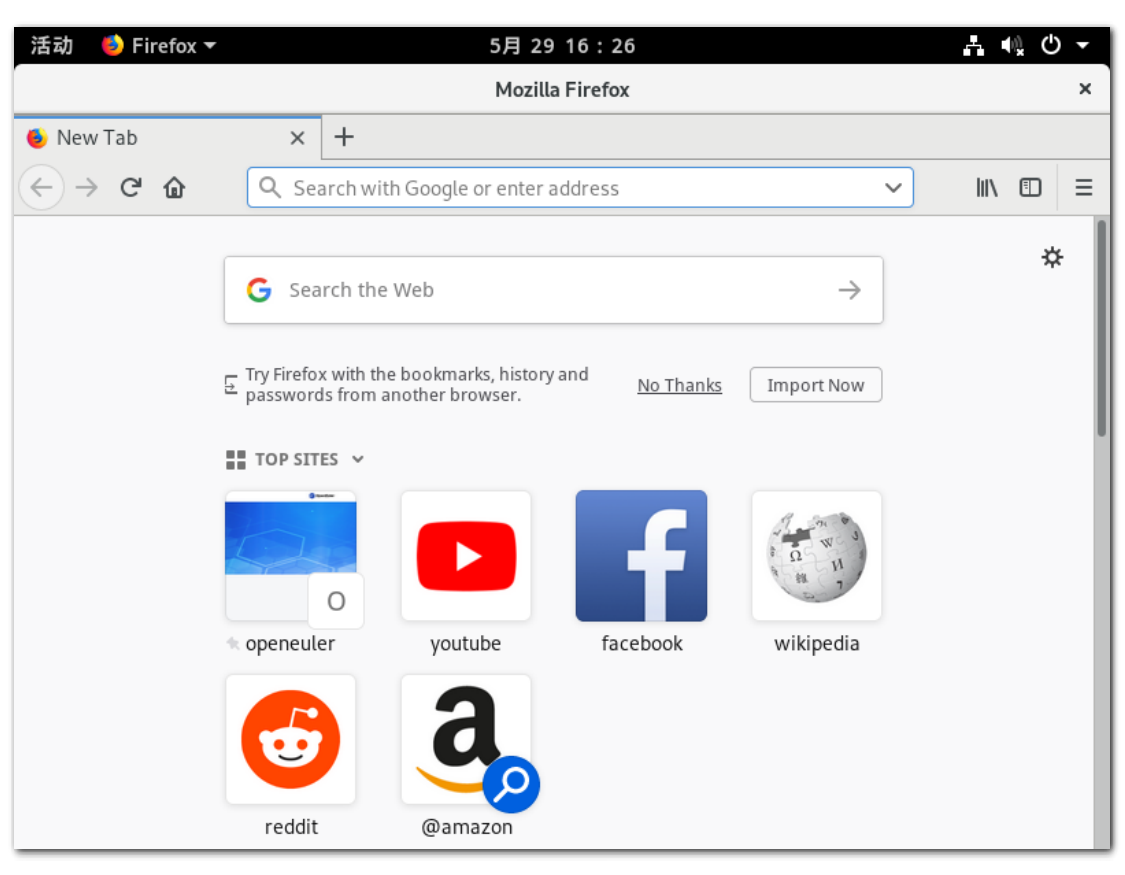
getconf PAGESIZE



（5）安装Firefox

yum -y install firefox *#顺便安装firefox*





2、内核编译安装

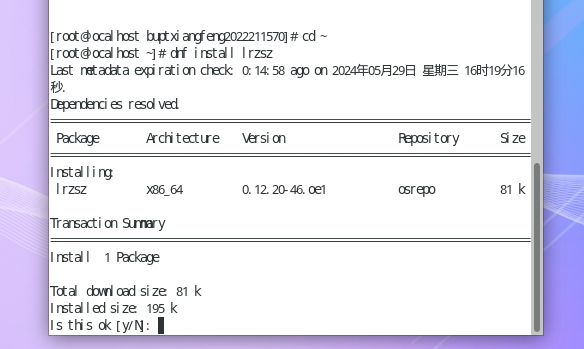
（1）系统备份

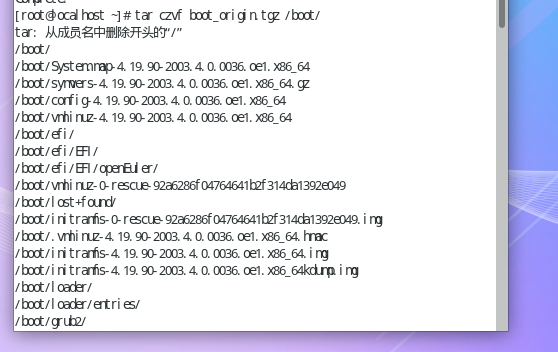
cd ~

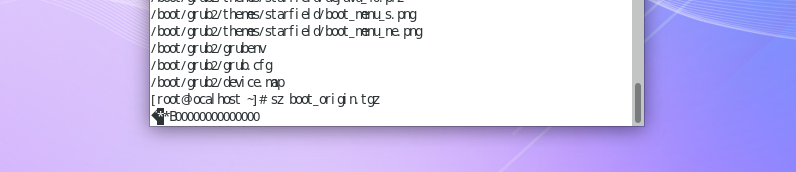
dnf install lrzsz      *# rz和sz可以在终端下很方便的传输文件*

tar czvf boot\_origin.tgz /boot/

sz boot\_origin.tgz    *# 将备份文件发送到本地*

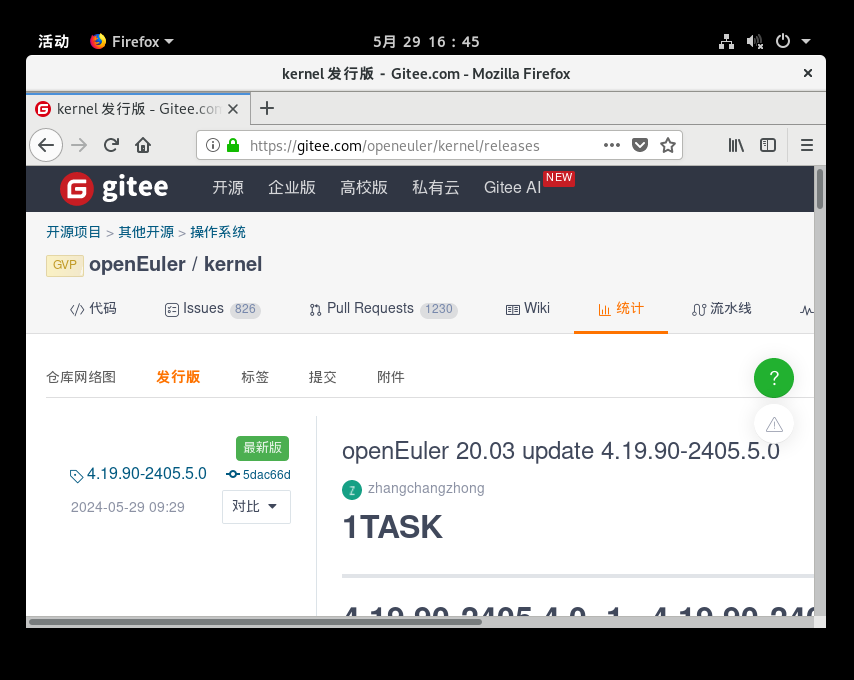




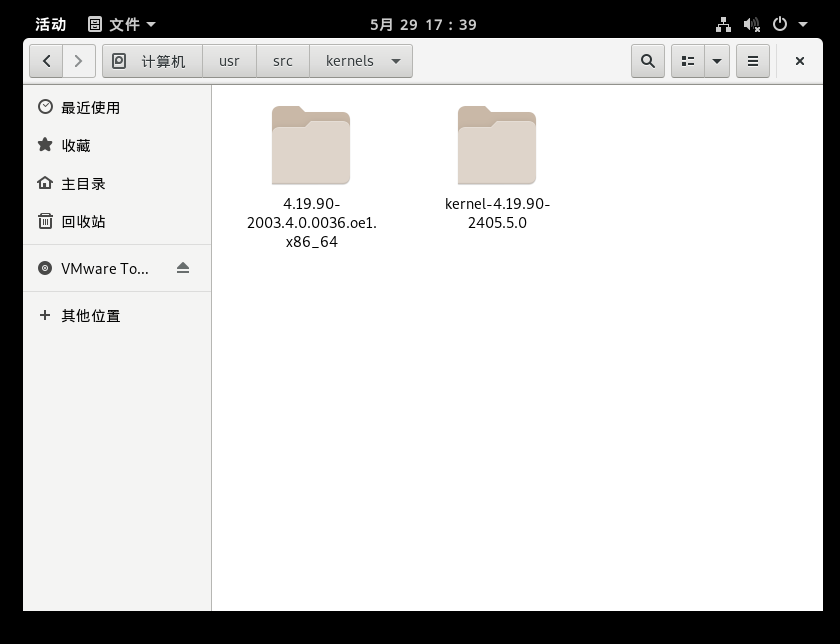


（2）内核源码下载

①在gitee仓库中下载openEuler内核压缩文件



②解压缩至/usr/src/kernels



（3）清理代码树

进入解压好的源码文件夹执行命令，清理过去内核编译产生的文件。

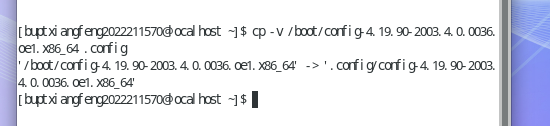
make mrproper



（4）生成内核配置文件.config

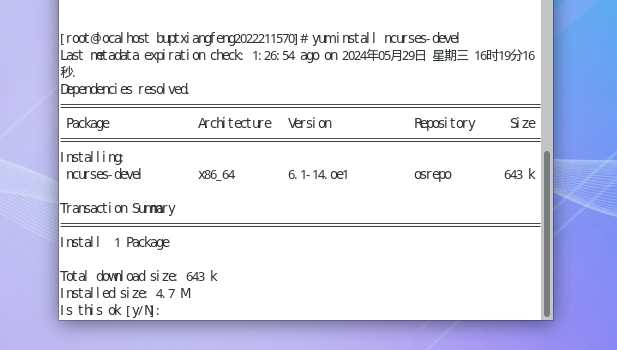
①先将将系统原配置文件拷贝过来

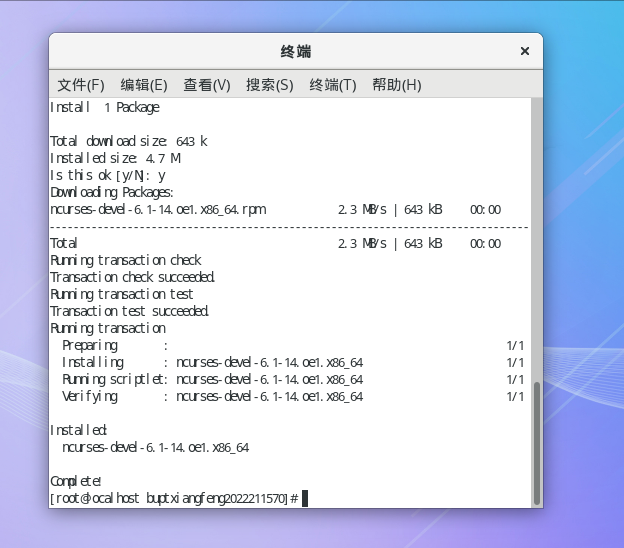
cp -v /boot/config-4.19.90-2003.4.0.0036.oe1.x86\_64.config



②执行依赖安装

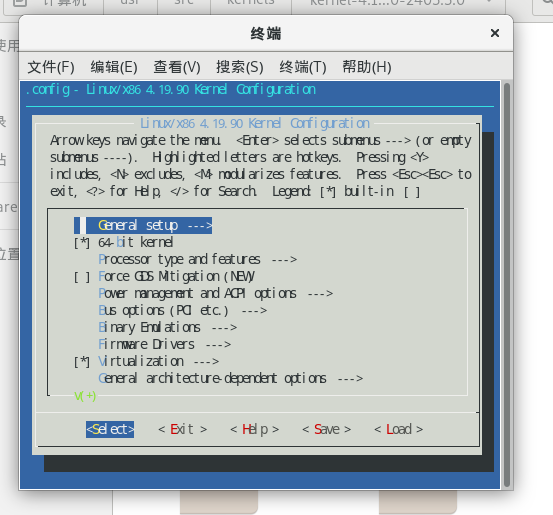
yum install ncurses-devel

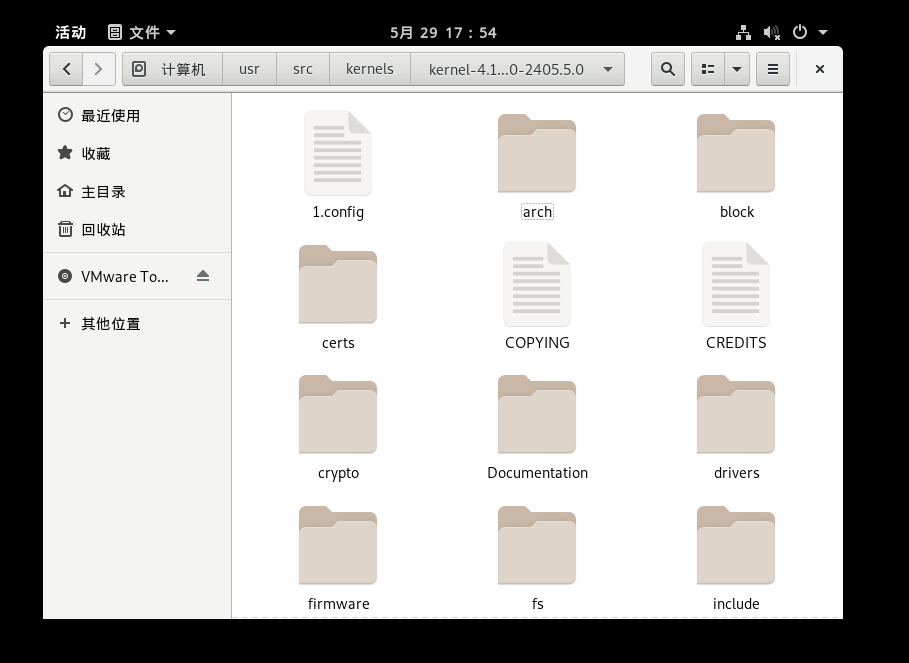


③对配置进行需要的更改

我没有改动直接默认配置，然后选择Save，生成了一个.config文件。

make menuconfig





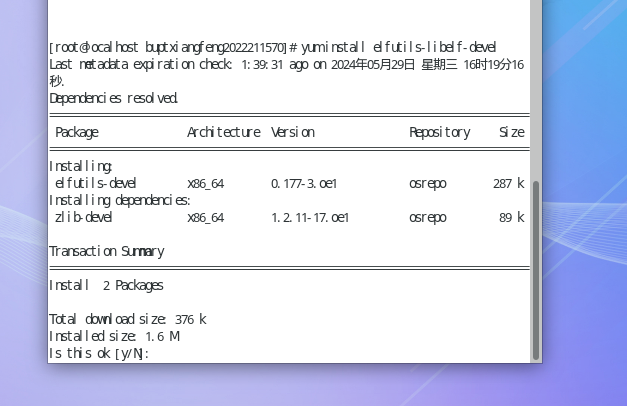
（5）内核编译及安装

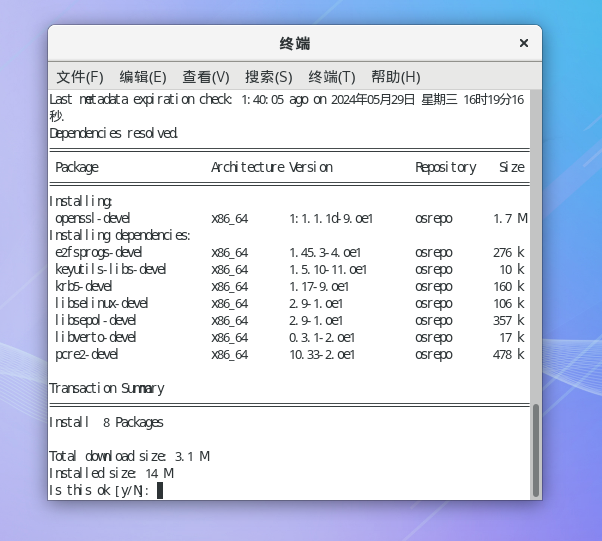
①安装所需组件

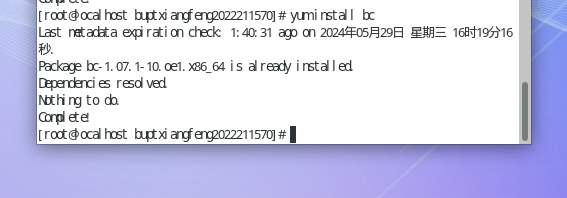
yum install elfutils-libelf-devel

yum install openssl-devel

yum install bc

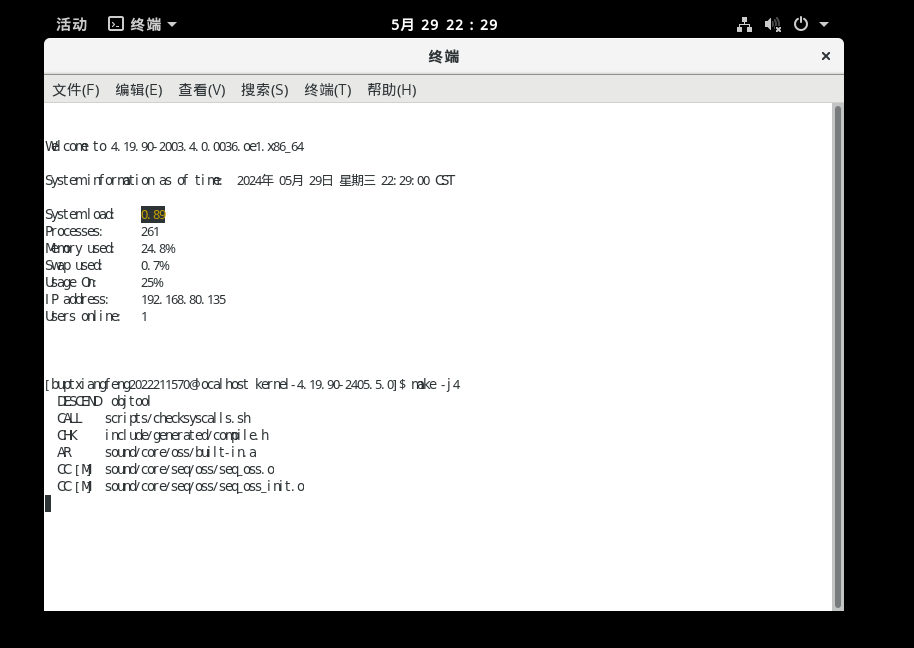






②开始编译

make -j4

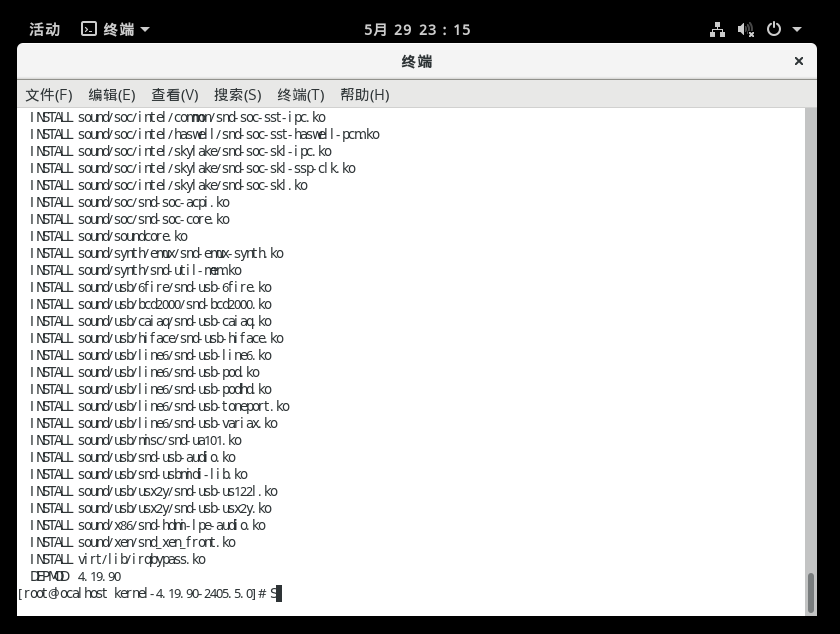




③安装模块

make modules\_install





④安装内核

make install



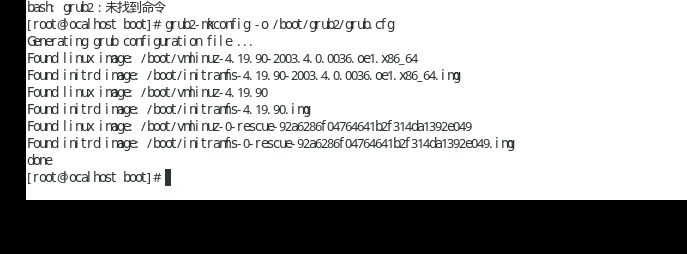
⑤在/boot目录下查看新安装的内核



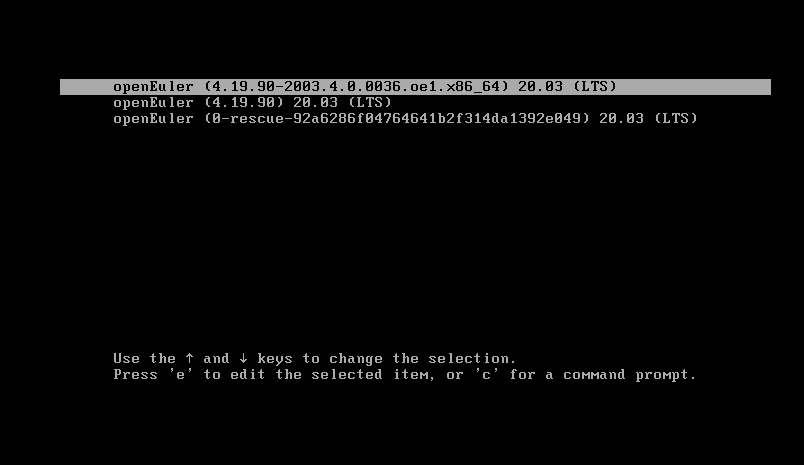
（6）更新引导

①下面的命令会根据 /boot/ 目录下的内核文件自动更新启动引导文件

grub2-mkconfig -o /boot/grub2/grub.cfg



②重启，箭头所指即为我们新安装的内核



（7）修改默认启动内核

①查看当前系统所有可用内核

cat /boot/grub2/grub.cfg |grep "menuentry "



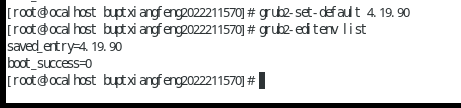
②查看当前默认启动内核

grub2-editenv list



③修改默认启动内核

grub2-set-default 4.19.90



④执行uname -a指令

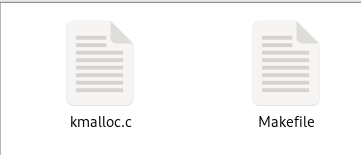
uname -a



（二）内存管理

1、使用 kmalloc 分配 1KB，8KB的内存，并打印指针地址

（1）创建kmalloc.c和Makefile文件



kmalloc.c

#include <linux/module.h>

#include <linux/slab.h>

MODULE\_LICENSE("GPL");

unsigned char \*kmallocmem1;

unsigned char \*kmallocmem2;

static int \_\_init mem\_module\_init(void)

{

    printk("Start kmalloc!\n");

    kmallocmem1 = (unsigned char\*)kmalloc(1024, GFP\_KERNEL);

    if (kmallocmem1 != NULL){

        printk(KERN\_ALERT "kmallocmem1 addr = %lx\n", (unsigned long)kmallocmem1);

    }else{

        printk("Failed to allocate kmallocmem1!\n");

    }

    kmallocmem2 = (unsigned char \*)kmalloc(8192, GFP\_KERNEL);

    if (kmallocmem2 != NULL){

        printk(KERN\_ALERT "kmallocmem2 addr = %lx\n", (unsigned long)kmallocmem2);

    }else{

        printk("Failed to allocate kmallocmem2!\n");

    }

    return 0;

}

static void \_\_exit mem\_module\_exit(void)

{

    kfree(kmallocmem1);

    kfree(kmallocmem2);

    printk("Exit kmalloc!\n");

}

module\_init(mem\_module\_init);

module\_exit(mem\_module\_exit);

Makefile

ifneq ($(KERNELRELEASE),)

    obj-m := kmalloc.o

else

    KERNELDIR ?=/usr/src/kernels/kernel-4.19.90-2405.5.0

    PWD := $(shell pwd)

default:

    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

.PHONY:clean

clean:

    -rm \*.mod.c \*.o \*.order \*.symvers \*.ko

（2）编译、加载模块

make

insmod kmalloc.ko

dmesg | tail -n 3

remmod kmalloc

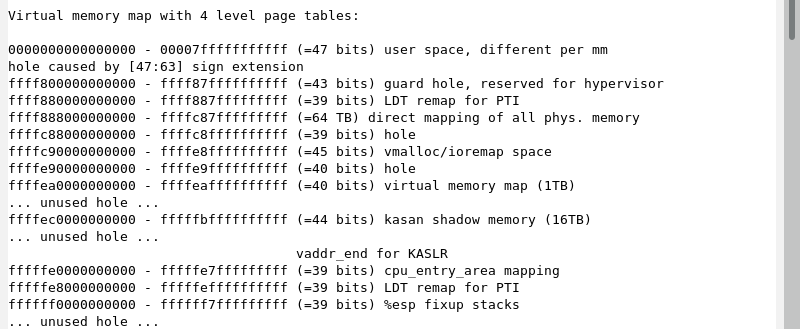
dmesg | tail -n 4

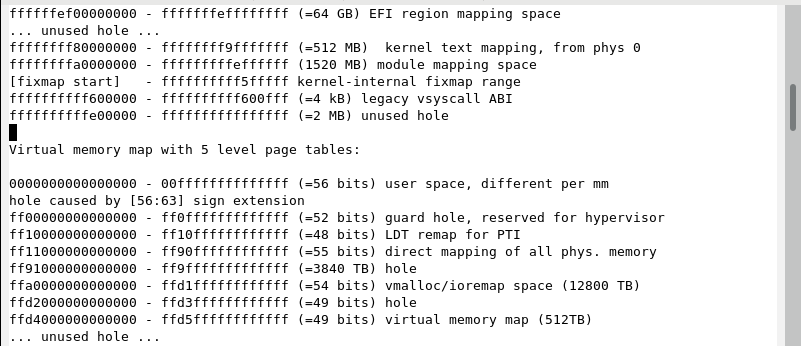


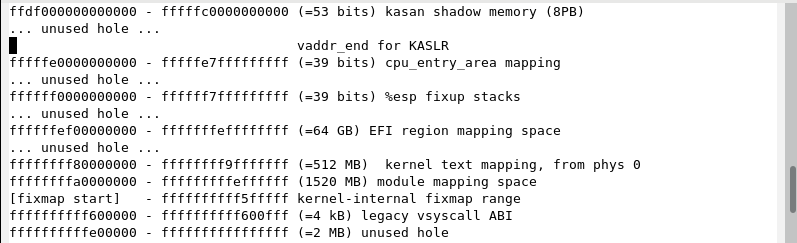
（3）查看内存布局

打开/usr/src/kernels/kernel-4.19.90-2405.5.0/Documentation/x86/x86\_64/

mm.txt文件。





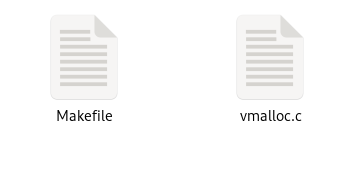


（4）结果分析

由运行结果可知，kmalloc 分配的内存地址，位于内核空间。

2、使用vmalloc分别分配8KB、1MB、64MB的内存，打印指针地址

（1）创建vmalloc.c和Makefile文件



vmalloc.c

#include <linux/module.h>

#include <linux/vmalloc.h>

MODULE\_LICENSE("GPL");

**unsigned** **char** \*vmallocmem1;

**unsigned** **char** \*vmallocmem2;

**unsigned** **char** \*vmallocmem3;

**static** **int** \_\_init mem\_module\_init(**void**)

{

    printk("Start vmalloc!\n");

    vmallocmem1 = (**unsigned** **char**\*)vmalloc(8192);

    if (vmallocmem1 != NULL){

        printk("vmallocmem1 addr = %lx\n", (**unsigned** **long**)vmallocmem1);

    }else{

        printk("Failed to allocate vmallocmem1!\n");

    }

    vmallocmem2 = (**unsigned** **char**\*)vmalloc(1048576);

    if (vmallocmem2 != NULL){

        printk("vmallocmem2 addr = %lx\n", (**unsigned** **long**)vmallocmem2);

    }else{

        printk("Failed to allocate vmallocmem2!\n");

    }

    vmallocmem3 = (**unsigned** **char**\*)vmalloc(67108864);

    if (vmallocmem3 != NULL){

        printk("vmallocmem3 addr = %lx\n", (**unsigned** **long**)vmallocmem3);

    }else{

        printk("Failed to allocate vmallocmem3!\n");

    }

    return 0;

}

**static** **void** \_\_exit mem\_module\_exit(**void**)

{

    vfree(vmallocmem1);

    vfree(vmallocmem2);

    vfree(vmallocmem3);

    printk("Exit vmalloc!\n");

}

module\_init(mem\_module\_init);

module\_exit(mem\_module\_exit);

Makefile

ifneq ($(KERNELRELEASE),)

    obj-m := vmalloc.o

else

    KERNELDIR ?= /usr/src/kernels/kernel-4.19.90-2405.5.0

    PWD := $(shell pwd)

default:

    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

.PHONY:clean

clean:

    -rm \*.mod.c \*.o \*.order \*.symvers \*.ko

（2）编译、加载模块

make

insmod vmalloc.ko

dmesg | tail -n 4

remmod vmalloc

dmesg | tail -n 5

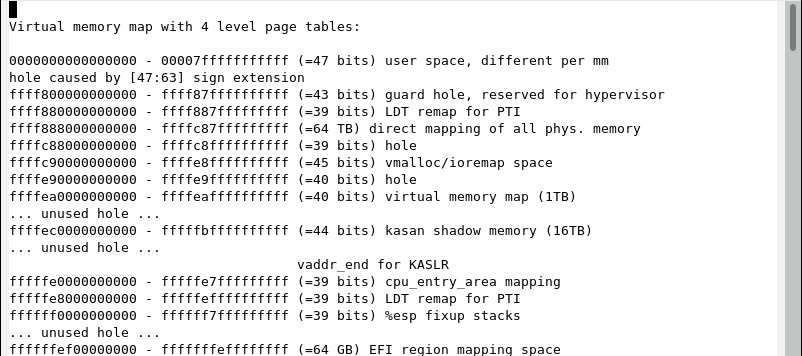


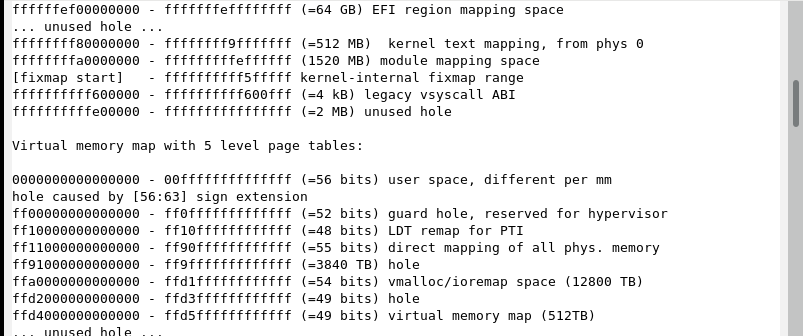


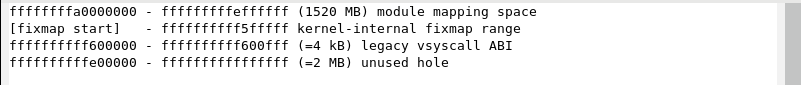
（3）查看内存布局

打开/usr/src/kernels/kernel-4.19.90-2405.5.0/Documentation/x86/x86\_64/

mm.txt文件。







（4）结果分析

由运行结果可知，vmalloc 分配的内存地址，位于内核空间。

（三）内核时间管理

1、调用内核时钟接口打印当前时间

（1）创建current\_time.c和Makefile文件



current\_time.c

#include <linux/module.h>

#include <linux/time.h>

#include <linux/rtc.h>

MODULE\_LICENSE(“GPL”);

**struct** timeval tv;

**struct** rtc\_time tm;

**static** **int** \_\_init currenttime\_init(**void**)

{

**int** year, mon, day, hour, min, sec;

printk(“Start current\_time module…\n”);

do\_gettimeofday(&tv);

rtc\_time\_to\_tm(tv.tv\_sec, &tm);

year = tm.tm\_year + 1900;

mon = tm.tm\_mon + 1;

day = tm.tm\_mday;

hour = tm.tm\_hour + 8;

min = tm.tm\_min;

sec = tm.tm\_sec;

printk(“Current time: %d-%02d-%02d %02d:%02d:%02d\n”, year, mon, day, hour, min, sec);

return 0;

}

**static** **void** \_\_exit currenttime\_exit(**void**)

{

printk(“Exit current\_time module…\n”);

}

module\_init(currenttime\_init);

module\_exit(currenttime\_exit);

Makefile

ifneq ($(KERNELRELEASE),)

obj-m := current\_time.o

else

KERNELDIR ?= /usr/src/kernels/kernel-4.19.90-2405.5.0

PWD := $(shell pwd)

default:

$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

.PHONY:clean

clean:

-rm \*.mod.c \*.o \*.order \*.symvers \*.ko

（2）编译运行

make

insmod current\_time.ko

dmesg | tail -n 2

remmod current\_time

dmesg | tail -n 3

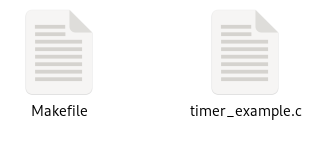


（3）结果分析

成功在在屏幕上打印出格式化的时间、日期，并正确地加载和卸载。

2、编写timer，在特定时刻打印 hello,world

（1）创建timer\_example.c和Makefile文件



timer\_example.c

#include <linux/module.h>

#include <linux/timer.h>

MODULE\_LICENSE("GPL");

**struct** timer\_list timer;

**void** print(**struct** timer\_list \*timer)

{

    printk("hello,world!\n");

}

**static** **int** \_\_init timer\_init(**void**)

{

    printk("Start timer\_example module...\n");

    timer.expires = jiffies + 10 \* HZ;

    timer.function = print;

    add\_timer(&timer);

    return 0;

}

**static** **void** \_\_exit timer\_exit(**void**)

{

    printk("Exit timer\_example module...\n");

}

module\_init(timer\_init);

module\_exit(timer\_exit);

Makefile

ifneq ($(KERNELRELEASE),)

    obj-m := timer\_example.o

else

    KERNELDIR ?= /usr/src/kernels/kernel-4.19.90-2405.5.0

    PWD := $(shell pwd)

default:

    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

.PHONY:clean

clean:

    -rm \*.mod.c \*.o \*.order \*.symvers \*.ko

（2）编译运行

make

insmod timer\_example.ko

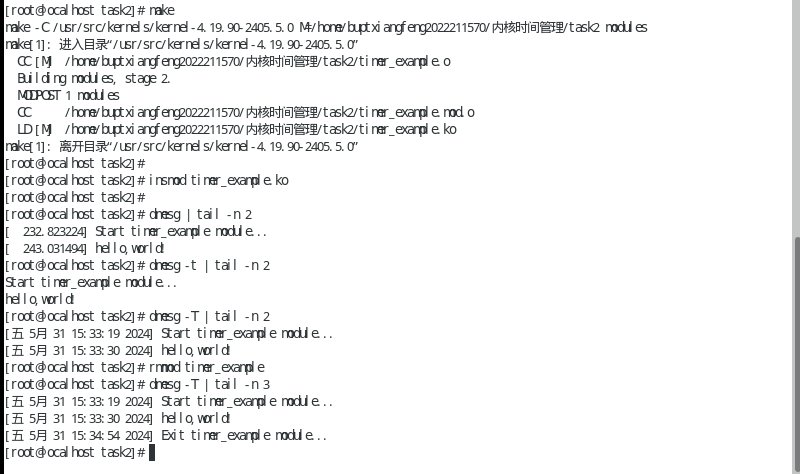
dmesg | tail -n 2

dmesg -t | tail -n 2

dmesg -T | tail -n 2

remmod timer\_example

dmesg -T | tail -n 3

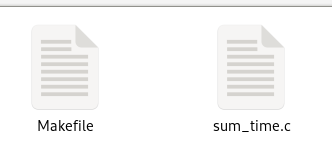


（3）结果分析

加载该内核模块10秒后打印 “hello,world!”，因为定时器执行了定时操作。合理使用定时器，可以使工作在指定时间点上执行，我们只需要执行一些初始化工作，设置一个超时时间，指定超时发生后执行的函数，然后激活定时器就可以了。指定的函数在定时器到期时自动执行。

3、调用内核时钟接口，监控累加计算代码的运行时间

（1）创建sum\_time.c和Makefile文件



sum\_time.c

#include <linux/module.h>

#include <linux/time.h>

MODULE\_LICENSE("GPL");

#define NUM 100000

**struct** timeval tv;

**static** **long** sum(**int** num)

{

**int** i;

**long** total = 0;

    for (i = 1; i <= NUM; i++)

        total = total + i;

    printk("The sum of 1 to %d is: %ld\n", NUM, total);

    return total;

}

**static** **int** \_\_init sum\_init(**void**)

{

**int** start;

**int** start\_u;

**int** end;

**int** end\_u;

**long** time\_cost;

**long** s;

    printk("Start sum\_time module...\n");

    do\_gettimeofday(&tv);

    start = (**int**)tv.tv\_sec;

    start\_u = (**int**)tv.tv\_usec;

    printk("The start time is: %d s %d us \n", start, start\_u);

    s = sum(NUM);

    do\_gettimeofday(&tv);

    end = (**int**)tv.tv\_sec;

    end\_u = (**int**)tv.tv\_usec;

    printk("The end time is: %d s %d us \n", end, end\_u);

    time\_cost = (end - start) \* 1000000 + end\_u - start\_u;

    printk("The cost time of sum from 1 to %d is: %ld us \n", NUM, time\_cost);

    return 0;

}

**static** **void** \_\_exit sum\_exit(**void**)

{

    printk("Exit sum\_time module...\n");

}

module\_init(sum\_init);

module\_exit(sum\_exit);

Makefile

ifneq ($(KERNELRELEASE),)

    obj-m := sum\_time.o

else

    KERNELDIR ?= /usr/src/kernels/kernel-4.19.90-2405.5.0

    PWD := $(shell pwd)

default:

    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

.PHONY:clean

clean:

    -rm \*.mod.c \*.o \*.order \*.symvers \*.ko

（2）编译运行

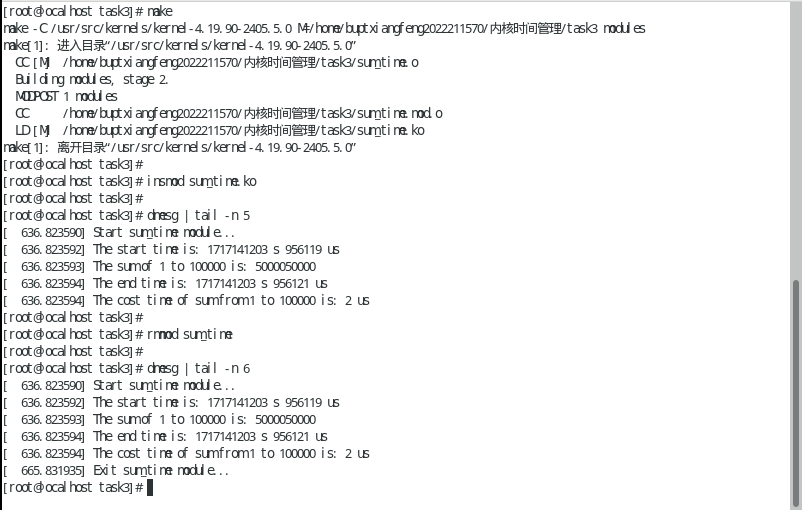
make

insmod sum\_time.ko

dmesg | tail -n 5

remmod sum\_time

dmesg | tail -n 6



（3）结果分析

由程序运行结果可以看出，从 1 到 100000 的累加和所花时间是2 us。

四、问题及解决方案

|  |  |
| --- | --- |
| 问题 | 解决方案 |
| 安装gnome后，登录后无法进入桌面，重新安装后，可正常进入。 | [在 openEuler 上安装桌面环境 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/229861153)  [实验一、openEuler操作系统安装与内核编译\_安装openeuler操作系统实验报告-CSDN博客](https://blog.csdn.net/qq_46744173/article/details/122198126) |
| 内存管理、内核时间管理代码参考 | [NWPU\_OS教学/OpenEuler\_实验 (gitee.com)](https://gitee.com/lin-man/open-euler_sh) |
| 实验过程参考 | [OpenEuler实验\_本次实验服务器已完成内核编译(openeuler 4.19.08),可直接开始实验-CSDN博客](https://blog.csdn.net/qq_46744173/article/details/122199462) |

五、实验总结

此次实验，不仅增强了我的技能，也使我对openEuler操作系统的安装与编译、内存管理以及内核时间管理有了较为深刻的理解，为我未来的学习和工作奠定了坚实基础。