

**实 验 报 告**

|  |  |
| --- | --- |
| 中文标题 | 操作系统openEuler实验 |
| 姓名 | 李昊伦 |
| 学号 | 2023211595 |
| 班级 | 2023211805 |
| 贡献度 | 100% |
| 指导教师 | 李灵慧 |

**目 录**

[**操作系统实验二：openEuler实验 3**](#_Toc198315254)

[**1 实验内容与环境 3**](#_Toc198315255)

[1.1 实验内容 3](#_Toc198315256)

[1.2 实验分工 3](#_Toc198315257)

[1.3 实验环境 3](#_Toc198315258)

[**2 实验过程 3**](#_Toc198315259)

[2.1 openEuler操作系统安装 3](#_Toc198315260)

[2.2 内核更新 20](#_Toc198315261)

[2.3 内核模块编程 31](#_Toc198315262)

[2.4 内存管理 33](#_Toc198315263)

[2.5 内核时间管理 41](#_Toc198315264)

[**3 问题及解决方案 47**](#_Toc198315265)

[**4 实验总结 48**](#_Toc198315266)

操作系统实验二：openEuler实验

# 1 实验内容与环境

## 1.1 实验内容

完成openEuler操作系统的安装，完成内核更新，在此基础上，增加完成其他功能。本次实验已增添完成：内核模块编程、内存管理、内核时间管理。

## 1.2 实验分工

姓名：李昊伦 班级：2023211805 学号：2023211595 分工：100%

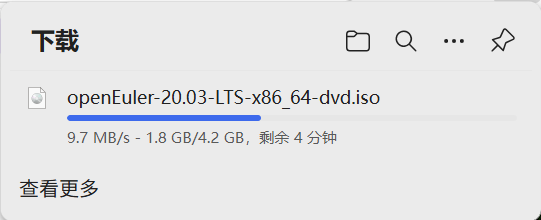
## 1.3 实验环境

虚拟机Vmware和openEuler操作系统。

# 2 实验过程

## 2.1 openEuler操作系统安装

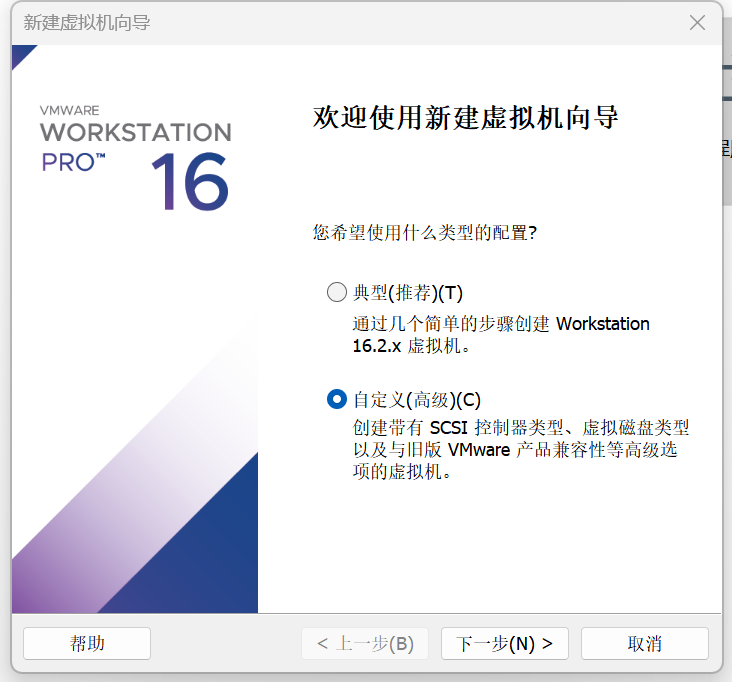
1.下载OpenEular镜像

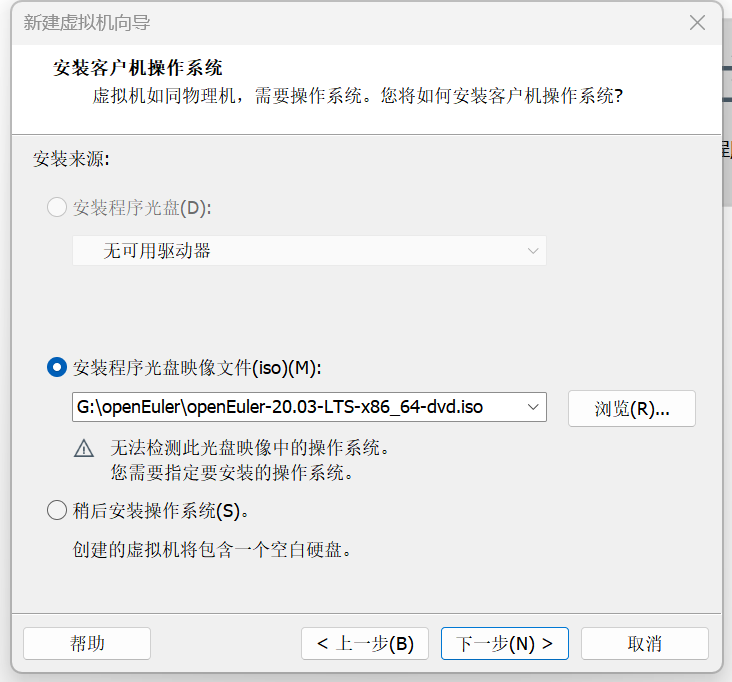


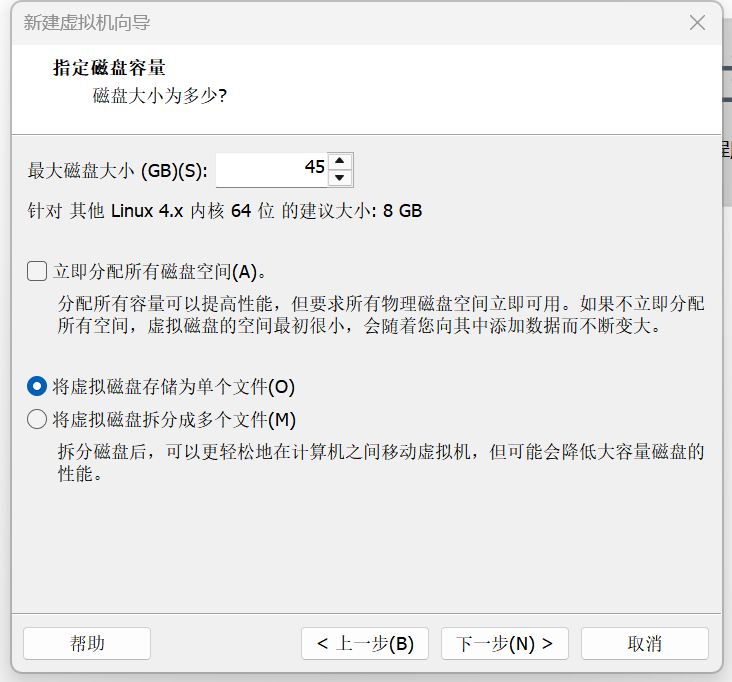
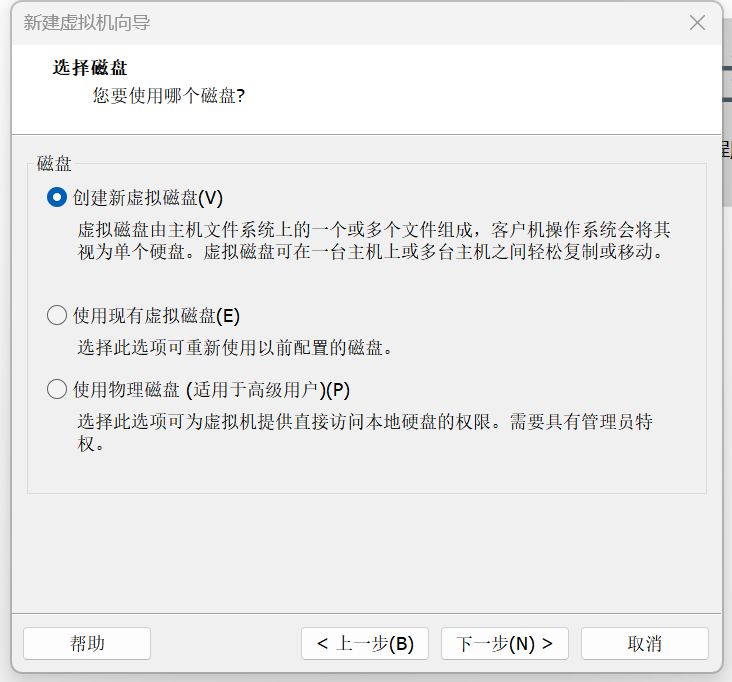
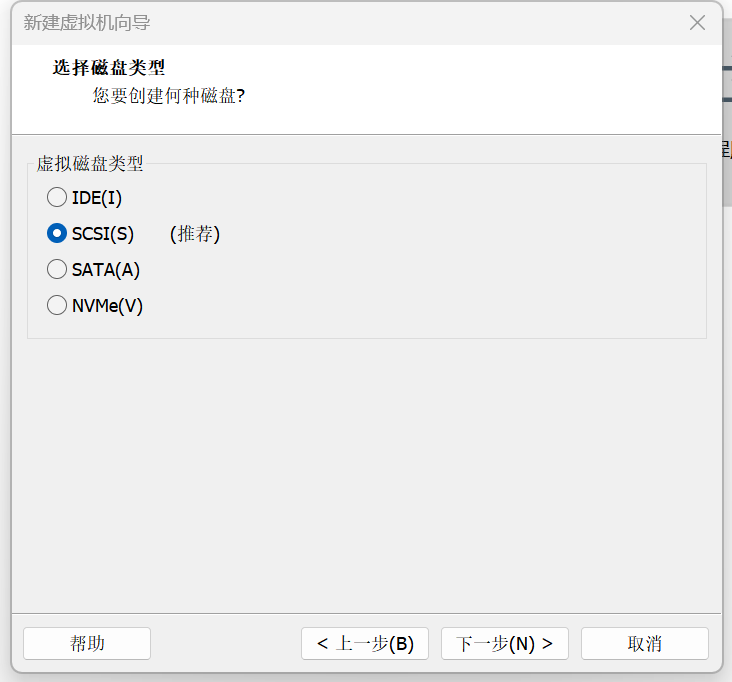
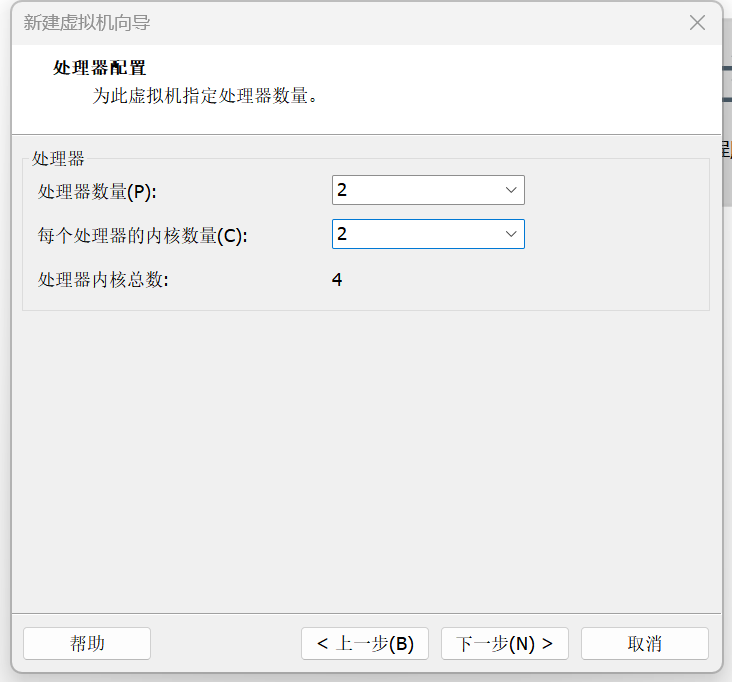
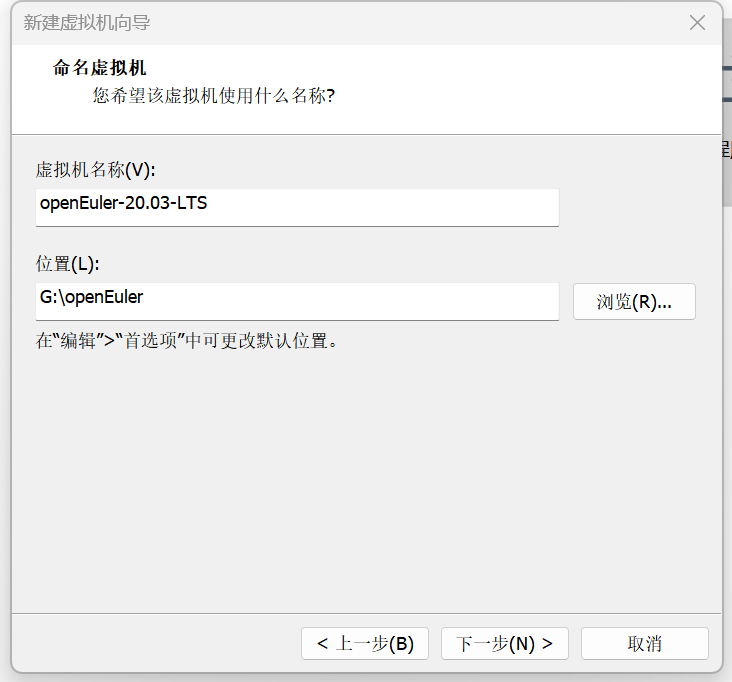
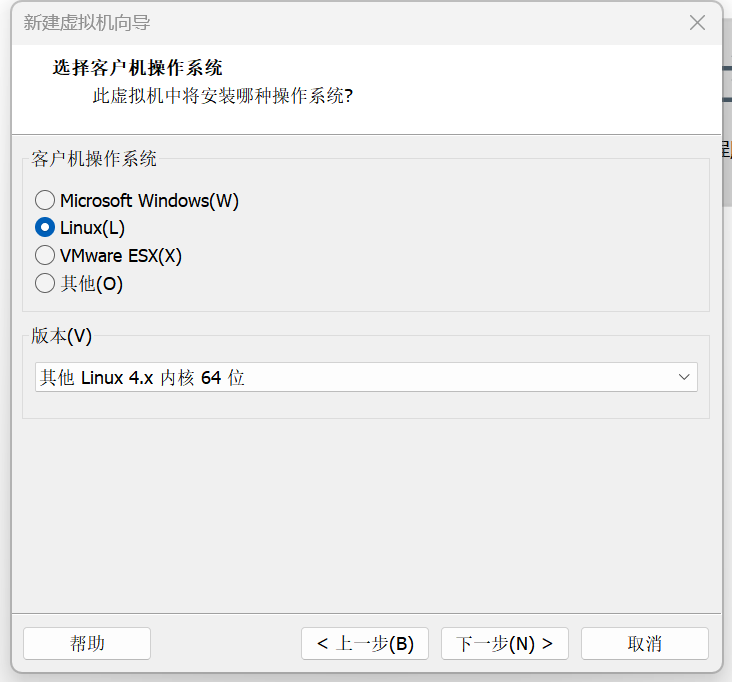


2.安装到虚拟机

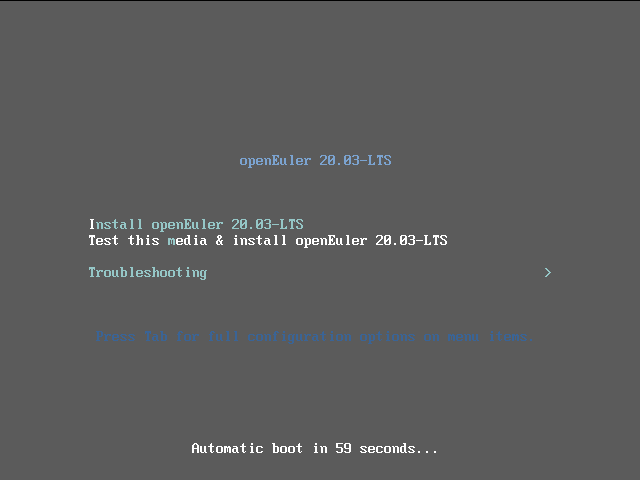
（1）新建虚拟机







（2）进入虚拟机，开始安装

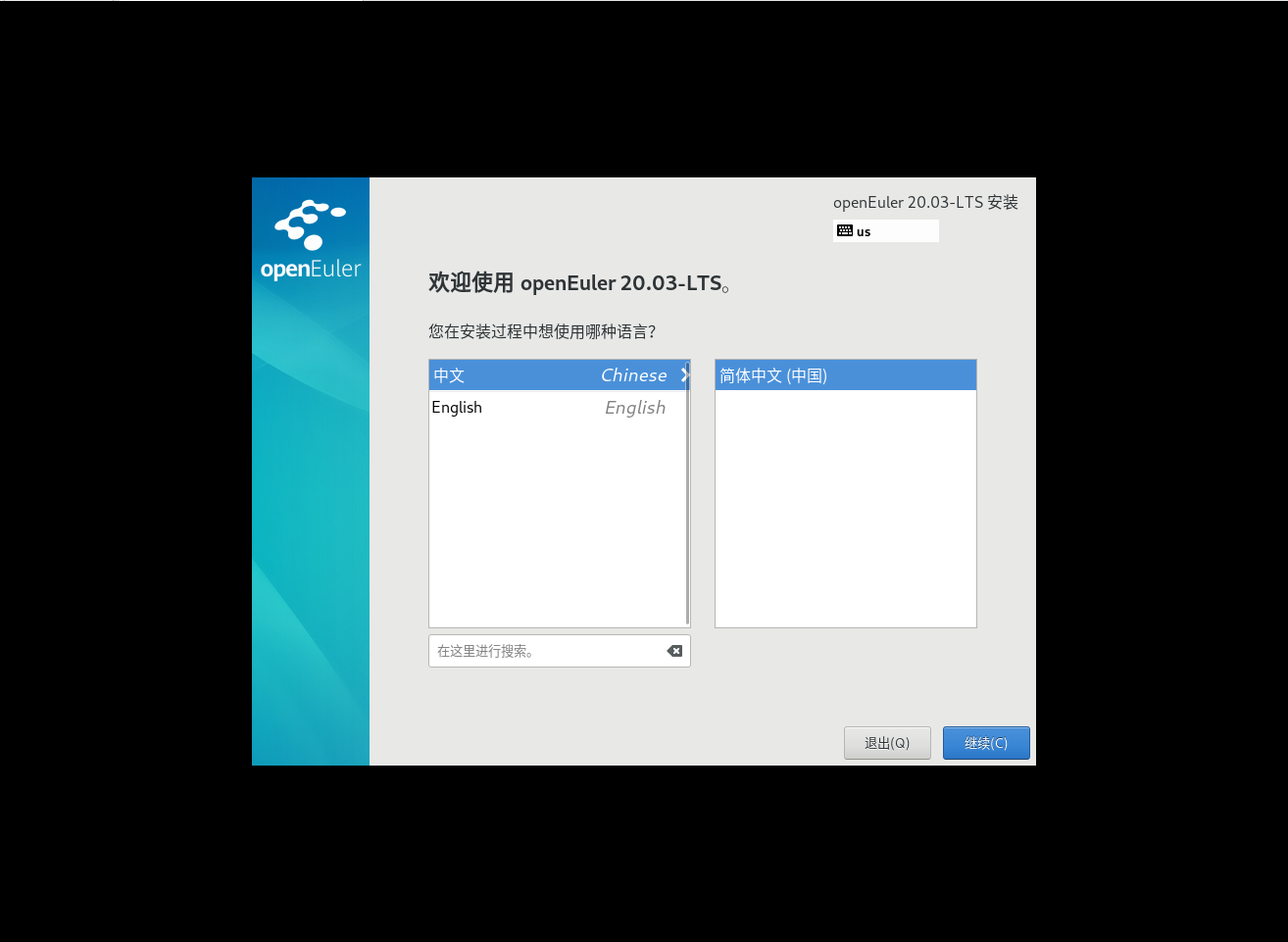


①等待安装检查

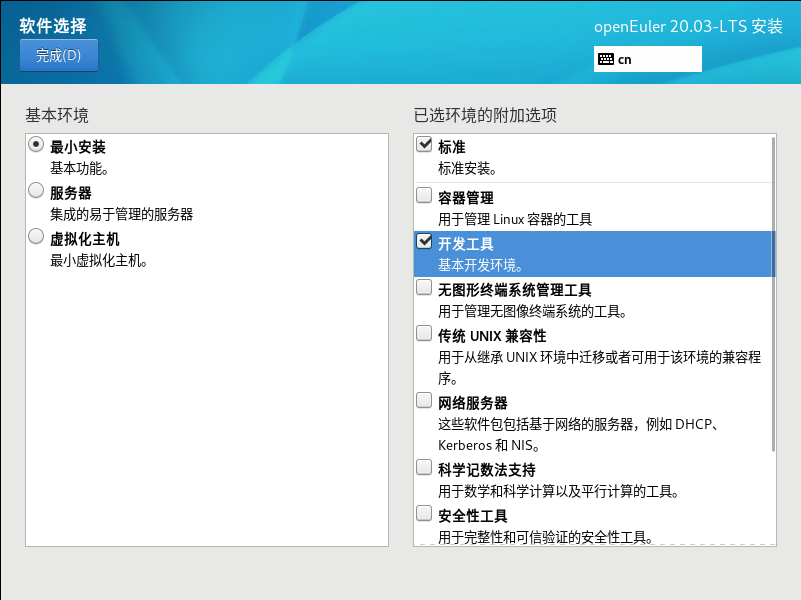




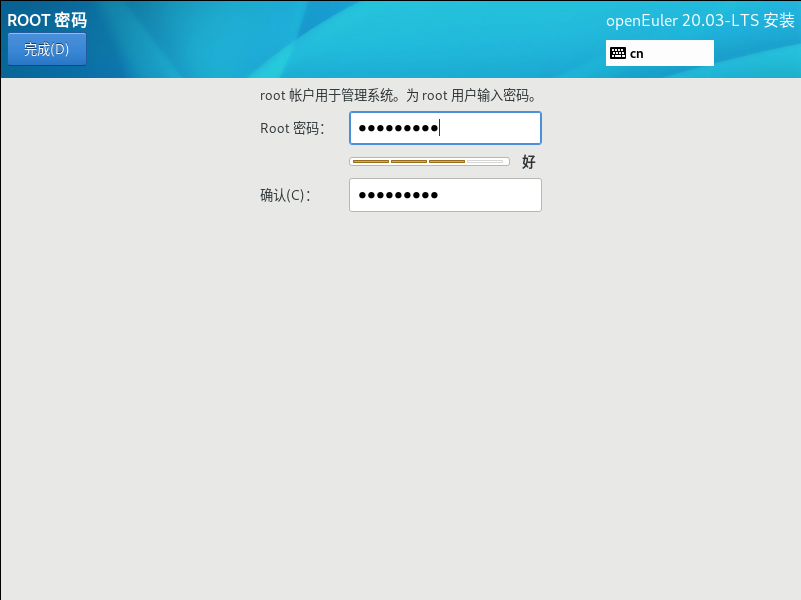
②进行语言设置：选择中文



③进行软件选择：这里我们选择最小安装，附加选项为标准和开发工具

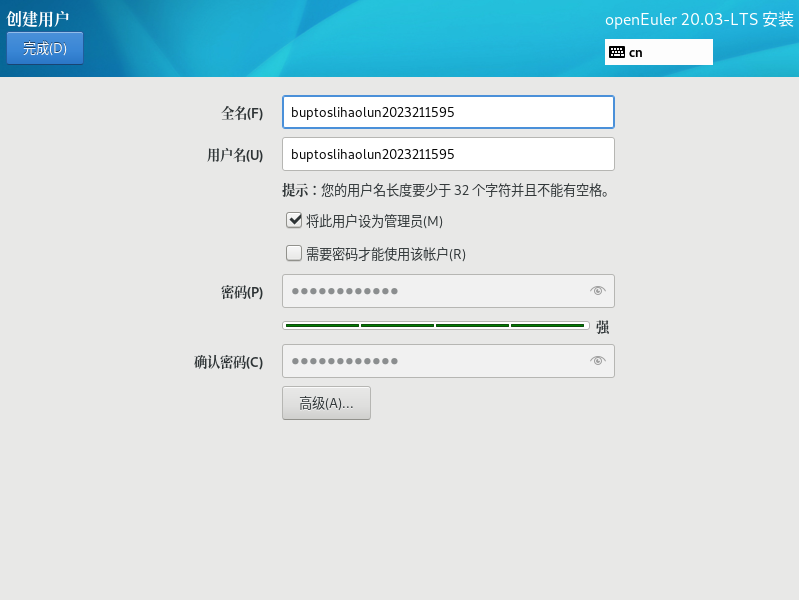


④设置ROOT密码：12Lhl0408



⑤创建用户：设置用户名为buptoslihaolun2023211595

设置密码为123456@Lhl04，这里我设置了不需要密码即可登录账户



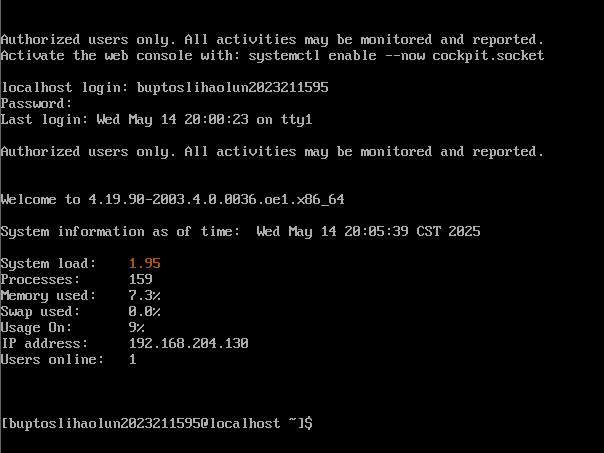
⑥安装完成：进行重启



⑦随后进入系统。

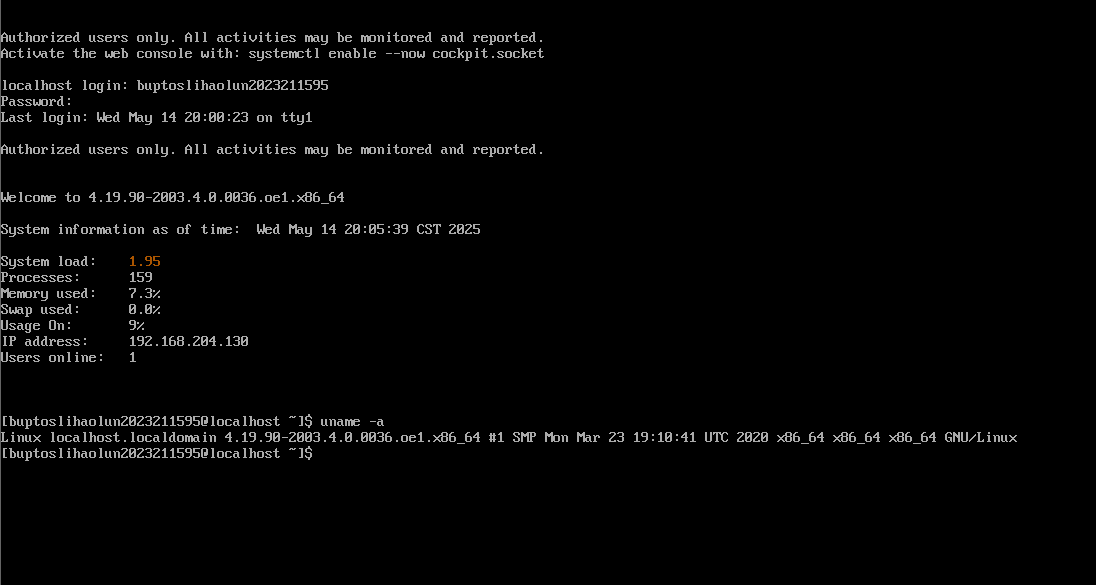


⑧输入刚刚设置的账号密码进行登录

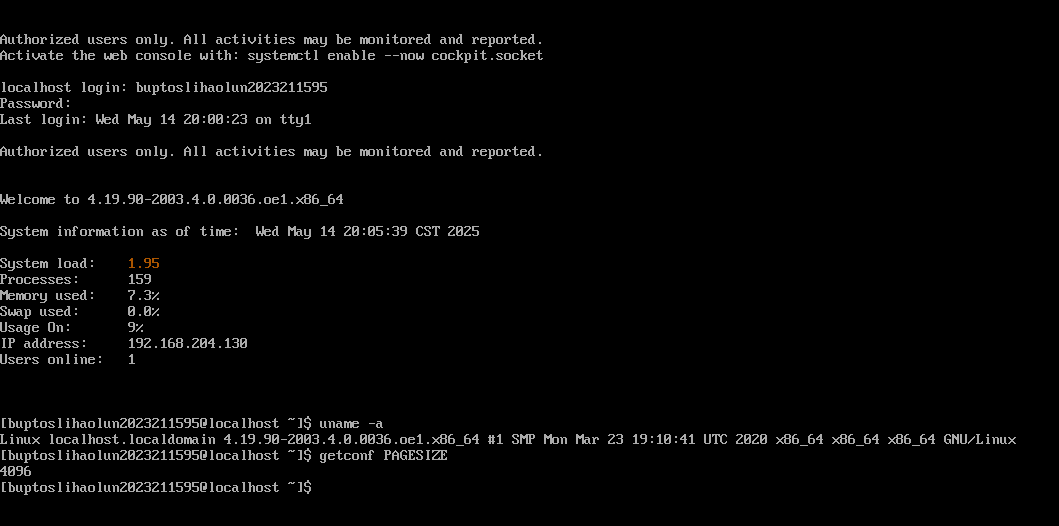


（3）执行相应指令

①执行uname -a指令



②执行getconf PAGESIZE指令，查看openEuler的分页大小，为4096



（4）安装桌面环境（gnome）以及terminal

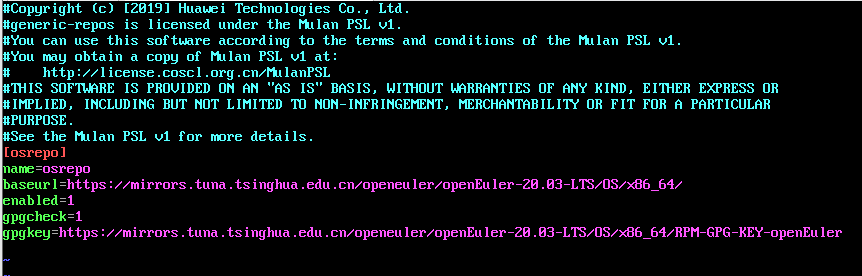
①配置清华源

先输入以下指令打开配置文件

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

打开后按下键盘i键进入编辑模式，添加以下内容。添加完成后按下esc结束编辑模式。输入:wq!保存并退出。

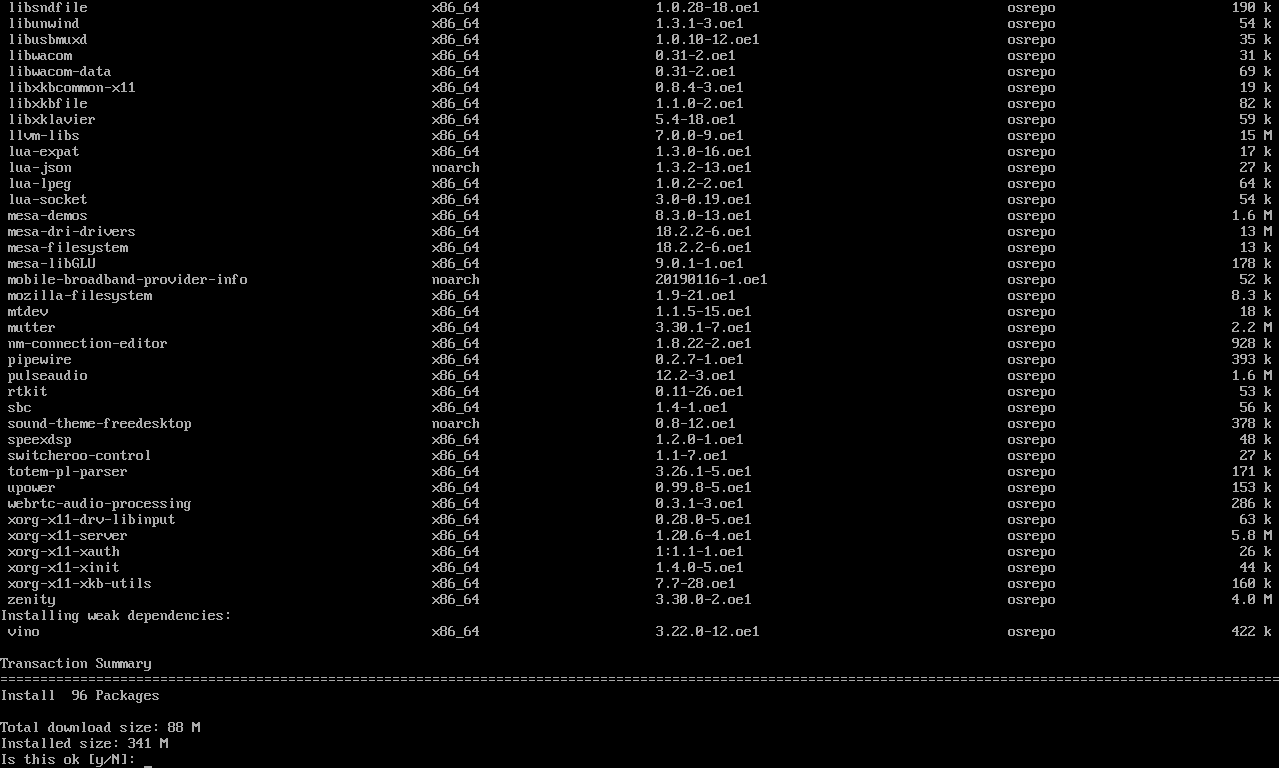
|  |
| --- |
| #添加如下内容  [osrepo]  name=osrepo  baseurl=https://mirrors.tuna.tsinghua.edu.cn/openeuler/openEuler-20.03-LTS/OS/x86\_64/  enabled=1  gpgcheck=1  gpgkey=https://mirrors.tuna.tsinghua.edu.cn/openeuler/openEuler-20.03-LTS/OS/x86\_64/RPM-GPG-KEY-openEuler |



②安装gnome、terminal

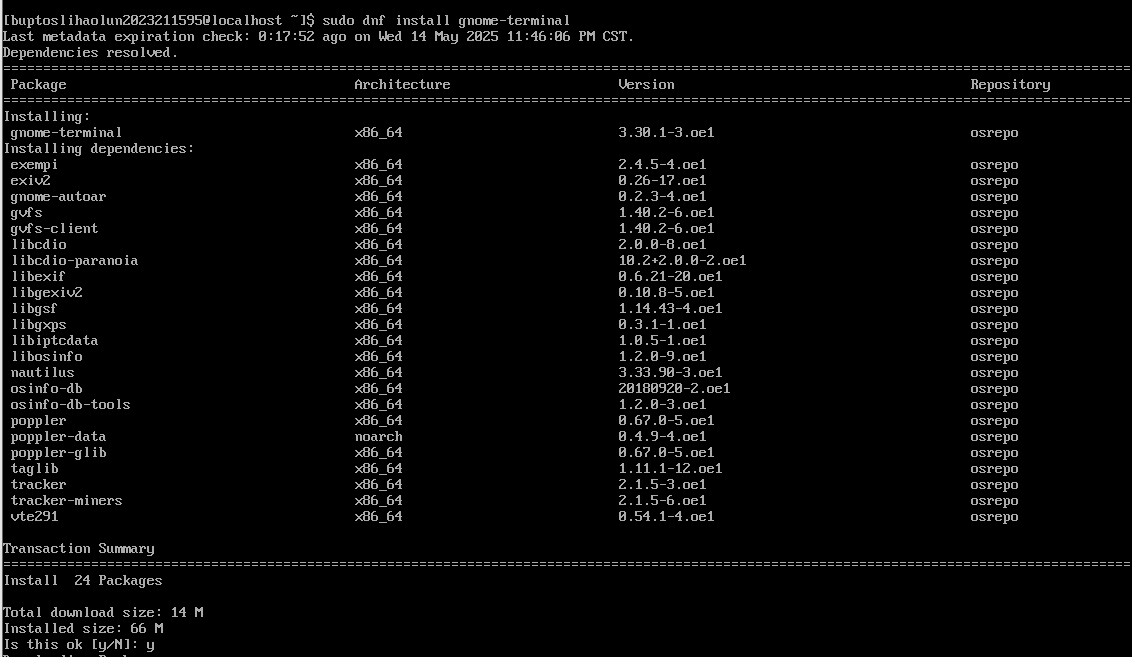
|  |
| --- |
| dnf install gnome-shell gdm gnome-session #安装 gnome 及相关组件 |

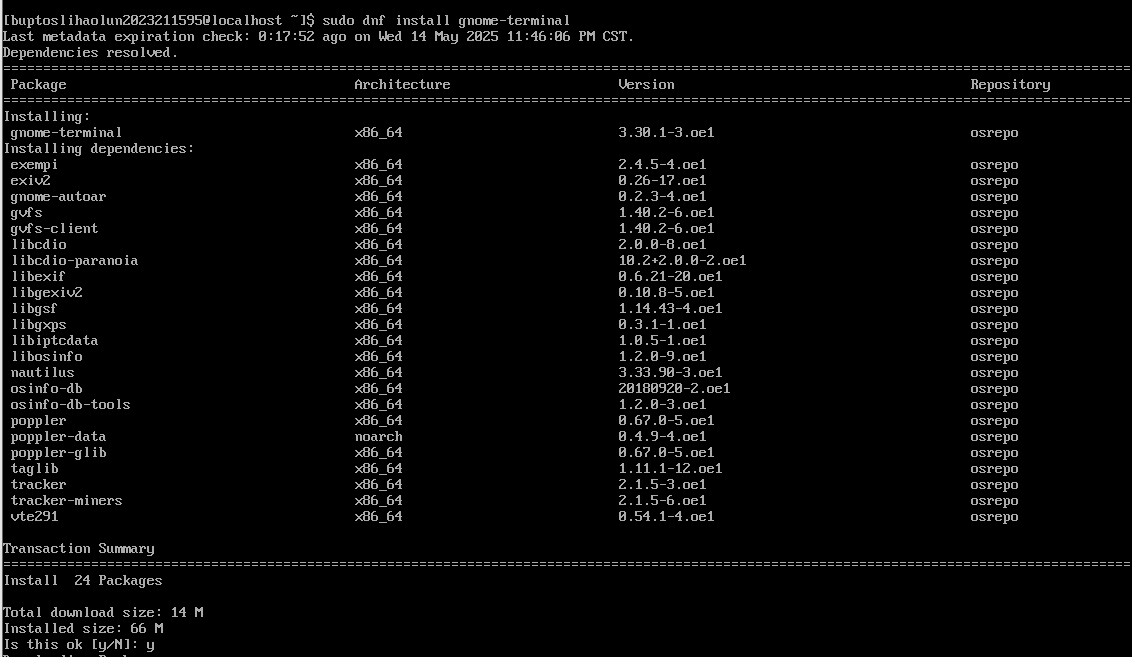


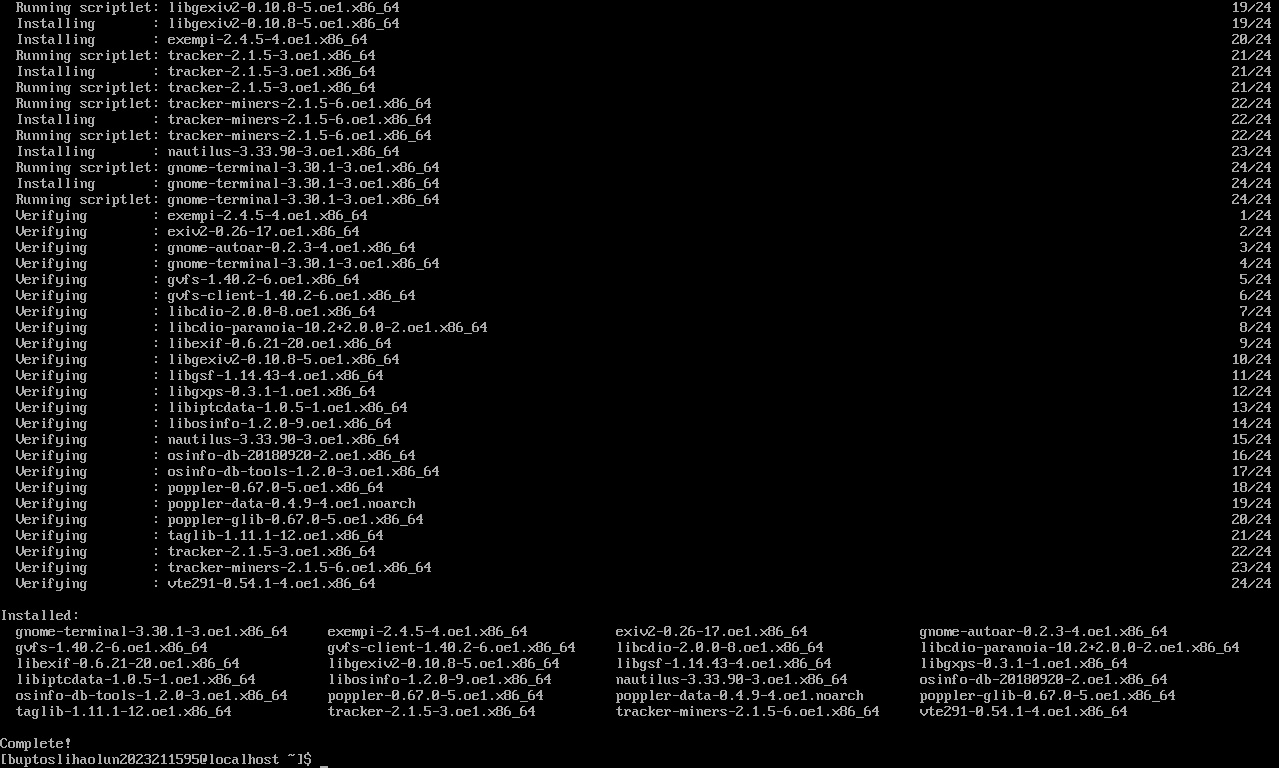




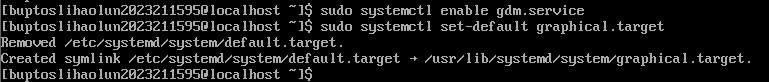
|  |
| --- |
| dnf install gnome-terminal #安装 terminal |



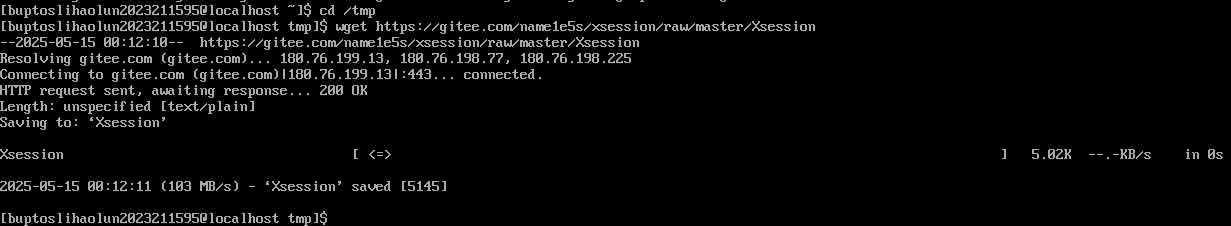




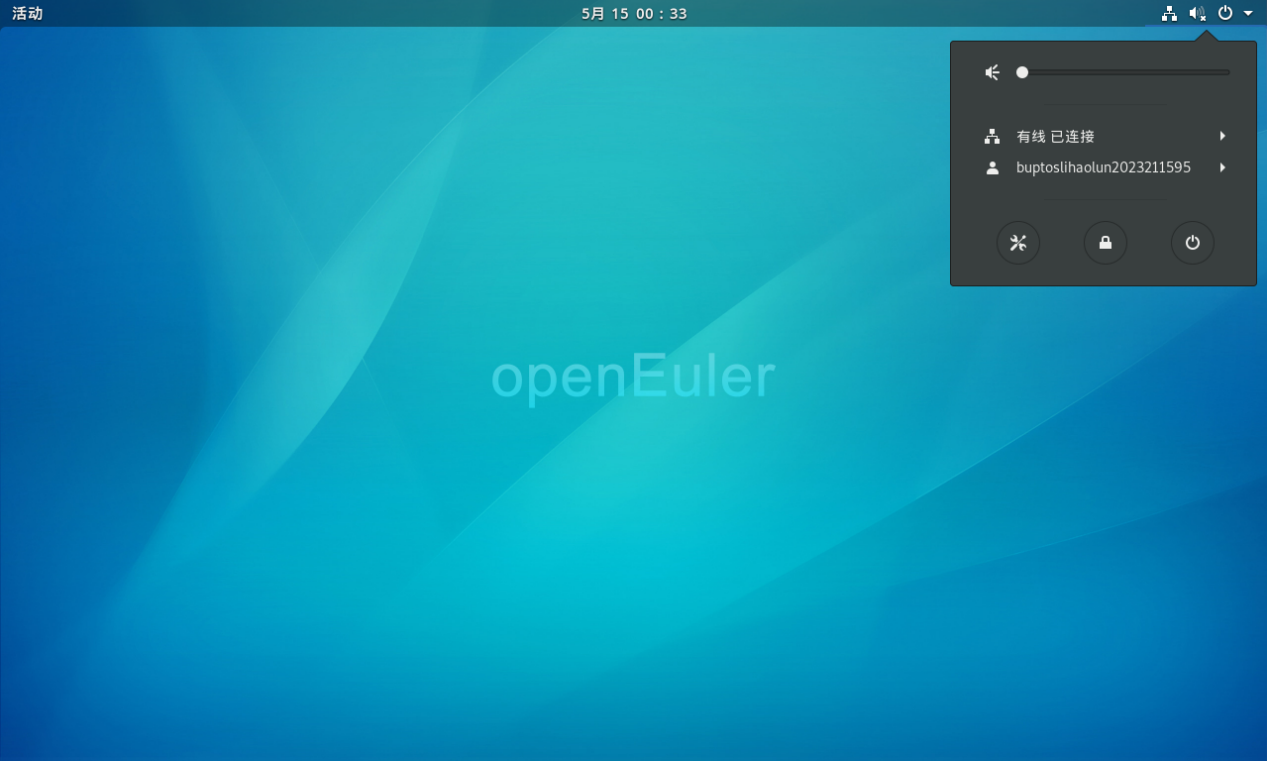
|  |
| --- |
| #设置开机自启动  systemctl enable gdm.service  systemctl set-default graphical.target |



|  |
| --- |
| #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 |

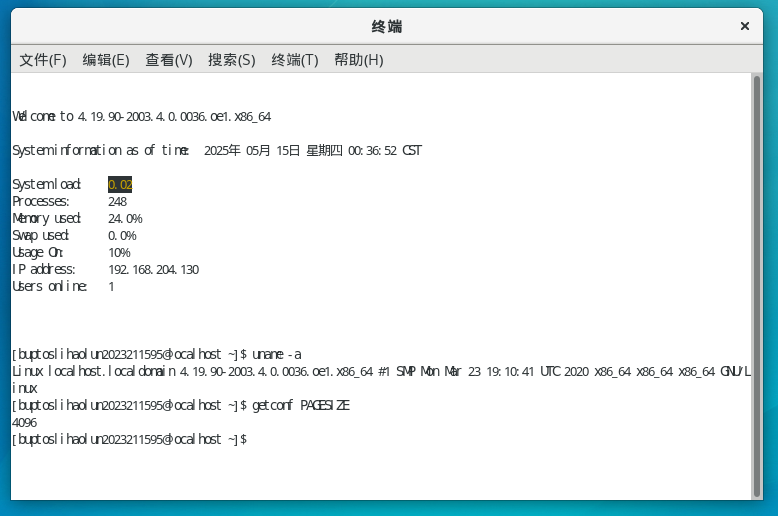




③gnome 桌面安装成功

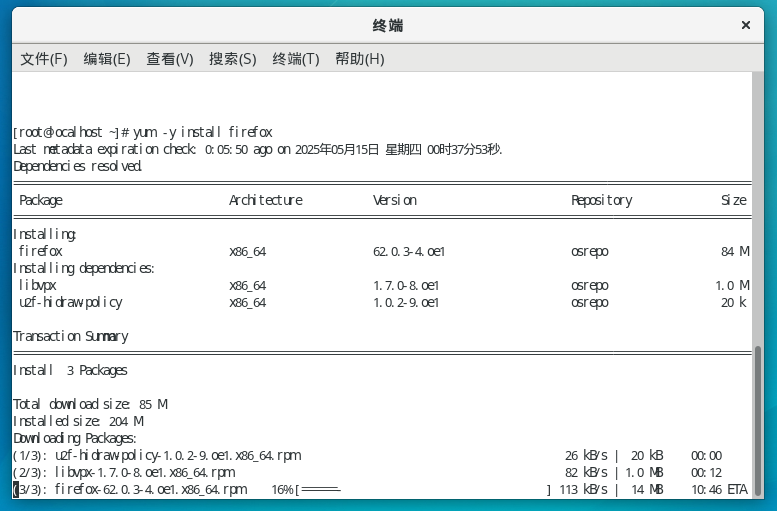
（4）执行uname -a指令、getconf PAGESIZE指令

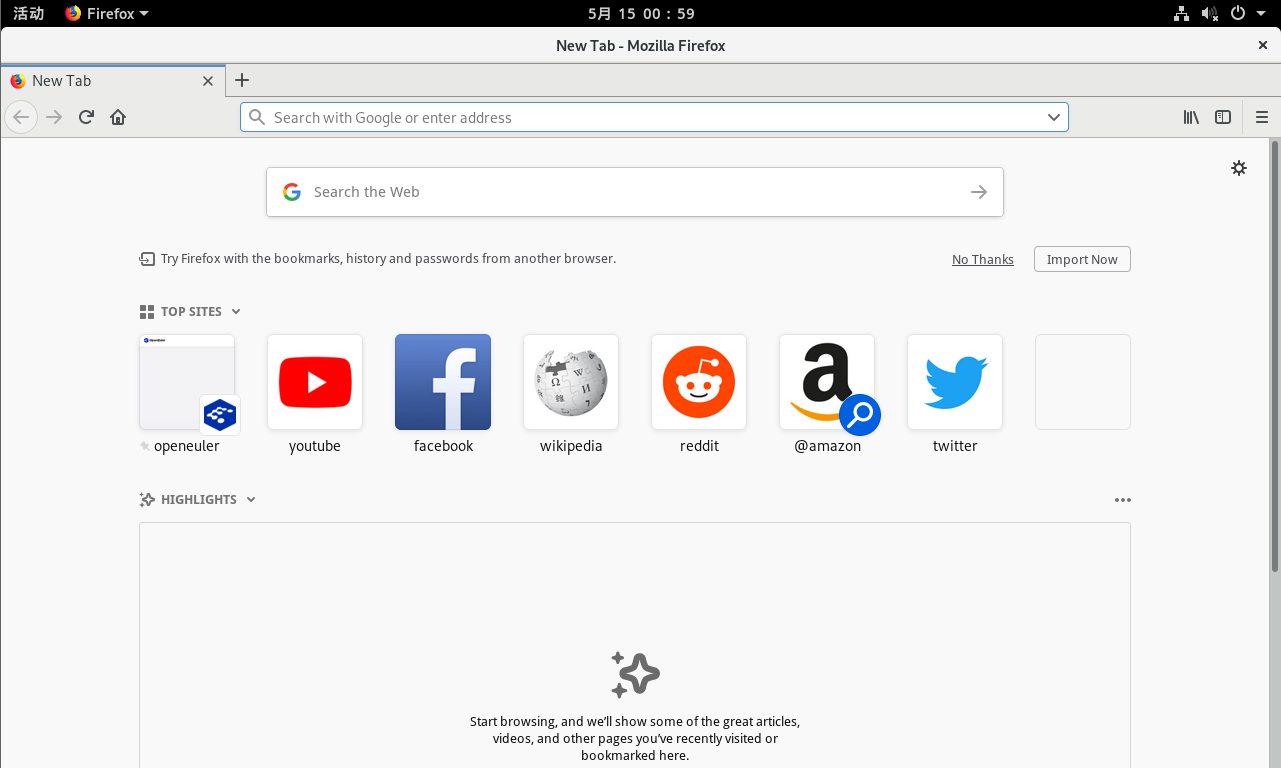
|  |
| --- |
| uname -a  getconf PAGESIZE |



（5）安装Firefox

|  |
| --- |
| yum -y install firefox *#顺便安装 firefox* |

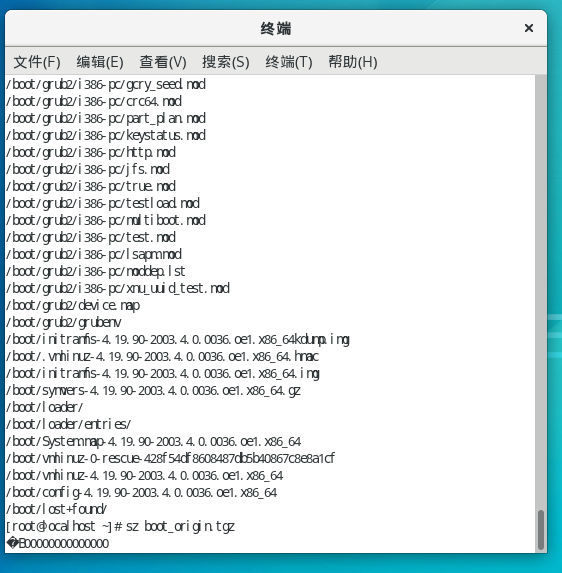
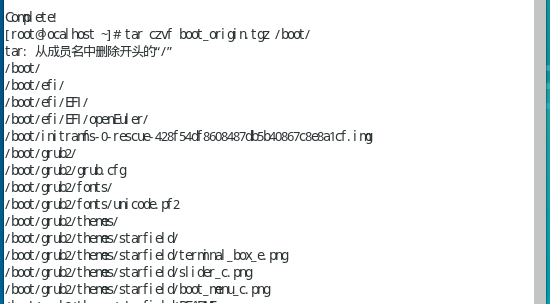
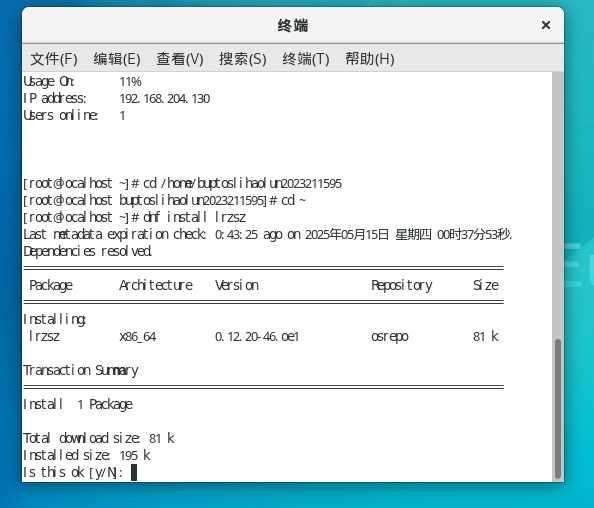




## 2.2 内核更新

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

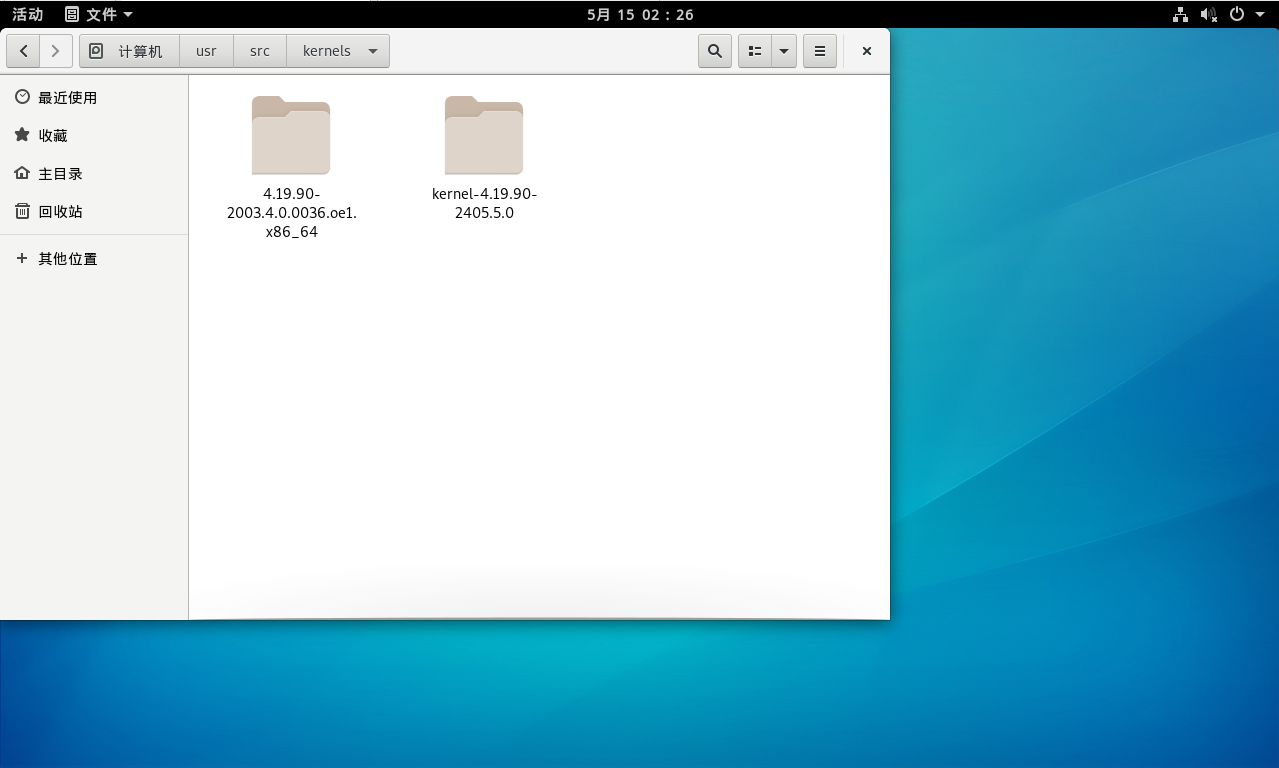
（1）在gitee仓库中下载openEuler内核压缩文件



（2）解压缩至/usr/src/kernels

移动文件指令：

|  |
| --- |
| sudo mv /home/buptoslihaolun2023211595/Downloads/kernel-4.19.90-2405.5.0 /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。

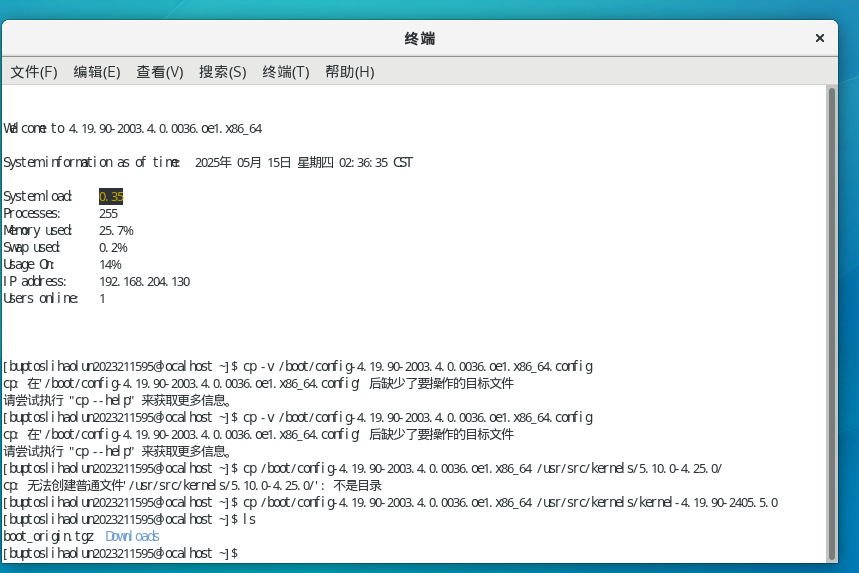
|  |
| --- |
| make mrproper |



4.生成内核配置文件.config

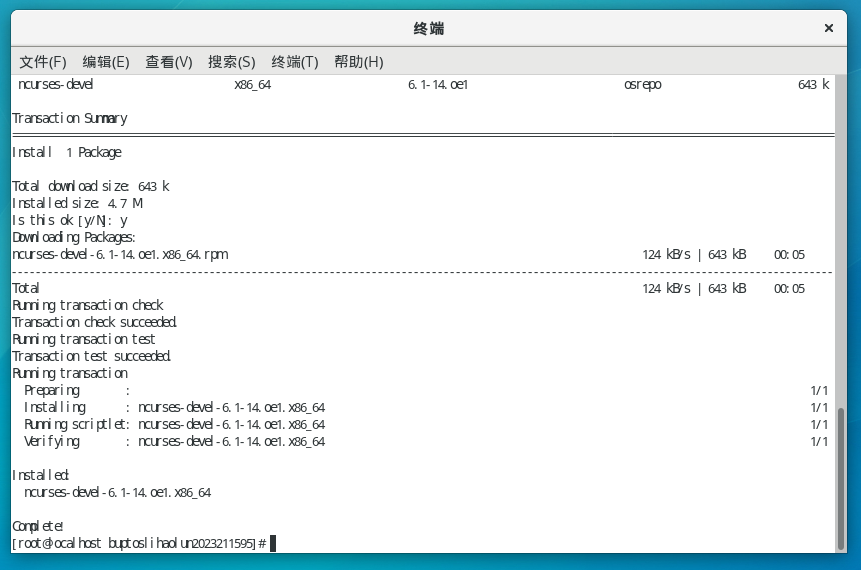
（1）先将将系统原配置文件拷贝过来

|  |
| --- |
| cp /boot/config-4.19.90-2003.4.0.0036.oe1.x86\_64 /usr/src/kernels/kernel-4.19.90-2405.5.0 |



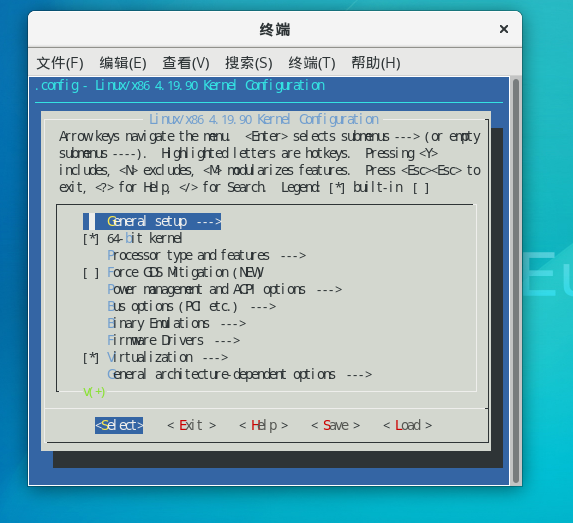
（2）执行依赖安装

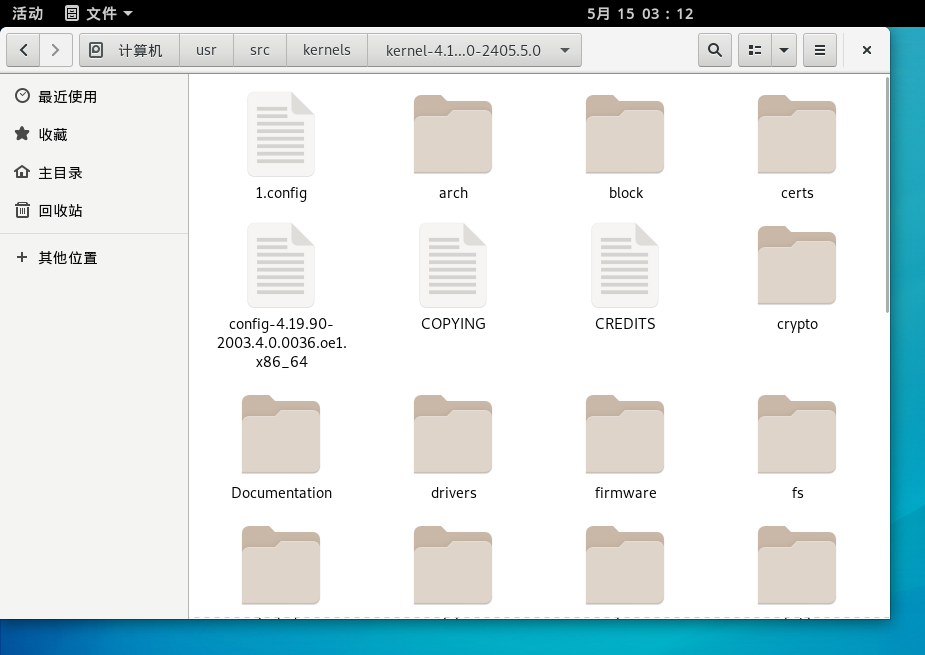
|  |
| --- |
| yum install ncurses-devel |



（3）对配置进行需要的更改：我没有进行改动，直接默认配置，然后选择Save，生成了一个.config文件。

|  |
| --- |
| make menuconfig |

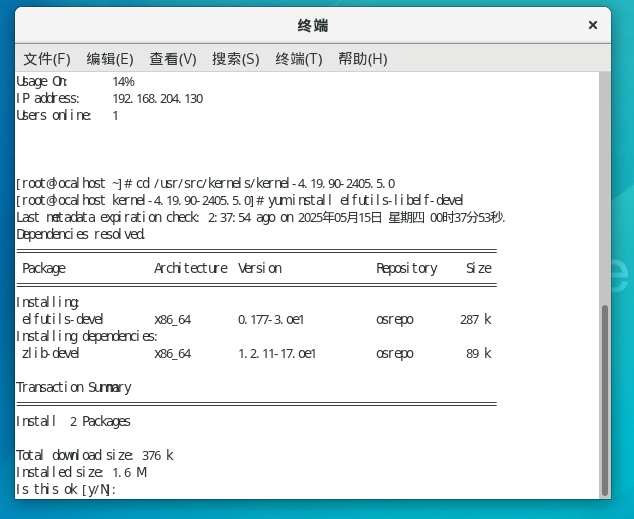


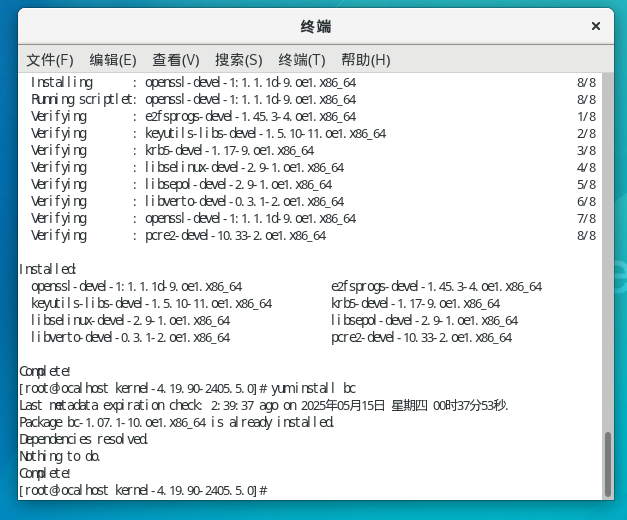
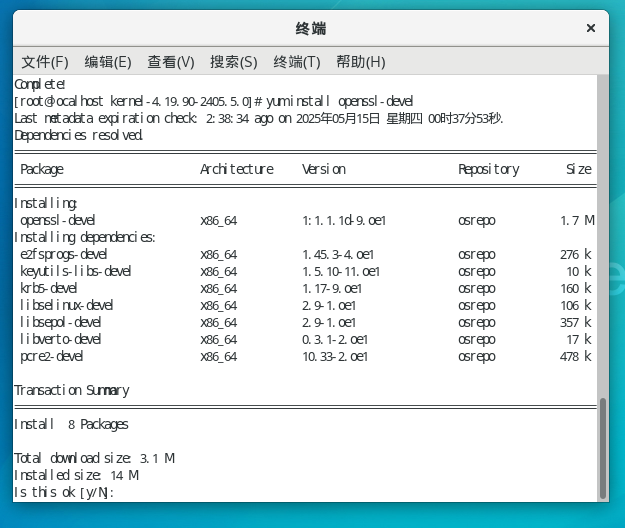


5.内核编译及安装

（1）安装所需组件

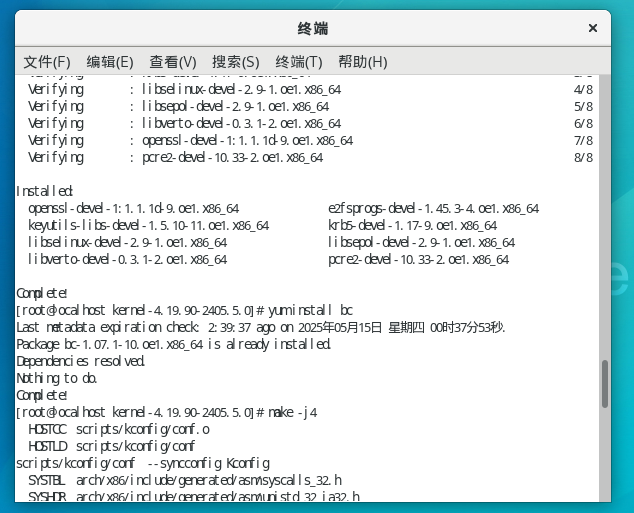
|  |
| --- |
| yum install elfutils-libelf-devel  yum install openssl-devel  yum install bc |



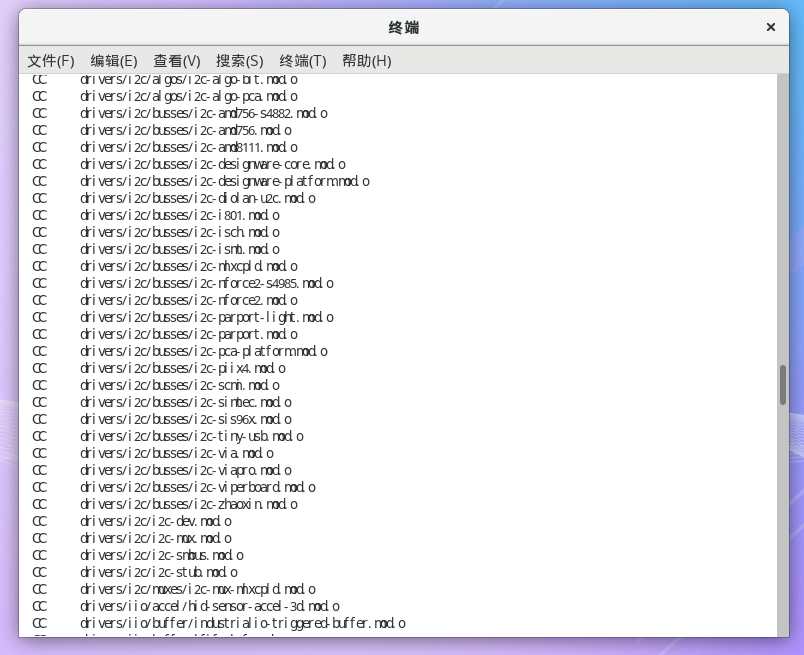


（2）开始编译

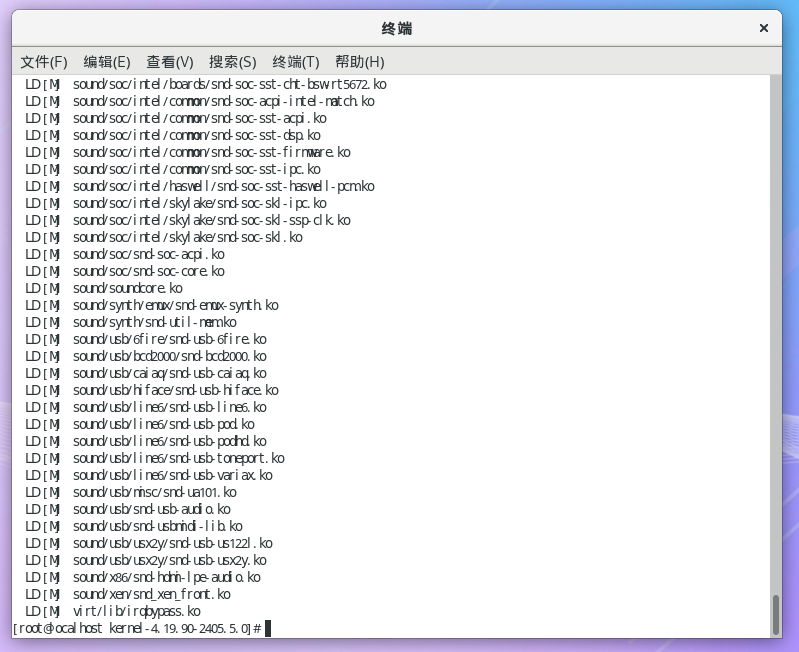
|  |
| --- |
| make -j4 |



①时间很长，大约3~4小时，需要耐心等待。

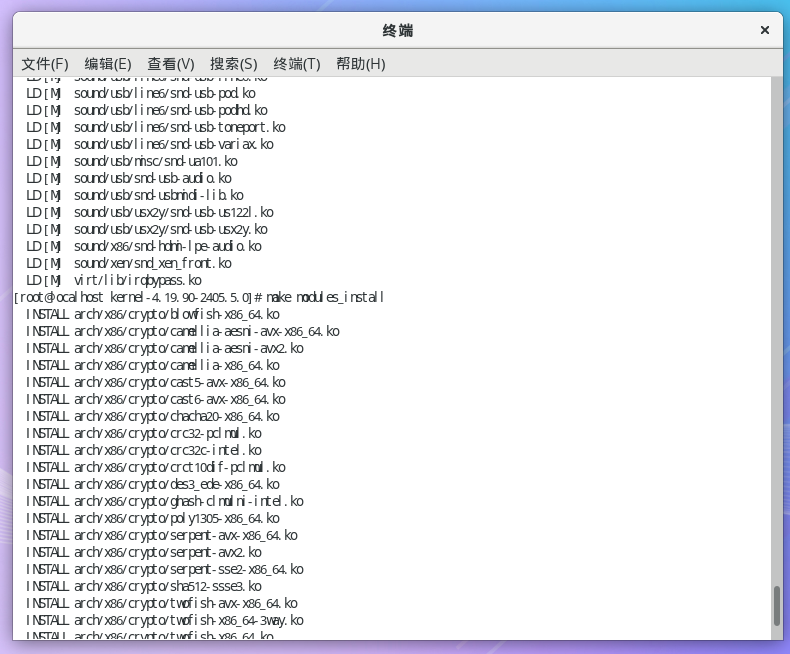


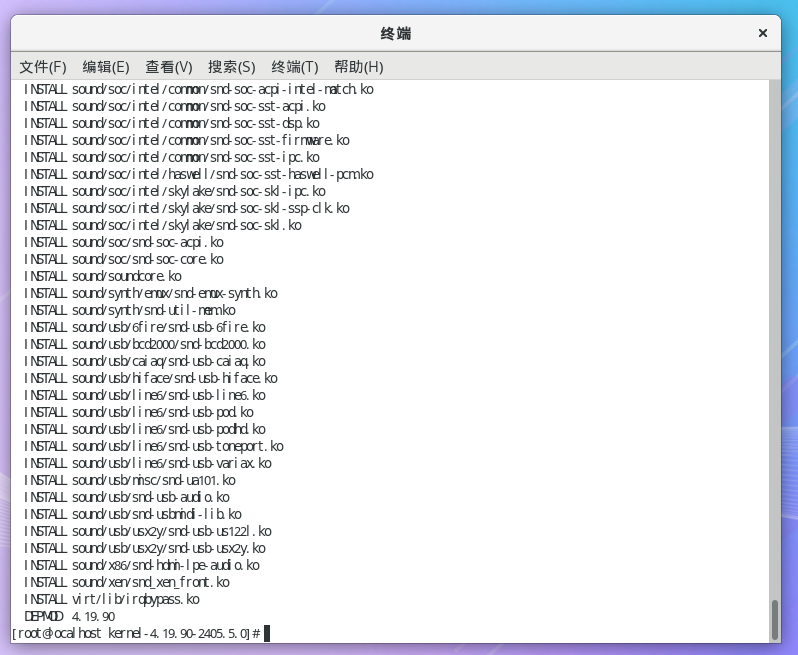
②编译完成



（3）安装模块

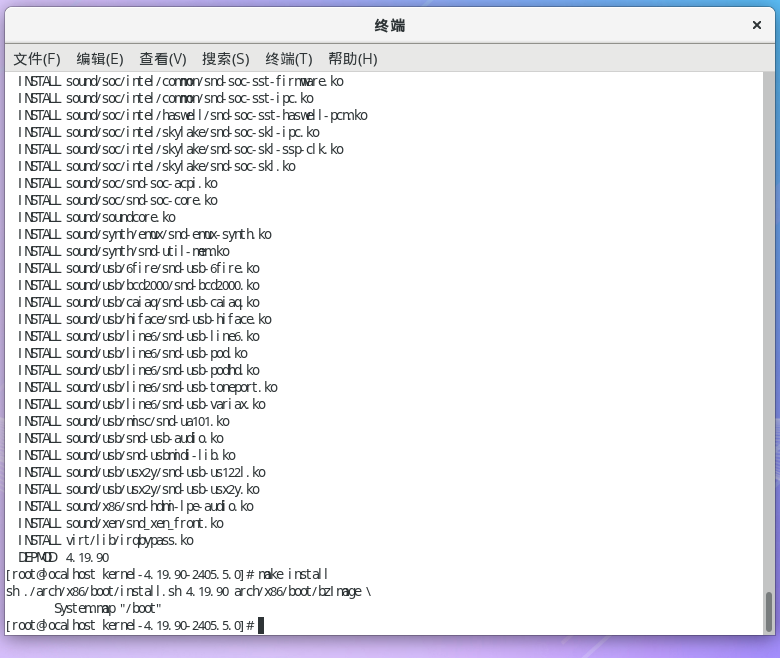
|  |
| --- |
| make modules\_install |



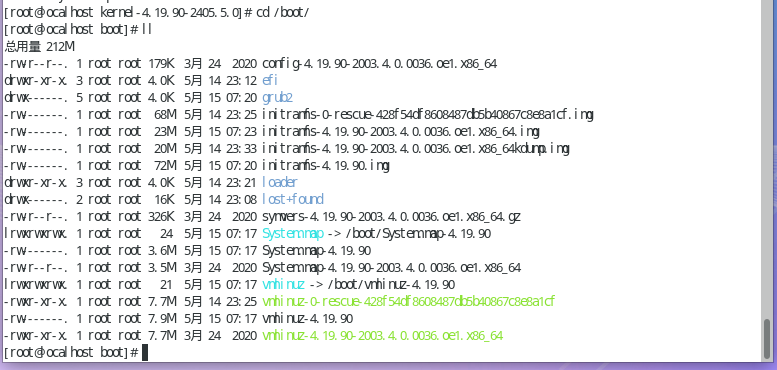


（4）安装内核

|  |
| --- |
| make install |



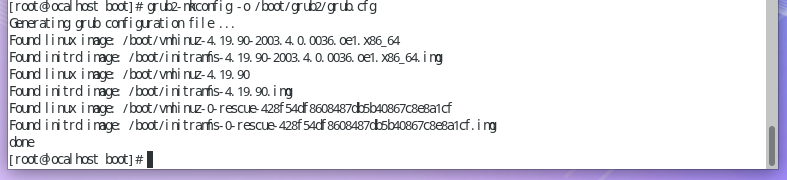
（5）在/boot目录下查看新安装的内核



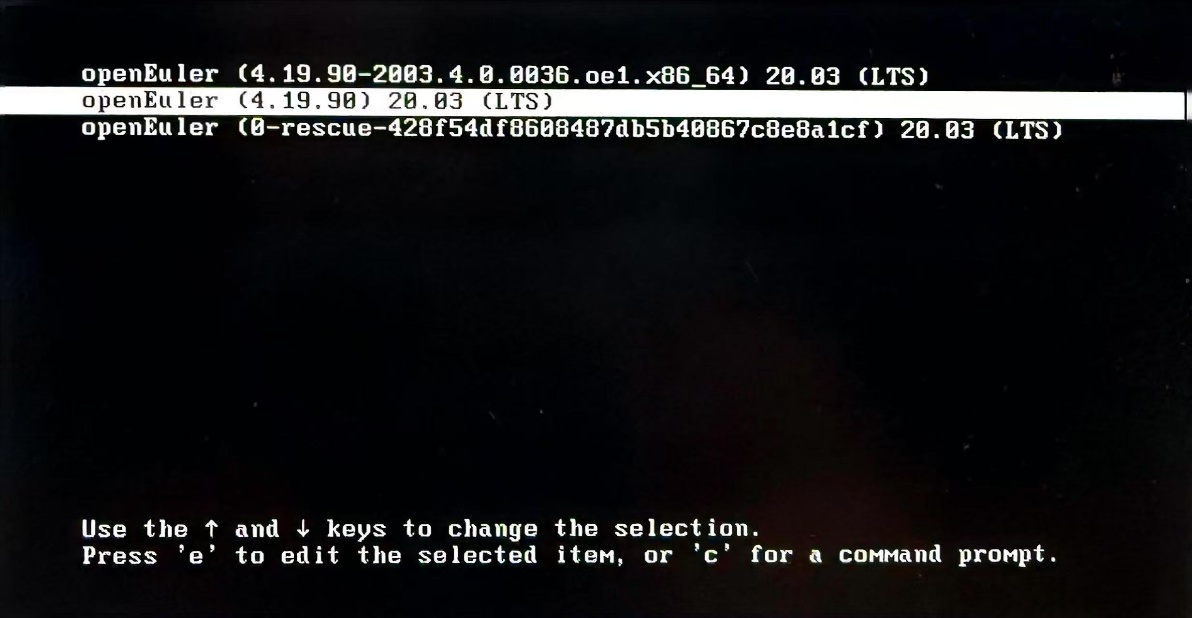
6.更新引导

（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 |



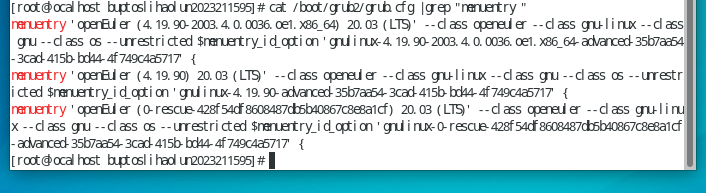
（2）重启，选择第二个，这是我们新安装的内核



7.修改默认启动内核

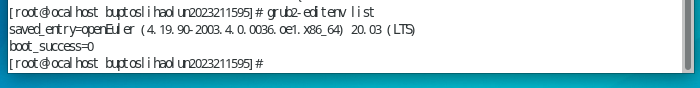
（1）查看当前系统所有可用内核

|  |
| --- |
| 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，输入后再次查看内核版本，发现已经更新为新内核，原内核为4.19.90-2003，新内核为4.19.90

|  |
| --- |
| grub2-set-default 4.19.90 |



（4）执行uname -a指令

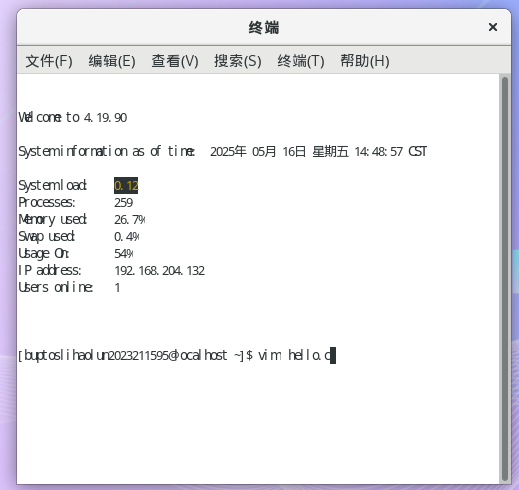
|  |
| --- |
| uname -a |

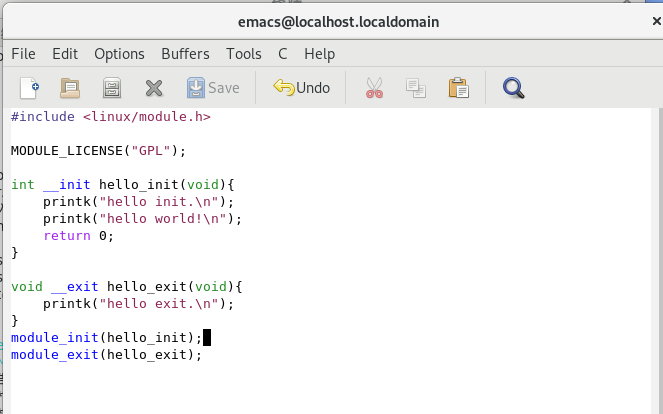


至此，实验步骤完成。

## 2.3 内核模块编程

1.尝试先编写hello world，先编辑c文件



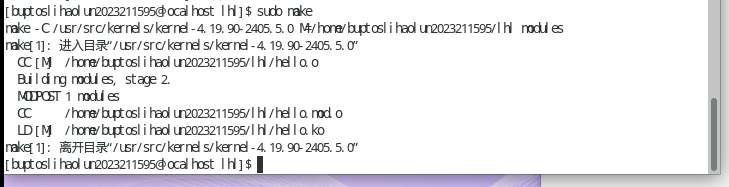


2.此时还需要一个Makefile文件

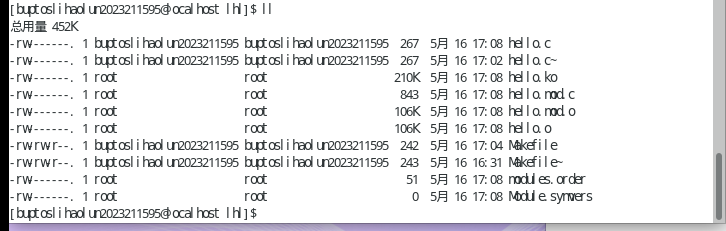




3.准备运行，运行成功



4.查看编译后文件列表



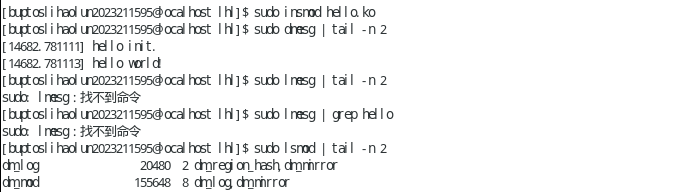
5.执行相关命令。

①加载内核模块：insmod

②查看打印信息：dmesg | tail -n 行数

③查看内核模块：lsmod

④卸载内核模块：rmmod





实验完成。

## 2.4 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

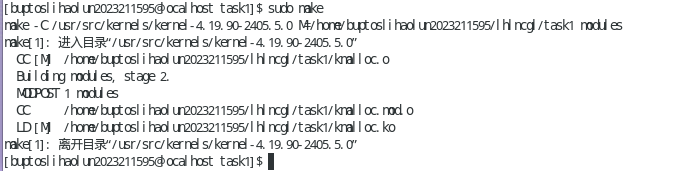
|  |
| --- |
| #include <linux/module.h>  #include <linux/slab.h>  ​  MODULE\_LICENSE("GPL");  ​  unsigned char \*kmallocmem1;  unsigned char \*kmallocmem2;  ​  static int \_\_init mem\_module\_init(void)  {      printk("Start kmalloc!\n");      kmallocmem1 = (unsigned char\*)kmalloc(1024, GFP\_KERNEL);      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。依次输入以下命令进行实验。

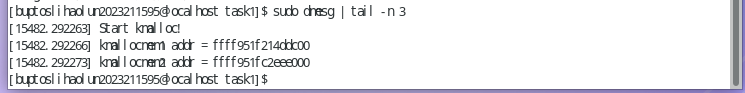
|  |
| --- |
| make |



|  |
| --- |
| insmod kmalloc.ko |



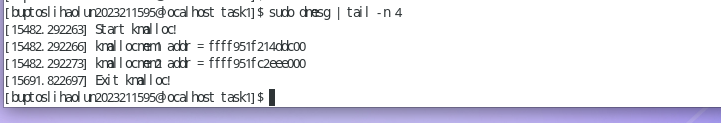
|  |
| --- |
| dmesg | tail -n 3 |



|  |
| --- |
| rmmod kmalloc |

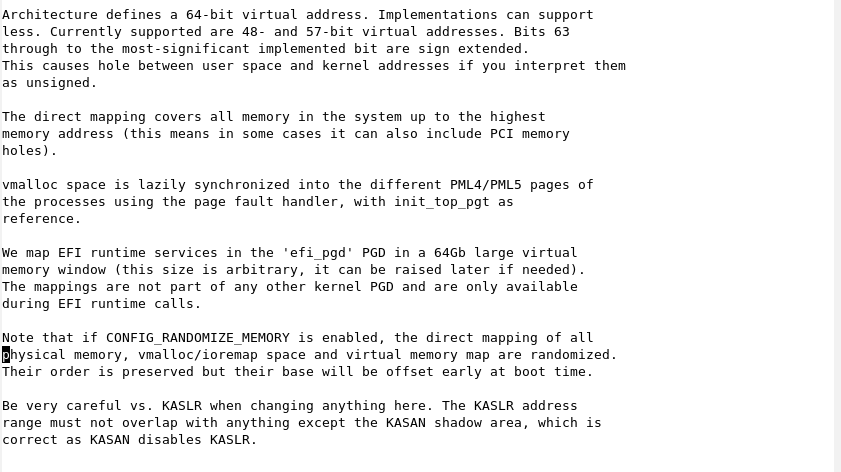
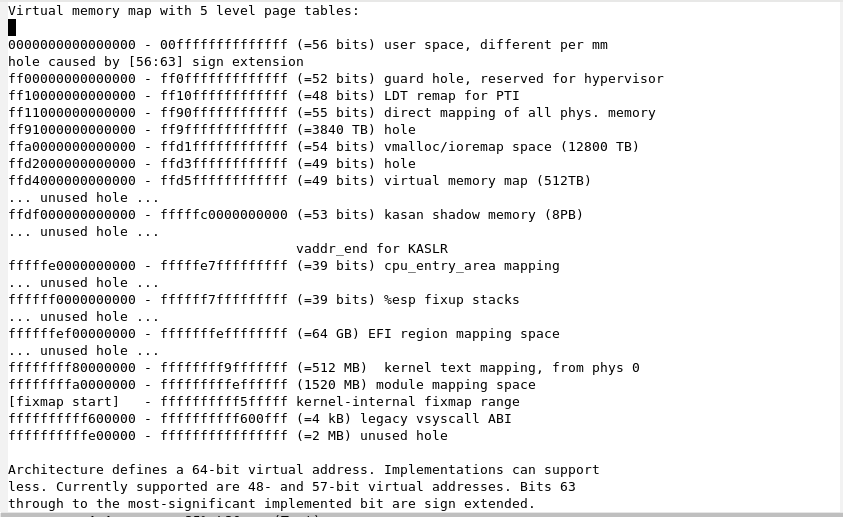
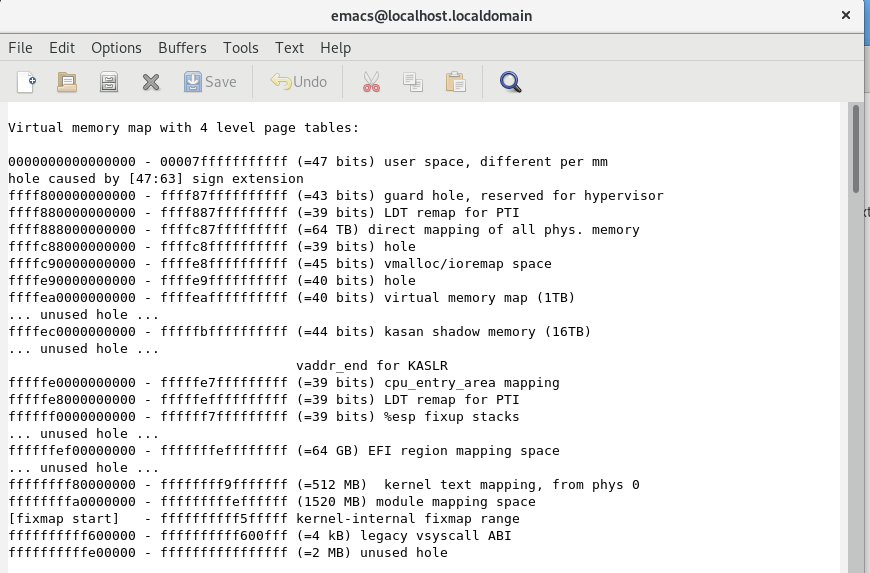


|  |
| --- |
| dmesg | tail -n 4 |



（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



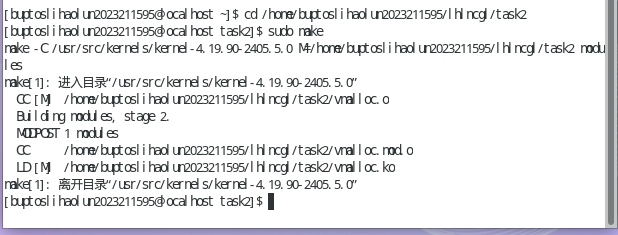
|  |
| --- |
| #include <linux/module.h>  #include <linux/vmalloc.h>  ​  MODULE\_LICENSE("GPL");  ​  unsigned char \*vmallocmem1;  unsigned char \*vmallocmem2;  unsigned char \*vmallocmem3;  ​  static int \_\_init mem\_module\_init(void)  {      printk("Start vmalloc!\n");      vmallocmem1 = (unsigned 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。依次输入以下命令进行实验。

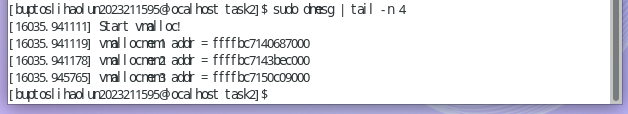
|  |
| --- |
| make |



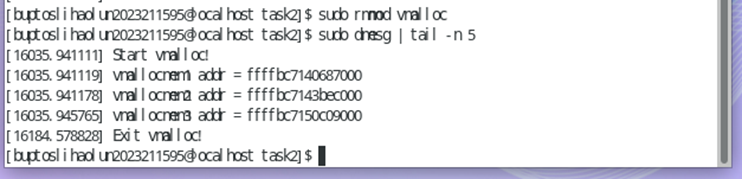
|  |
| --- |
| insmod vmalloc.ko |



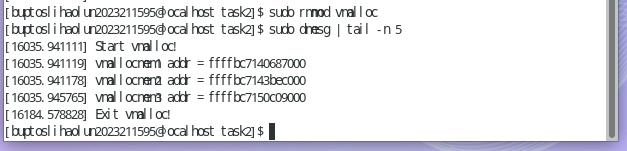
|  |
| --- |
| dmesg | tail -n 4 |



|  |
| --- |
| rmmod vmalloc |

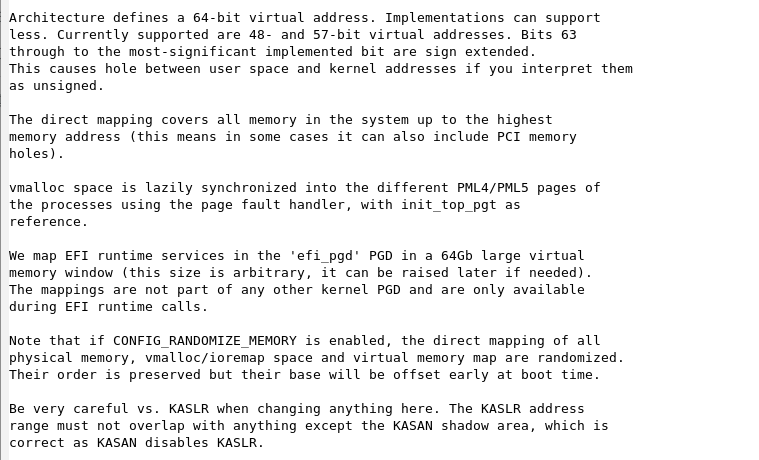
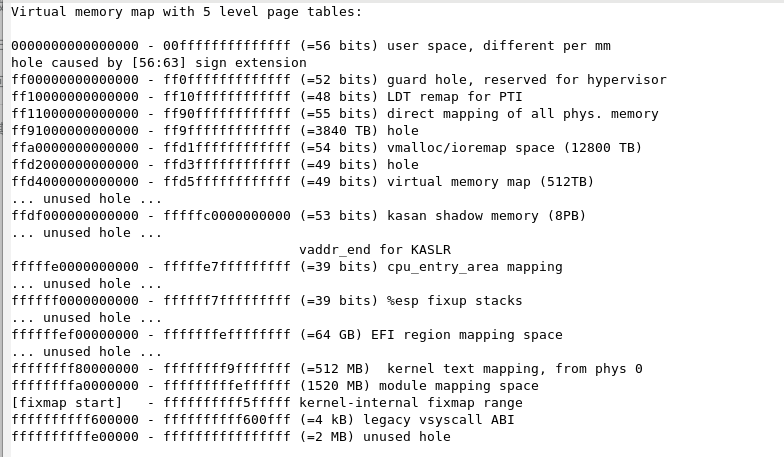
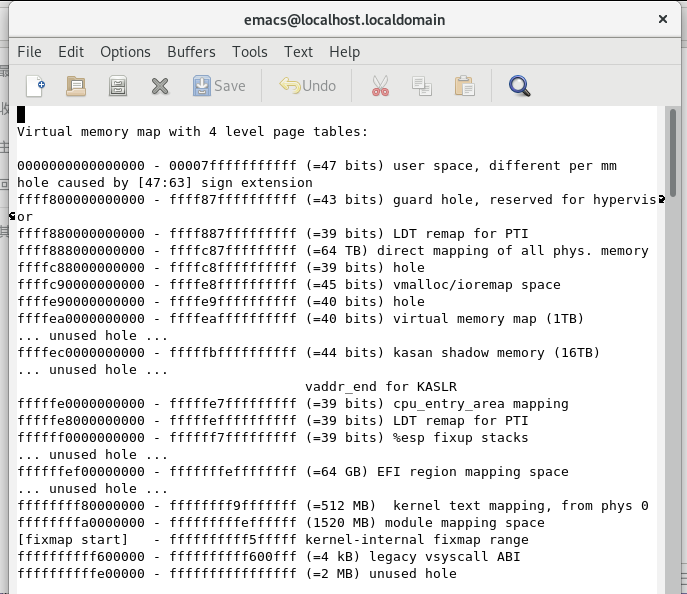


|  |
| --- |
| dmesg | tail -n 5 |



（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。

## 2.5 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

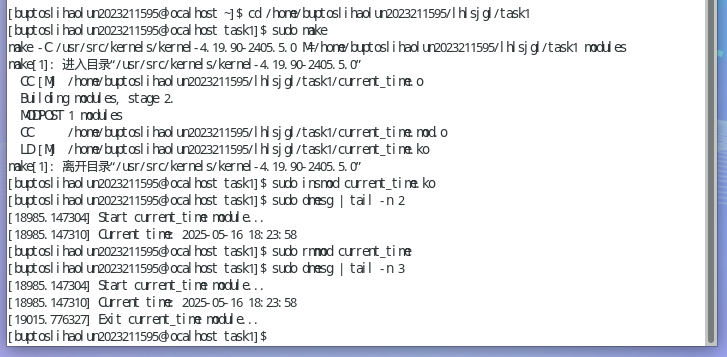
|  |
| --- |
| #include <linux/module.h>  #include <linux/time.h>  #include <linux/rtc.h>  MODULE\_LICENSE("GPL");  struct timeval tv;  struct rtc\_time tm;  static int \_\_init currenttime\_init(void)  {      int year, mon, day, hour, min, sec;      printk("Start current\_time module...\n");      do\_gettimeofday(&tv);      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

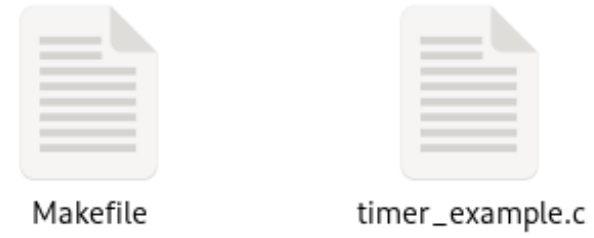
|  |
| --- |
| make  insmod current\_time.ko  dmesg | tail -n 2  rmmod current\_time  dmesg | tail -n 3 |



（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

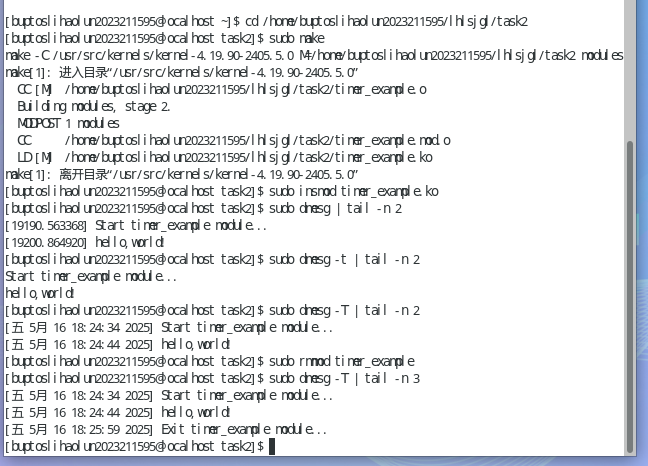
|  |
| --- |
| #include <linux/module.h>  #include <linux/timer.h>  MODULE\_LICENSE("GPL");  struct timer\_list timer;  void print(struct timer\_list \*timer)  {      printk("hello,world!\n");  }  static int \_\_init timer\_init(void)  {      printk("Start timer\_example module...\n");      timer.expires = 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  rmmod 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

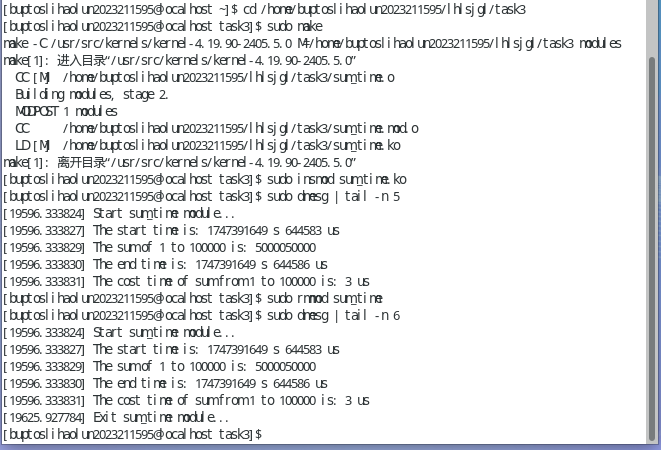
|  |
| --- |
| #include <linux/module.h>  #include <linux/time.h>  MODULE\_LICENSE("GPL");  #define NUM 100000  struct timeval tv;  static long sum(int num)  {      int i;      long total = 0;      for (i = 1; i <= NUM; i++)          total = total + i;      printk("The sum of 1 to %d is: %ld\n", NUM, 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  rmmod 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 3 us。

# 3 问题及解决方案

**问题一：**进行2.1操作系统安装时，进行可视化桌面配置清华源时，没有添加sudo或者切换到root用户，导致一直出现“E212: Can't open file for writing”。

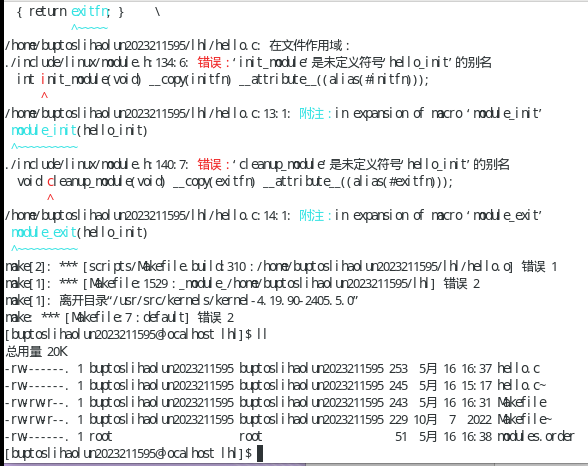
**解决方案：**通过切换root用户或使用sudo命令成功解决，认识到系统文件修改需要管理员权限。

**问题二：**进行2.2内核更新中，无法像主机一样复制剪切粘贴移动内核源码文件。

**解决方案：**安装VMware Tools或使用命令行进行移动

**问题三：**进行2.3内核模块编程部分，总是频繁出现报错。

（1）比如hello.c文件出现编译出错问题



（2）输入make后会出现Mkefile 1:\*\*\* 遗漏分隔符(nu)。停止这种错误。

**解决方案：**最后发现是hello,c代码出现了一些小的错误。Makefile文件中缺少制表符（Tab）​​或使用了错误的缩进方式，因为​Makefile要求命令必须以Tab开头，不能用空格。

**解决问题过程中的参考：**

1.解决桌面可视化问题：https://zhuanlan.zhihu.com/p/229861153

2.[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)

**本实验代码参考：**[LM/OpenEuler\_实验](https://gitee.com/lin-man/open-euler_sh)

# 4 实验总结

通过这次实验，我收获了很多实用的知识和经验。在动手安装openEuler系统的过程中，我学会了如何正确配置和使用这个操作系统。从下载镜像到完成安装，再到创建个性化的用户账号，每一个步骤都让我对系统有了更直观的认识。

在实验中，我不仅成功编译了最新版本的内核，还通过实际操作深入了解了系统是如何进行编程，以及管理内存和时间的。这些经历让我对计算机系统的工作原理有了更清晰的理解，也让我掌握了更多实用的技术。这些知识和技能，不仅对现在的学习很有帮助，也为我以后的工作打下了很好的基础。