OpenEuler 实验

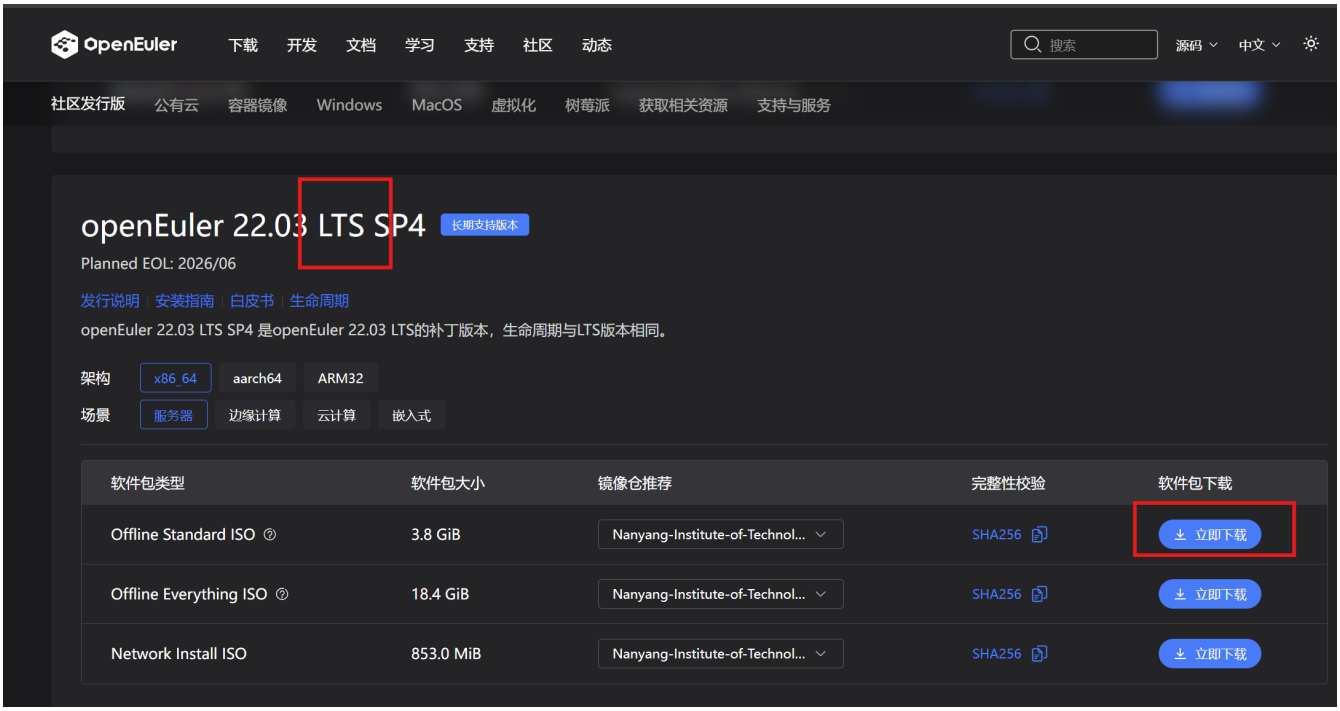
实验内容

1. 在 VMware 中安装 OpenEuler 操作系统；
2. 使用重新编译源代码的方式更新内核；
3. 更新完成后，完成基础操作系统实验，如内核模块编程、内存管理、进程管理、设备管理、中断实验等。

实验过程

第一部分 安装 OpenEuler

1. 安装 VMware，直接根据自己要求选择即可，都默认也可以，不细说。官网：https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion 。
2. 获取 OpenEuler 的镜像，官网：https://www.openeuler.org/zh/download/ ,进入后选择 LTS 即长期维护版本，此处选择22.03版标准版。



3. 进入 VMware 创建虚拟机（自定义），这一部分就相当于准备一台裸机，然后将操作系统这个最基础的软件放进去。具体操作如下：

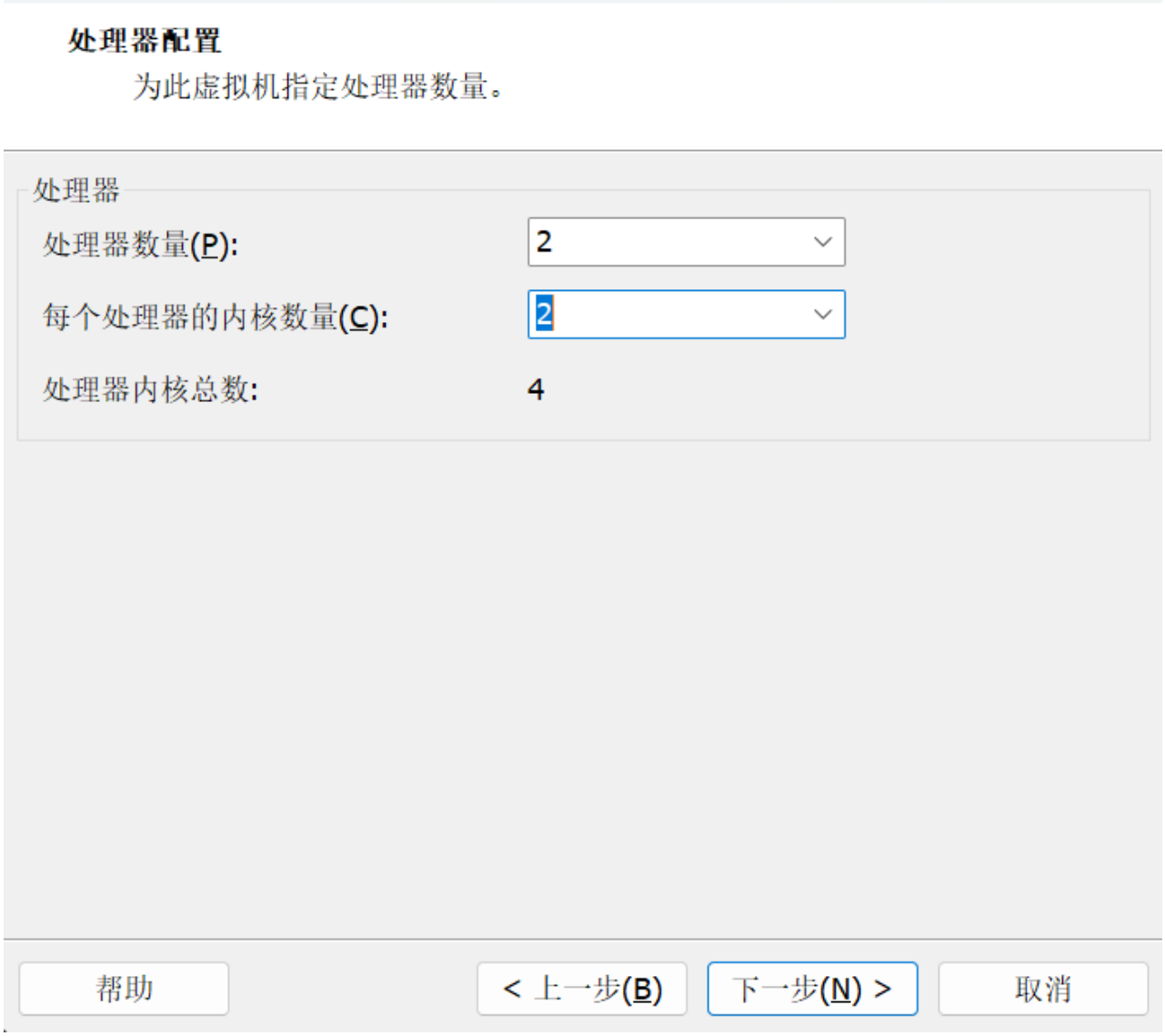
1. 选择硬件兼容，直接下一步；

2. 选择“稍后安装操作系统”，下一步；

3. 选择Linux，使用其他Linux 64位版本，下一步；

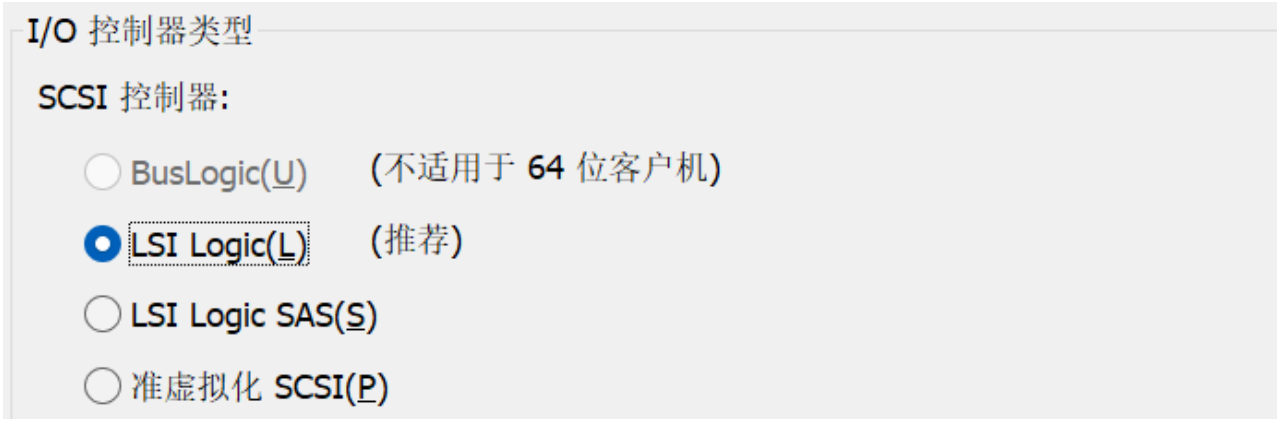
4. 给虚拟机起个名字并指定路径，也可以不改，下一步；

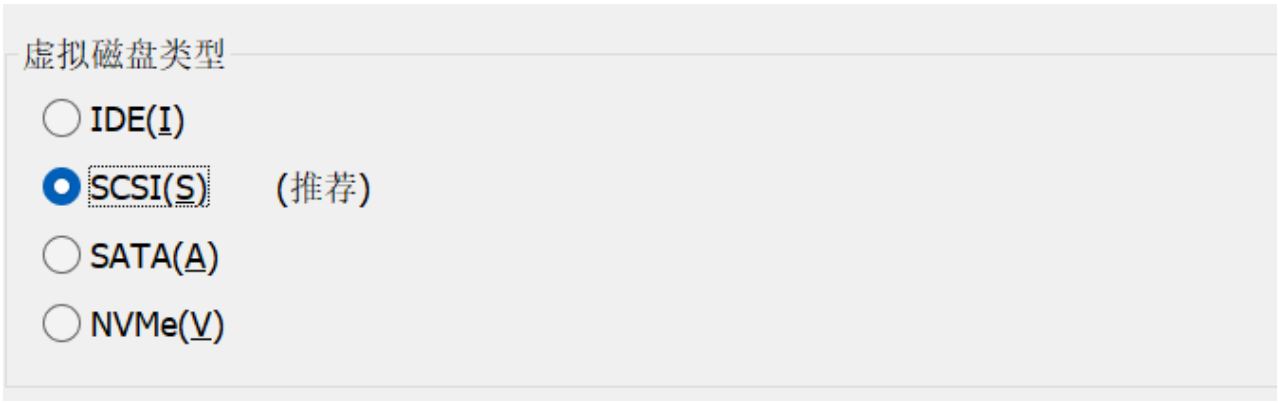
5. 选择CPU配置，改不改都行；



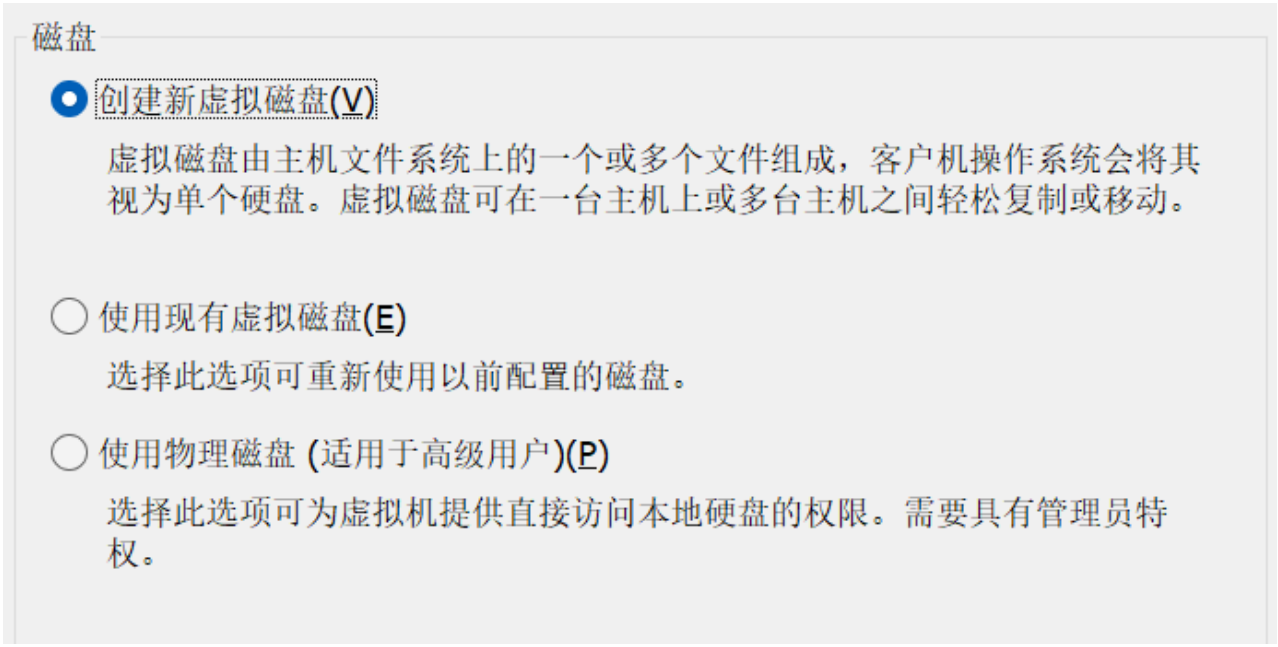
6. 内存给了4GB，应该够用；

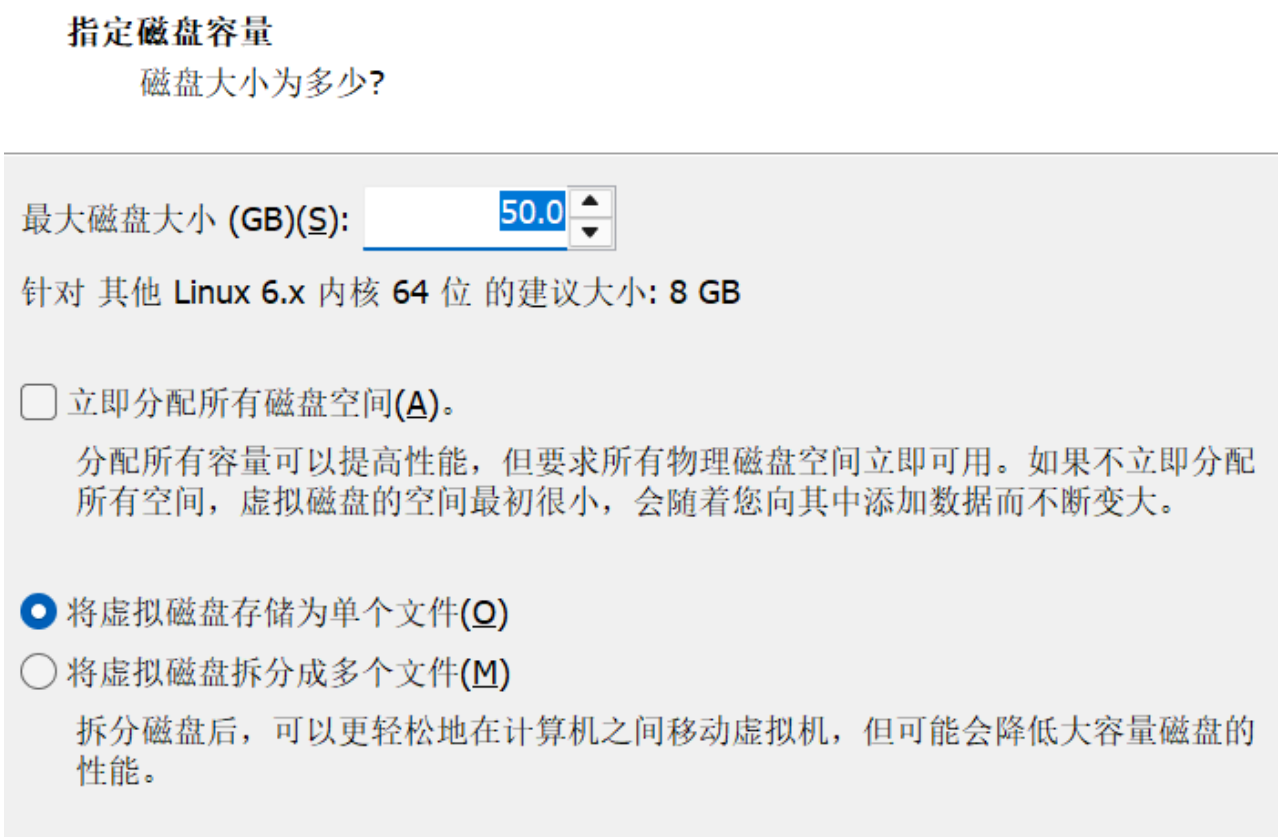
7. 后面三步都默认即可（ NAT、LSI Logic、SCSI） ；



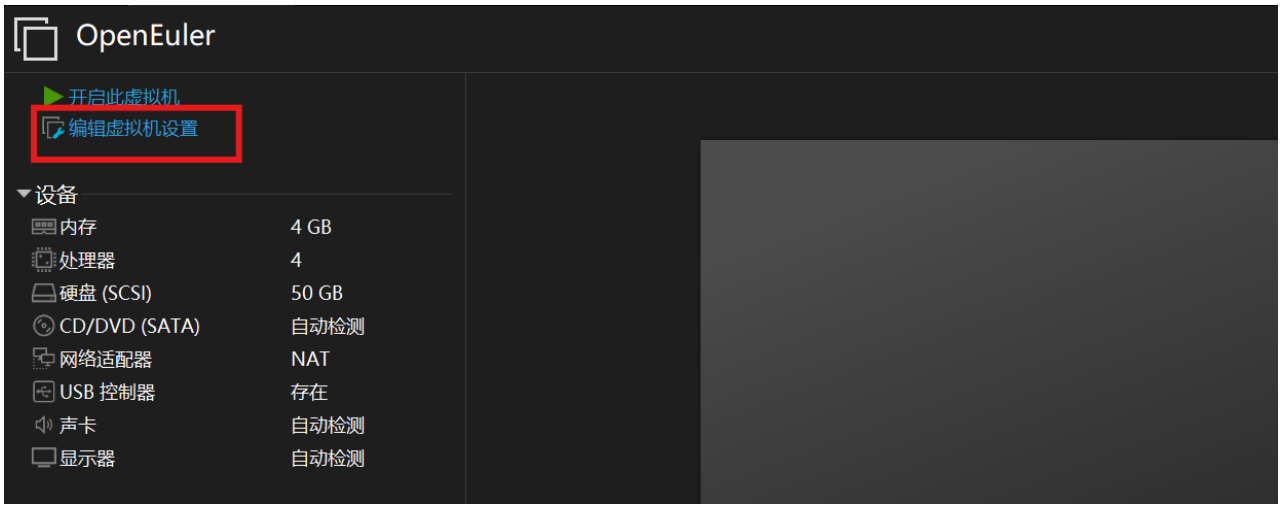


8. 创建新的磁盘，给定磁盘大小（我给了50GB），为了性能把它放在单个文件里；

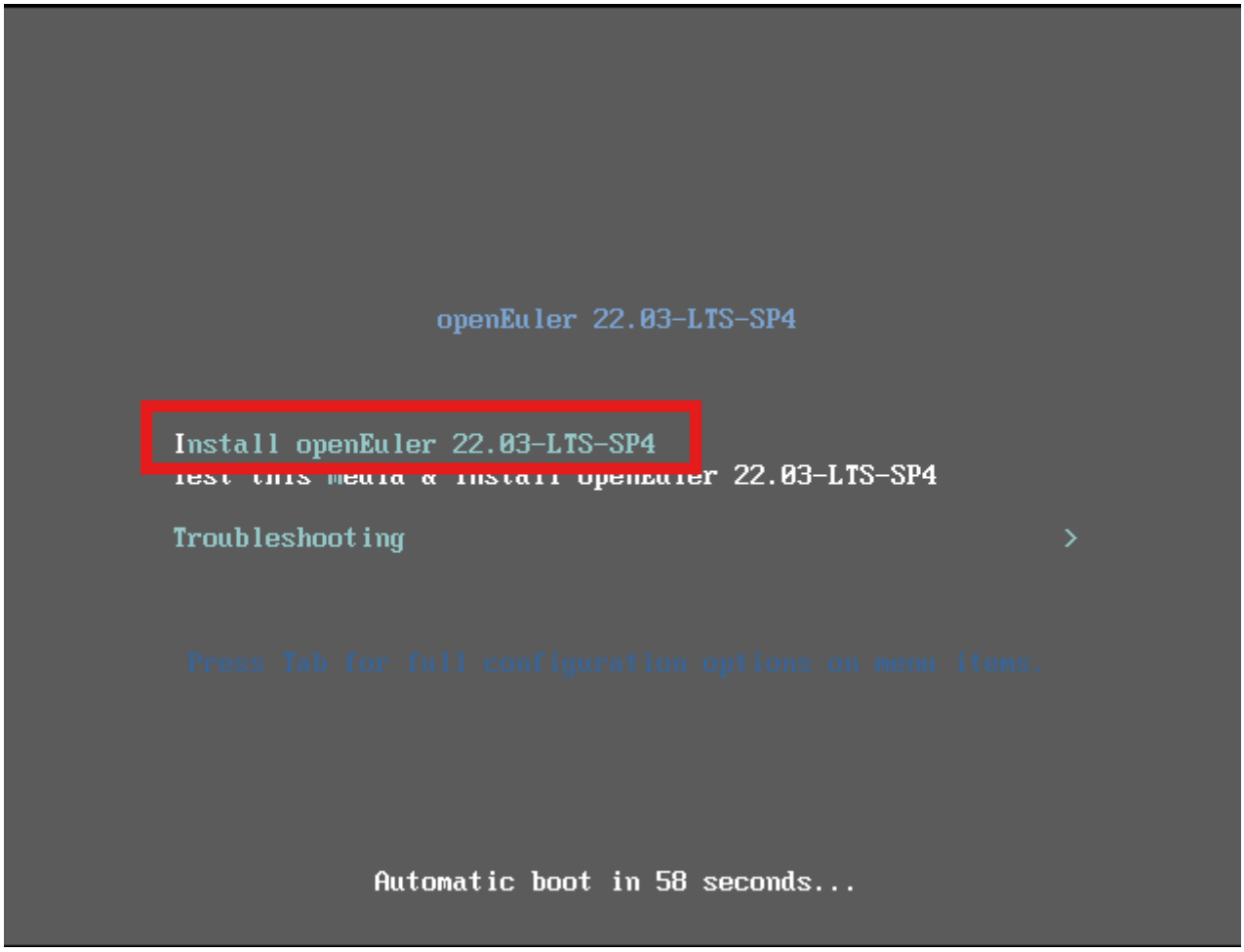




9. 然后就下一步、完成即可；
10. 编辑虚拟机设置，指定操作系统镜像（刚才下载的ISO文件），最后点击确定，成功创建虚拟机。



4. 开启虚拟机进行操作系统的安装：
    1. 选择直接安装，回车选择后等待自动加载；



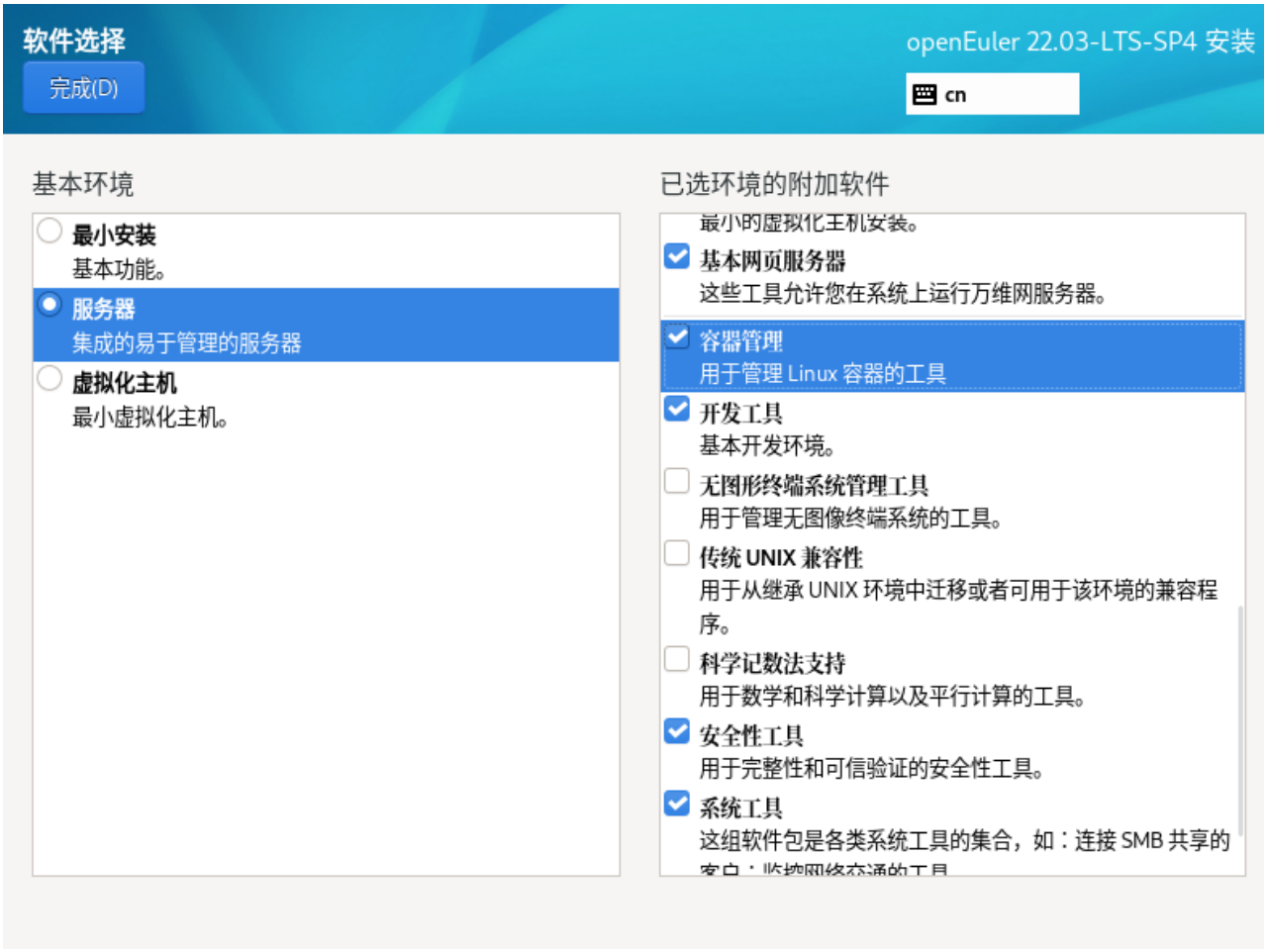
2. 完成后选择语言，为了方便选择中文（不挂图了），开始安装前需要配置图中几个部分：

3. 安装目的地：选择自定义，选定本地磁盘然后点击“完成”，在手动分区里，直接自动创建，然后再次点击“完成”，最后在弹窗中选择接受更改；


完成(D)
openEuler 22.03-LTS-SP4 安装
cn
本地标准磁盘
50 GiB
VMware, VMware Virtual S
sda / 50 GiB 空闲
此处未选择的磁盘将不会被操作。
专用磁盘 & 网络磁盘
添加磁盘(A)...
此处未选择的磁盘将不会被操作。
存储配置
自动(U)
自定义(C)
完整磁盘摘要以及引导程序(F)...
已选择 1 个磁盘；容量 50 GiB；50 GiB 空闲
刷新(R)...


手动分区
完成(D)
openEuler 22.03-LTS-SP4 安装
cn
新 openEuler 22.03-LTS-SP4 安装
您还没有为 openEuler 22.03-LTS-SP4 的安装创建任何挂载点。您可以：
点击这里自动创建它们(C)。
通过点击"+"按钮创建新挂载点。
新挂载点将使用以下分区方案(N)：
LVM
自动地加密默认创建的挂载点(E)：
加密我的数据(_E)。
在您为 openEuler 22.03-LTS-SP4 安装创建挂载点后，您可在这里浏览它们的详细信息。
可用空间
50 GiB
总空间
50 GiB
已选择 1 个存储设备(S)
放弃所有更改(D)

4. 软件选择：这里选择服务器和一些基本功能；



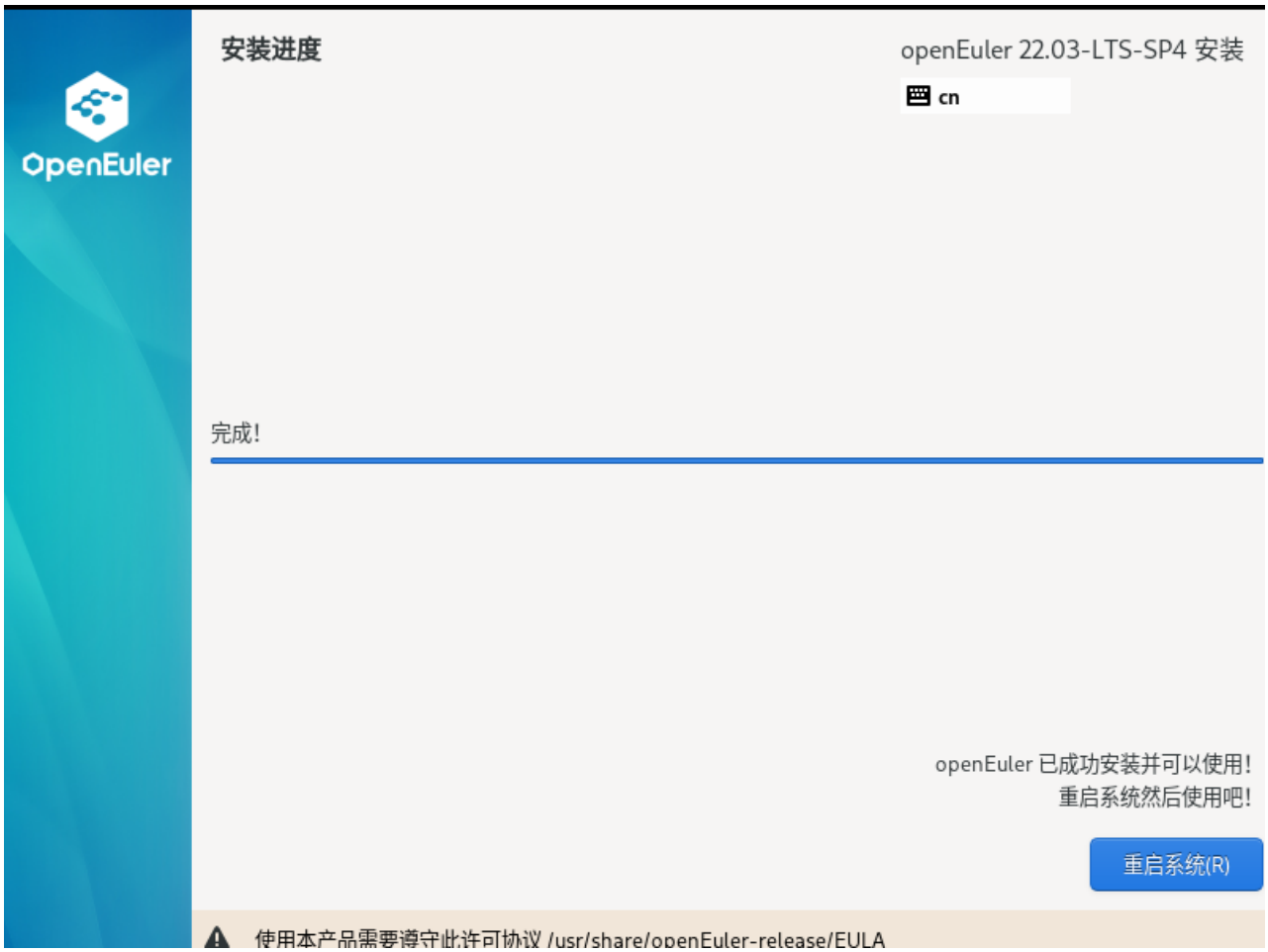
5. 网络和主机名；

6. root账户：设置密码，要求强密码（至少三种字符）；

7. 创建用户：类似注册过程，此处创建的是普通用户；

8. 点击“开始安装”，然后等待即可；

9. 安装完成后重启系统，成功安装。



5. 创建用户（安装OS后）：

- 使用命令 useradd 创建用户，passwd 修改密码（OpenEuler对密码复杂度有要求）；

```
[root@localhost /]# useradd dzh
[root@localhost /]# id dzh
uid=1001(dzh) gid=1001(dzh) groups=1001(dzh)
[root@localhost /]# passwd dzh
Changing password for user dzh.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@localhost /]# id dzh
uid=1001(dzh) gid=1001(dzh) groups=1001(dzh)
[root@localhost /]# _
```

- 验证：

```
[root@localhost /]# su dzh


Welcome to 5.10.0-216.0.0.115.oe2203sp4.x86_64

System information as of time:  Tue May 13 05:34:32 PM CST 2025

System load:    0.00
Memory used:    7.7%
Swap used:      0%
Usage On:       8%
IP address:     192.168.31.133
Users online:   1
To run a command as administrator(user "root"),use "sudo <command>".
[dzh@localhost /]$ whoami
dzh
[dzh@localhost /]$ _
```

6. 执行 uname -a：

- 结果：

```
[root@localhost /]# uname -a
Linux localhost.localdomain 5.10.0-216.0.0.115.oe2203sp4.x86_64 #1 SMP Thu Jun 27 15:13:44 CST 2024 x86_64 x86_64 x86_64 GNU/Linux
[root@localhost /]# _
```

- 解释：分别是内核名、主机名、内核发行号、内核版本、硬件架构、处理器类型、硬件平台、操作系统名。

```
[root@localhost /]# uname --help
Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
  -v, --kernel-version     print the kernel version
  -m, --machine            print the machine hardware name
  -p, --processor          print the processor type (non-portable)
  -i, --hardware-platform  print the hardware platform (non-portable)
  -o, --operating-system   print the operating system
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/uname>
or available locally via: info '(coreutils) uname invocation'
[root@localhost /]# _
```

7. 查看分页大小：直接输入 `getconf PAGESIZE` 即可，查出来是4096（单位是字节）。

```
[root@localhost /]# getconf PAGESIZE
4096
[root@localhost /]# _
```

8. 后续操作单用命令行不是很容易，所以这里使用vscode进行远程连接（SSH）：
   1. 安装OpenSSH-Server服务：`yum install -y openssl openssh-server`；

```
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                                                                    1/1
  Running scriptlet: openssl-libs-1:1.1.1wa-10.oe2203sp4.x86_64                                                         1/1
  Upgrading        : openssl-libs-1:1.1.1wa-10.oe2203sp4.x86_64                                                         1/12
  Running scriptlet: openssl-libs-1:1.1.1wa-10.oe2203sp4.x86_64                                                         1/12
  Running scriptlet: openssh-server-8.8p1-34.oe2203sp4.x86_64                                                           2/12
  Upgrading        : openssh-server-8.8p1-34.oe2203sp4.x86_64                                                           2/12
  Running scriptlet: openssh-server-8.8p1-34.oe2203sp4.x86_64                                                           2/12
  Running scriptlet: openssh-8.8p1-34.oe2203sp4.x86_64                                                                  3/12
  Upgrading        : openssh-8.8p1-34.oe2203sp4.x86_64                                                                  3/12
  Upgrading        : openssh-clients-8.8p1-34.oe2203sp4.x86_64                                                          4/12
  Upgrading        : openssl-1:1.1.1wa-10.oe2203sp4.x86_64                                                              5/12
  Upgrading        : openssl-devel-1:1.1.1wa-10.oe2203sp4.x86_64                                                        6/12
  Cleanup          : openssl-1:1.1.1wa-7.oe2203sp4.x86_64                                                               7/12
  Cleanup          : openssh-clients-8.8p1-30.oe2203sp4.x86_64                                                          8/12
  Cleanup          : openssl-devel-1:1.1.1wa-7.oe2203sp4.x86_64                                                         9/12
  Cleanup          : openssh-8.8p1-30.oe2203sp4.x86_64                                                                 10/12
  Running scriptlet: openssh-server-8.8p1-30.oe2203sp4.x86_64                                                          11/12
  Cleanup          : openssh-server-8.8p1-30.oe2203sp4.x86_64                                                          11/12
  Running scriptlet: openssh-server-8.8p1-30.oe2203sp4.x86_64                                                          11/12
Warning: The unit file, source configuration file or drop-ins of sshd.service changed on disk. Run 'systemctl daemon-reload' to reload units.

  Cleanup          : openssl-libs-1:1.1.1wa-7.oe2203sp4.x86_64                                                          12/12
  Running scriptlet: openssl-libs-1:1.1.1wa-7.oe2203sp4.x86_64                                                          12/12
  Verifying        : openssh-8.8p1-34.oe2203sp4.x86_64                                                                   1/12
  Verifying        : openssh-8.8p1-30.oe2203sp4.x86_64                                                                   2/12
  Verifying        : openssh-clients-8.8p1-34.oe2203sp4.x86_64                                                           3/12
  Verifying        : openssh-clients-8.8p1-30.oe2203sp4.x86_64                                                           4/12
  Verifying        : openssh-server-8.8p1-34.oe2203sp4.x86_64                                                            5/12
  Verifying        : openssh-server-8.8p1-30.oe2203sp4.x86_64                                                            6/12
  Verifying        : openssl-1:1.1.1wa-10.oe2203sp4.x86_64                                                               7/12
  Verifying        : openssl-1:1.1.1wa-7.oe2203sp4.x86_64                                                                8/12
  Verifying        : openssl-devel-1:1.1.1wa-10.oe2203sp4.x86_64                                                         9/12
  Verifying        : openssl-devel-1:1.1.1wa-7.oe2203sp4.x86_64                                                         10/12
  Verifying        : openssl-libs-1:1.1.1wa-10.oe2203sp4.x86_64                                                         11/12
  Verifying        : openssl-libs-1:1.1.1wa-7.oe2203sp4.x86_64                                                          12/12

Upgraded:
  openssh-8.8p1-34.oe2203sp4.x86_64              openssh-clients-8.8p1-34.oe2203sp4.x86_64              openssh-server-8.8p1-34.oe2203sp4.x86_64
  openssl-1:1.1.1wa-10.oe2203sp4.x86_64          openssl-devel-1:1.1.1wa-10.oe2203sp4.x86_64            openssl-libs-1:1.1.1wa-10.oe2203sp4.x86_64

Complete!
[root@localhost /]#
```

   2. 修改配置文件（记得备份），在 `/etc/ssh/sshd-config` 中添加/修改相关信息：

```
Port 22
UsePrivilegeSeparation no
PasswordAuthentication yes_
PermitRootLogin yes
AllowUsers dzh
-- INSERT --
```

```
Protocol 2
LogLevel VERBOSE
PubkeyAuthentication yes
IgnoreRhosts yes
HostbasedAuthentication no
PermitEmptyPasswords no
PermitUserEnvironment no
Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,chacha20-poly1305@openssh.com
ClientAliveCountMax 0
Banner /etc/issue.net
MACs hmac-sha2-512,hmac-sha2-512-etm@openssh.com,hmac-sha2-256,hmac-sha2-256-etm@openssh.com
AllowTcpForwarding yes
GatewayPorts no
PermitTunnel no
KexAlgorithms curve25519-sha256,curve25519-sha256@libssh.org,diffie-hellman-group-exchange-sha256
HostbasedAcceptedKeytypes ssh-ed25519,ssh-ed25519-cert-v01@openssh.com,rsa-sha2-256,rsa-sha2-512
"/etc/ssh/sshd_config" 166L, 5055B written
[root@localhost ~]# systemctl restart sshd
[root@localhost ~]#
```

   3. 重启服务：`systemctl restart sshd.service`；

4. 使用vscode连接。

```
Welcome to 5.10.0-216.0.0.115.oe2203sp4.x86_64

System information as of time:  2025年 05月 13日 星期二 18:12:12 CST

System load:    1.13
Memory used:    28.8%
Swap used:      0%
Usage On:       9%
IP address:     192.168.31.133
Users online:   1
To run a command as administrator(user "root"),use "sudo <command>".
[dzh@localhost ~]$
```

第二部分 编译更新内核

1. 备份：把 `/boot` 的内容压缩放到本地存储；
2. 下载源码，在 https://gitee.com/openeuler/kernel/releases 选择最新版本下载；

CVE-2024-35976	#I9QRIQ:CVE-2024-35976
CVE-2024-35997	#I9QRN6:CVE-2024-35997

最后提交信息为： !8066drm/amd: Fix UBSAN array-index-out-of-bounds for SMU7

下载

下载 Source code (zip)

下载 Source code (tar.gz)

3. 传到 `openeuler` 中，运行 `tar vxf kernel-4.19.90-2405.5.0.tar.gz` 进行解压（内容很多，截取一部分）；

```
kernel-4.19.90-2405.5.0/virt/kvm/arm/vgic/vgic.h
kernel-4.19.90-2405.5.0/virt/kvm/async_pf.c
kernel-4.19.90-2405.5.0/virt/kvm/async_pf.h
kernel-4.19.90-2405.5.0/virt/kvm/coalesced_mmio.c
kernel-4.19.90-2405.5.0/virt/kvm/coalesced_mmio.h
kernel-4.19.90-2405.5.0/virt/kvm/eventfd.c
kernel-4.19.90-2405.5.0/virt/kvm/irqchip.c
kernel-4.19.90-2405.5.0/virt/kvm/kvm_main.c
kernel-4.19.90-2405.5.0/virt/kvm/vfio.c
kernel-4.19.90-2405.5.0/virt/kvm/vfio.h
kernel-4.19.90-2405.5.0/virt/lib/
kernel-4.19.90-2405.5.0/virt/lib/Kconfig
kernel-4.19.90-2405.5.0/virt/lib/Makefile
kernel-4.19.90-2405.5.0/virt/lib/irqbypass.c
[dzh@localhost ~]$
```

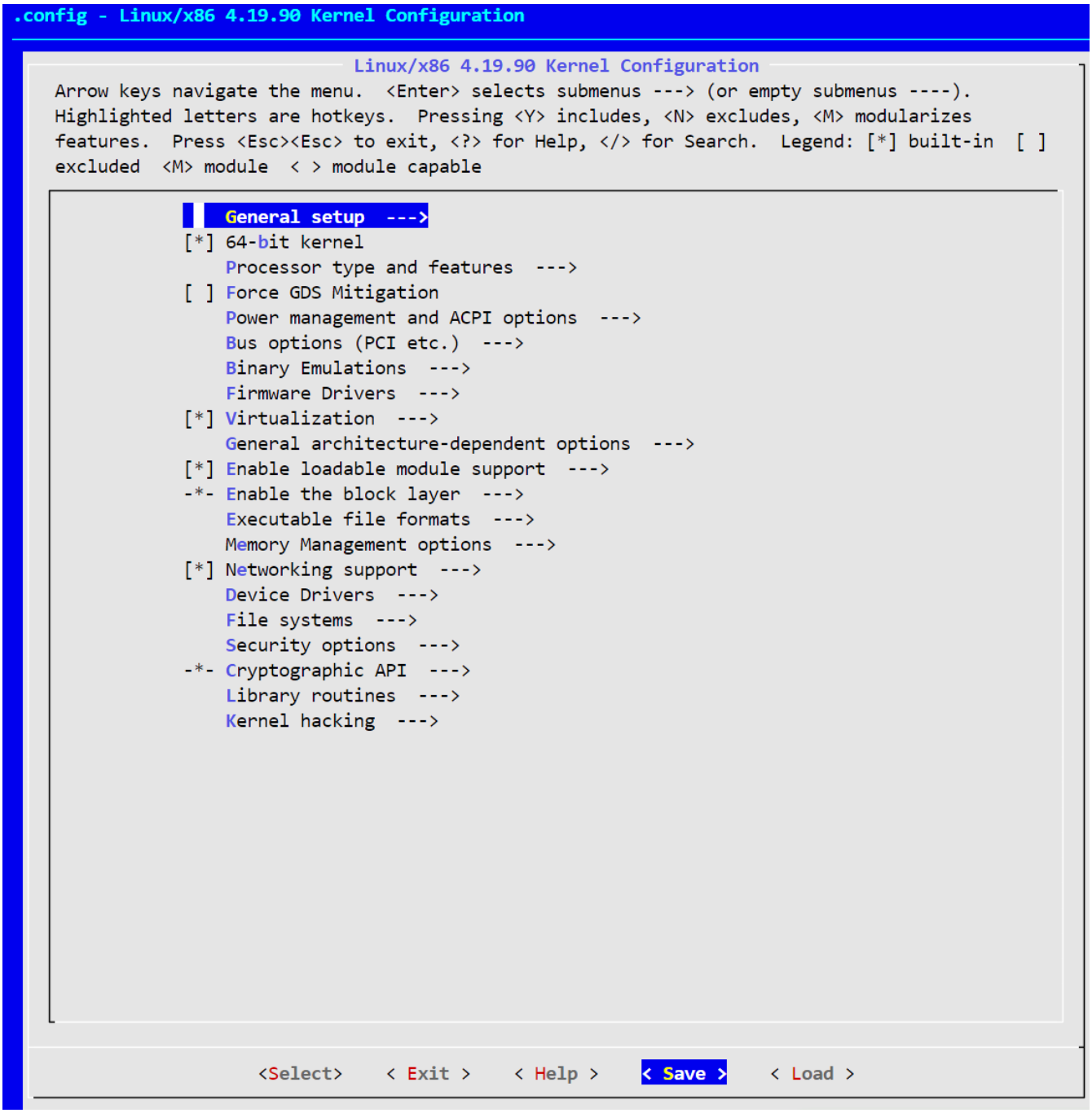
4. `make mrproper`：清理过去内核编译产生的文件,第一次编译可以不使用；
5. 备份原配置文件；

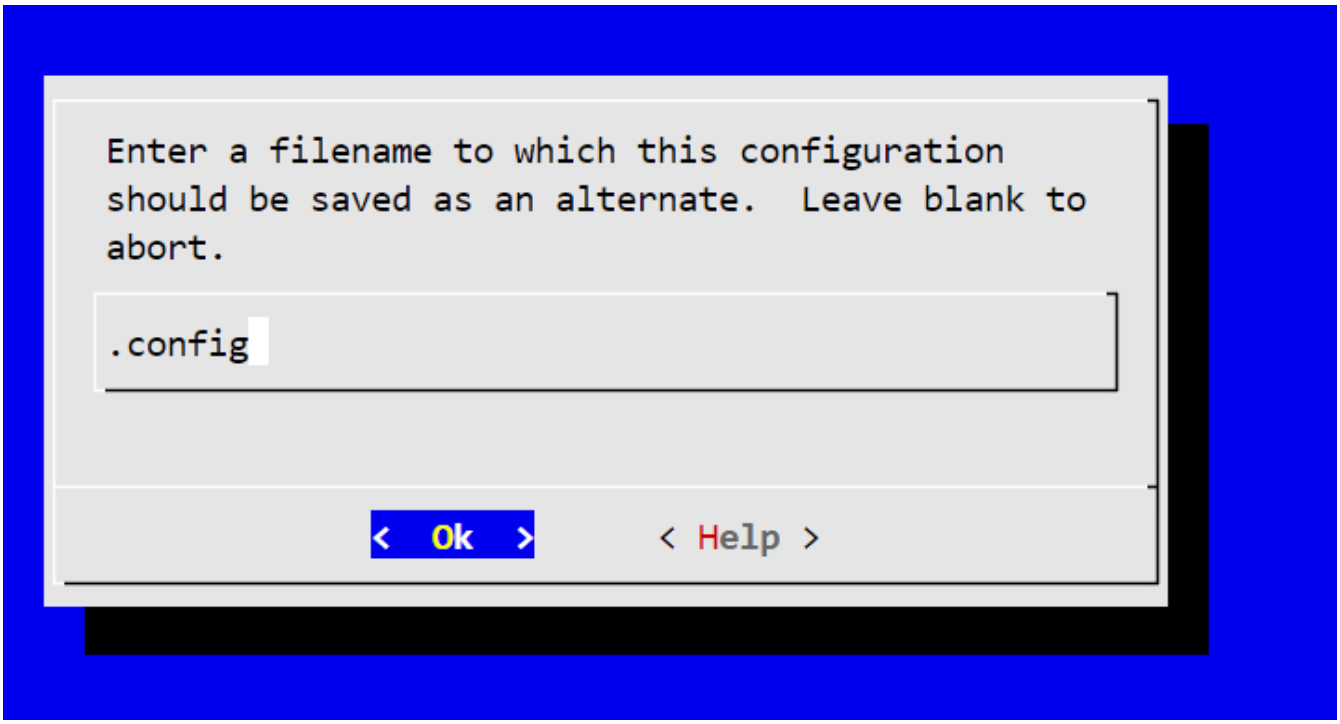
```
[dzh@localhost ~]$ ls /boot
config-5.10.0-216.0.0.115.oe2203sp4.x86_64
dracut
efi
grub2
initramfs-0-rescue-61e953f7f28243d89a7b4f9549c6168c.img
initramfs-5.10.0-216.0.0.115.oe2203sp4.x86_64.img
initramfs-5.10.0-216.0.0.115.oe2203sp4.x86_64kdump.img
loader
lost+found
symvers-5.10.0-216.0.0.115.oe2203sp4.x86_64.gz
System.map-5.10.0-216.0.0.115.oe2203sp4.x86_64
vmlinuz-0-rescue-61e953f7f28243d89a7b4f9549c6168c
vmlinuz-5.10.0-216.0.0.115.oe2203sp4.x86_64
[dzh@localhost ~]$ cp -v /boot/config-5.10.0-216.0.0.115.oe2203sp4.x86_64 ./config_bak
'/boot/config-5.10.0-216.0.0.115.oe2203sp4.x86_64' -> './config_bak'
```

6. 执行依赖安装，这里的 `ncurses-devel` 是一个用于创建终端用户界面(TUI)的开源库，提供了一种跨平台的，以文本方式显示的用户界面编程接口，使开发人员可以创建具有图形用户界面活动的命令行应用程序；

```
[root@localhost kernel-4.19.90-2405.5.0]# yum install ncurses-devel
Last metadata expiration check: 0:07:41 ago on 2025年05月13日 星期二 18时26分01秒.
Dependencies resolved.
================================================================================
 Package                 Architecture        Version                  Repository        Size
================================================================================
Installing:
 ncurses-devel           x86_64              6.3-15.oe2203sp4         OS                78 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 78 k
Installed size: 370 k
Is this ok [y/N]: y
Downloading Packages:
ncurses-devel-6.3-15.oe2203sp4.x86_64.rpm                  118 kB/s |  78 kB     00:00
--------------------------------------------------------------------------------
Total                                                       81 kB/s |  78 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : ncurses-devel-6.3-15.oe2203sp4.x86_64                  1/1
  Running scriptlet: ncurses-devel-6.3-15.oe2203sp4.x86_64                  1/1
  Verifying        : ncurses-devel-6.3-15.oe2203sp4.x86_64                  1/1

Installed:
  ncurses-devel-6.3-15.oe2203sp4.x86_64

Complete!
[root@localhost kernel-4.19.90-2405.5.0]#
```

7. 使用 `make menuconfig` 命令进行配置生成 `.config` 文件，此处使用默认的配置并未修改，选择save生成 `.config` 文件；


.config - Linux/x86 4.19.90 Kernel Configuration
Linux/x86 4.19.90 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
excluded <M> module < > module capable
General setup --->
[*] 64-bit kernel
Processor type and features --->
[ ] Force GDS Mitigation
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
[*] Virtualization --->
General architecture-dependent options --->
[*] Enable loadable module support --->
-*- Enable the block layer --->
Executable file formats --->
Memory Management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Security options --->
-*- Cryptographic API --->
Library routines --->
Kernel hacking --->
<Select> < Exit > < Help > < Save > < Load >




```
[root@localhost kernel-4.19.90-2405.5.0]# ls -a
.       .clang-format  crypto         .get_maintainer.ignore  ipc      lib          mm        security
..      .cocciconfig   Documentation  .gitattributes         kabi     LICENSES     net       sound
arch    .config        drivers        .gitignore              Kbuild   .mailmap     README    tools
block   COPYING        firmware       include                 Kconfig  MAINTAINERS  samples   usr
certs   CREDITS        fs             init                    kernel   Makefile     scripts   virt
[root@localhost kernel-4.19.90-2405.5.0]#
```

8. 安装编译所需的依赖 `yum install elfutils-libelf-devel openssl-devel bc`；

```
[root@localhost kernel-4.19.90-2405.5.0]# yum install elfutils-libelf-devel openssl-devel bc
Last metadata expiration check: 0:18:47 ago on 2025年05月13日 星期二 18时26分01秒.
Package elfutils-devel-0.185-20.oe2203sp4.x86_64 is already installed.
Package openssl-devel-1:1.1.1wa-10.oe2203sp4.x86_64 is already installed.
Package bc-1.07.1-12.oe2203sp4.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@localhost kernel-4.19.90-2405.5.0]#
```

9. 开始编译（部分截取）`make`；

```
[root@localhost kernel-4.19.90-2405.5.0]# make
  HOSTCC  scripts/kconfig/conf.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --syncconfig Kconfig
  SYSTBL  arch/x86/include/generated/asm/syscalls_32.h
  SYSHDR  arch/x86/include/generated/asm/unistd_32_ia32.h
  SYSHDR  arch/x86/include/generated/asm/unistd_64_x32.h
  SYSTBL  arch/x86/include/generated/asm/syscalls_64.h
  HYPERCALLS arch/x86/include/generated/asm/xen-hypercalls.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_32.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_64.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_x32.h
  HOSTCC  arch/x86/tools/relocs_32.o
  HOSTCC  arch/x86/tools/relocs_64.o
  HOSTCC  arch/x86/tools/relocs_common.o
  HOSTLD  arch/x86/tools/relocs
  UPD     include/config/kernel.release
  WRAP    arch/x86/include/generated/uapi/asm/bpf_perf_event.h
  WRAP    arch/x86/include/generated/uapi/asm/poll.h
  WRAP    arch/x86/include/generated/asm/dma-contiguous.h
  WRAP    arch/x86/include/generated/asm/early_ioremap.h
  WRAP    arch/x86/include/generated/asm/export.h
  WRAP    arch/x86/include/generated/asm/mcs_spinlock.h
  WRAP    arch/x86/include/generated/asm/mm-arch-hooks.h
  UPD     include/generated/uapi/linux/version.h
  UPD     include/generated/utsrelease.h
```

10. 安装模块（部分截取）`make modules_install`；

```
[root@localhost kernel-4.19.90-2405.5.0]# make modules_install
  INSTALL arch/x86/crypto/blowfish-x86_64.ko
  INSTALL arch/x86/crypto/camellia-aesni-avx-x86_64.ko
  INSTALL arch/x86/crypto/camellia-aesni-avx2.ko
  INSTALL arch/x86/crypto/camellia-x86_64.ko
  INSTALL arch/x86/crypto/cast5-avx-x86_64.ko
  INSTALL arch/x86/crypto/cast6-avx-x86_64.ko
  INSTALL arch/x86/crypto/chacha20-x86_64.ko
  INSTALL arch/x86/crypto/crc32-pclmul.ko
  INSTALL arch/x86/crypto/crc32c-intel.ko
  INSTALL arch/x86/crypto/crct10dif-pclmul.ko
  INSTALL arch/x86/crypto/des3_ede-x86_64.ko
  INSTALL arch/x86/crypto/ghash-clmulni-intel.ko
  INSTALL arch/x86/crypto/poly1305-x86_64.ko
  INSTALL arch/x86/crypto/serpent-avx-x86_64.ko
  INSTALL arch/x86/crypto/serpent-avx2.ko
  INSTALL arch/x86/crypto/serpent-sse2-x86_64.ko
  INSTALL arch/x86/crypto/sha512-ssse3.ko
  INSTALL arch/x86/crypto/twofish-avx-x86_64.ko
  INSTALL arch/x86/crypto/twofish-x86_64-3way.ko
```

11. 安装内核（部分截取）`make install`，完成安装后查看 `/boot` 发现已经完成；

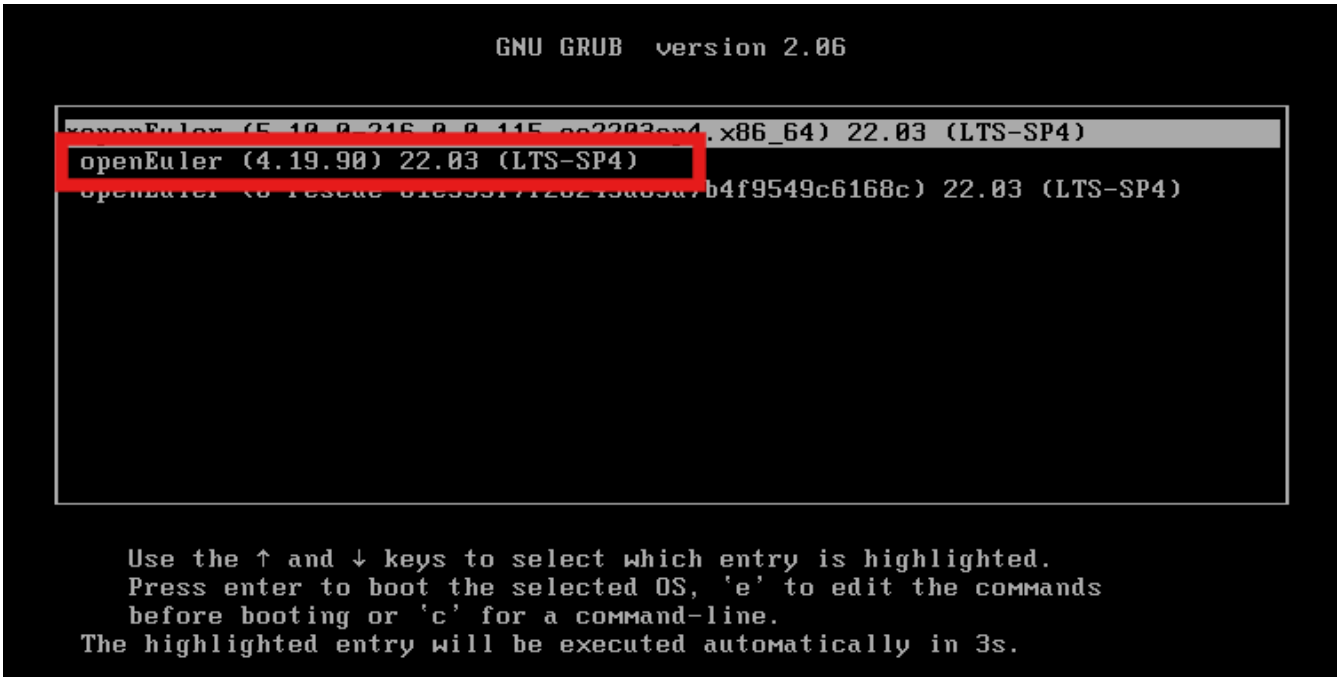
```
[root@localhost kernel-4.19.90-2405.5.0]# make install
sh ./arch/x86/boot/install.sh 4.19.90 arch/x86/boot/bzImage \
        System.map "/boot"
```

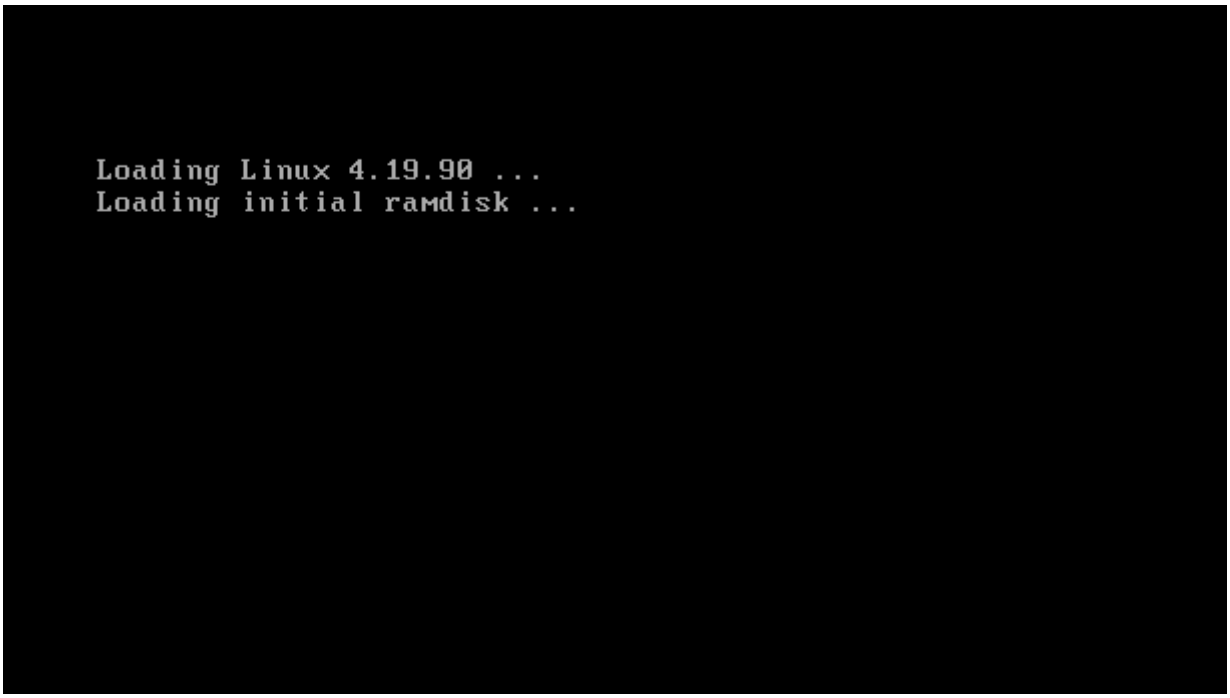
```
[root@localhost kernel-4.19.90-2405.5.0]# ll /boot
总用量 236420
-rw-r--r--. 1 root root   212338  6月 27  2024 config-5.10.0-216.0.0.115.oe2203sp4.x86_64
drwxr-xr-x. 2 root root     4096  6月 23  2024 dracut
drwxr-xr-x. 3 root root     4096  5月 13 16:59 efi
drwx------. 6 root root     4096  5月 14 16:58 grub2
-rw-------. 1 root root 76992264  5月 13 17:06 initramfs-0-rescue-61e953f7f28243d89a7b4f9549c6168c.img
-rw-------. 1 root root 66618858  5月 14 16:58 initramfs-4.19.90.img
-rw-------. 1 root root 25971194  5月 13 17:09 initramfs-5.10.0-216.0.0.115.oe2203sp4.x86_64.img
-rw-------. 1 root root 30882304  5月 13 17:17 initramfs-5.10.0-216.0.0.115.oe2203sp4.x86_64kdump.img
drwxr-xr-x. 3 root root     4096  5月 13 17:03 loader
drwx------. 2 root root    16384  5月 13 16:58 lost+found
-rw-r--r--. 1 root root   370801  6月 27  2024 symvers-5.10.0-216.0.0.115.oe2203sp4.x86_64.gz
lrwxrwxrwx. 1 root root       24  5月 14 16:55 System.map -> /boot/System.map-4.19.90
-rw-r--r--. 1 root root  3985844  5月 14 16:55 System.map-4.19.90
-rw-r--r--. 1 root root  5175717  6月 27  2024 System.map-5.10.0-216.0.0.115.oe2203sp4.x86_64
lrwxrwxrwx. 1 root root       21  5月 14 16:55 vmlinuz -> /boot/vmlinuz-4.19.90
-rw-r--r--. 1 root root 11589408  5月 13 17:06 vmlinuz-0-rescue-61e953f7f28243d89a7b4f9549c6168c
-rw-r--r--. 1 root root  8652608  5月 14 16:55 vmlinuz-4.19.90
-rw-r--r--. 1 root root 11589408  6月 27  2024 vmlinuz-5.10.0-216.0.0.115.oe2203sp4.x86_64
[root@localhost kernel-4.19.90-2405.5.0]#
```

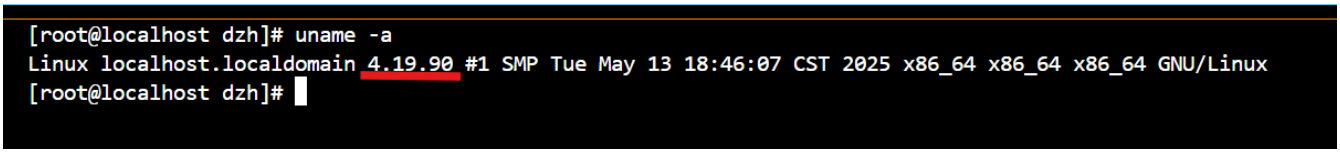
12. 更新引导；

```
[root@localhost boot]# ls grub2/
device.map  fonts  grub.cfg  grubenv  i386-pc  locale  themes
[root@localhost boot]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
File descriptor 19 (/home/dzh/.vscode-server/data/logs/20250514T165425/ptyhost.log) leaked on vgs invocation. Parent PID 12583: /usr/sbin/grub2-probe
File descriptor 20 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteTelemetry.log) leaked on vgs invocation. Parent PID 12583: /usr/sbin/grub2-probe
File descriptor 22 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteagent.log) leaked on vgs invocation. Parent PID 12583: /usr/sbin/grub2-probe
File descriptor 19 (/home/dzh/.vscode-server/data/logs/20250514T165425/ptyhost.log) leaked on vgs invocation. Parent PID 12583: /usr/sbin/grub2-probe
File descriptor 20 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteTelemetry.log) leaked on vgs invocation. Parent PID 12583: /usr/sbin/grub2-probe
File descriptor 22 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteagent.log) leaked on vgs invocation. Parent PID 12583: /usr/sbin/grub2-probe
Found linux image: /boot/vmlinuz-5.10.0-216.0.0.115.oe2203sp4.x86_64
Found initrd image: /boot/initramfs-5.10.0-216.0.0.115.oe2203sp4.x86_64.img
Found linux image: /boot/vmlinuz-4.19.90
Found initrd image: /boot/initramfs-4.19.90.img
Found linux image: /boot/vmlinuz-0-rescue-61e953f7f28243d89a7b4f9549c6168c
Found initrd image: /boot/initramfs-0-rescue-61e953f7f28243d89a7b4f9549c6168c.img
File descriptor 19 (/home/dzh/.vscode-server/data/logs/20250514T165425/ptyhost.log) leaked on vgs invocation. Parent PID 12864: /usr/sbin/grub2-probe
File descriptor 20 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteTelemetry.log) leaked on vgs invocation. Parent PID 12864: /usr/sbin/grub2-probe
File descriptor 22 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteagent.log) leaked on vgs invocation. Parent PID 12864: /usr/sbin/grub2-probe
File descriptor 19 (/home/dzh/.vscode-server/data/logs/20250514T165425/ptyhost.log) leaked on vgs invocation. Parent PID 12864: /usr/sbin/grub2-probe
File descriptor 20 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteTelemetry.log) leaked on vgs invocation. Parent PID 12864: /usr/sbin/grub2-probe
File descriptor 22 (/home/dzh/.vscode-server/data/logs/20250514T165425/remoteagent.log) leaked on vgs invocation. Parent PID 12864: /usr/sbin/grub2-probe
Adding boot menu entry for UEFI Firmware Settings ...
done
[root@localhost boot]#
```

13. 重启发现已经成功安装新的内核，使用 `uname -a` 查看验证；







第三部分 基础操作系统实验

内核模块编程

内核模块编程：指的是在操作系统内核层面编写代码，以扩展内核功能的一种技术。它允许开发者在不重新编译整个内核的情况下，动态地向运行中的内核加载或卸载代码片段。这些代码片段被称为内核模块（Kernel Modules）。本实验实现向正在运行的内核中加载和卸载 `Hello World` 程序。

1. 编写 `.c` 和 `Makefile` 文件：

```c
// Hello_world.c

#include <linux/init.h>   // 包含了 module_init() 和 module_exit() 宏
#include <linux/module.h> // 包含了加载模块所需要的大量符号和函数定义
#include <linux/kernel.h> // 包含了 printk() 等内核相关的函数

// 模块加载时调用的函数
static int __init hello_init(void) {
    printk(KERN_INFO "Hello, World! from the kernel module.\n");
    return 0; // 返回 0 表示模块加载成功
}

// 模块卸载时调用的函数
static void __exit hello_exit(void) {
    printk(KERN_INFO "Goodbye, World! from the kernel module.\n");
}

// 注册初始化和退出函数
module_init(hello_init);
module_exit(hello_exit);

// 模块许可声明 (GPL 是最常见的)
MODULE_LICENSE("GPL");
// 模块作者
MODULE_AUTHOR("dzh");
// 模块描述
MODULE_DESCRIPTION("Hello World kernel module");
// 模块版本
MODULE_VERSION("0.1");
```

```
# Makefile

# KERNELRELEASE 变量在内核构建系统递归调用此 Makefile 时会被设置
ifneq ($(KERNELRELEASE),)
    # 如果 KERNELRELEASE 不为空 (即内核构建系统调用时)，定义要编译成模块的目标文件。
    # 'obj-m' 表示编译成可加载模块 (.ko 文件)。
    obj-m := Hello_world.o
else
    # 如果 KERNELRELEASE 为空 (即我们直接在命令行执行 'make' 时)，执行这里的代码。

    # KERNELDIR: 指向内核源代码或内核头文件的目录，'?=' 表示如果 KERNELDIR 变量尚未定义，则将其赋值为后面的路径。
    KERNELDIR ?= /usr/lib/modules/$(shell uname -r)/build

    # PWD: 当前 Makefile 所在的目录的绝对路径。
    PWD := $(shell pwd)

# 'default' 是一个默认目标。只输入 'make' 时，这个目标会被执行。
default:
        # 调用 'make' 命令。
        # '-C $(KERNELDIR)' 表示切换到 $(KERNELDIR) 目录去执行 make。
        # 'M=$(PWD)' 表示要构建的外部模块的源代码位于 $(PWD) 目录 (即当前目录)。
        # 'modules' 是内核构建系统定义的一个目标，用于编译模块。
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif

# '.PHONY' 声明 'clean' 是一个伪目标，而不是一个文件名。这确保了即使当前目录下存在名为 'clean' 的文件，'make clean' 仍然会执行其定义的命令。
.PHONY: clean

# 'clean' 目标：用于清除编译过程中生成的各种文件。
clean:
    # '-rm' 中的 '-' 表示即使 rm 命令执行失败 (例如文件不存在)，make 也不会停止。
    # 删除模块编译过程中生成的临时文件和最终的 .ko 文件。
    # *.mod.c: 模块符号版本控制相关文件
    # *.o: 目标文件
    # *.order: Module.order 文件，记录模块依赖顺序
    # *.symvers: Module.symvers 文件，记录模块导出的符号
    # *.ko: 编译生成的内核模块文件
    -rm -f *.mod.c *.o *.order *.symvers *.ko *.mod Module.markers
modules.builtin
```

2. 执行 `make` 进行编译；完成后查看目录确定成功；

```
[root@localhost Kernel_Module_Programming]# make
make -C /usr/lib/modules/4.19.90/build M=/home/dzh/Kernel_Module_Programming modules
make[1]: 进入目录“/home/dzh/kernel-4.19.90-2405.5.0”
  CC [M]  /home/dzh/Kernel_Module_Programming/Hello_world.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/dzh/Kernel_Module_Programming/Hello_world.mod.o
  LD [M]  /home/dzh/Kernel_Module_Programming/Hello_world.ko
make[1]: 离开目录“/home/dzh/kernel-4.19.90-2405.5.0”
[root@localhost Kernel_Module_Programming]#
```

```
[root@localhost Kernel_Module_Programming]# ll
总用量 132
-rw-r--r--. 1 root root   866  5月 14 17:41 Hello_world.c
-rw-r--r--. 1 root root 56128  5月 14 18:09 Hello_world.ko
-rw-r--r--. 1 root root   843  5月 14 18:09 Hello_world.mod.c
-rw-r--r--. 1 root root 44624  5月 14 18:09 Hello_world.mod.o
-rw-r--r--. 1 root root 12760  5月 14 18:09 Hello_world.o
-rw-r--r--. 1 root root   266  5月 14 18:09 Makefile
-rw-r--r--. 1 root root    58  5月 14 18:09 modules.order
-rw-r--r--. 1 root root     0  5月 14 18:09 Module.symvers
[root@localhost Kernel_Module_Programming]#
```

3. 依次进行加载模块、查看模块、查看打印信息、卸载模块、查看模块（无结果证明成功卸载）、查看打印信息。

```
[root@localhost Kernel_Module_Programming]# insmod Hello_world.ko
[root@localhost Kernel_Module_Programming]# lsmod | grep Hello_world
Hello_world            16384  0
[root@localhost Kernel_Module_Programming]# dmesg | tail -n 1
[ 5355.089764] Hello, World! from the kernel module.
[root@localhost Kernel_Module_Programming]# rmmod Hello_world
[root@localhost Kernel_Module_Programming]# dmesg | tail -n 1
[ 5381.072948] Goodbye, World! from the kernel module.
[root@localhost Kernel_Module_Programming]# lsmod | grep Hello_world
[root@localhost Kernel_Module_Programming]#
```

进程管理

1. 创建内核级线程，功能为每2秒打印一次"Dzh is printing something."，编译，`Makefile` 内容如图所示，依次进行加载模块、卸载模块、查看打印信息验证每2s打印一次信息；

```c
#include <linux/kthread.h>    // 提供内核线程的相关函数
#include <linux/module.h>     // 提供模块的初始化与清除相关宏
#include <linux/delay.h>      // 提供延时函数

MODULE_LICENSE("GPL");        // 指定模块许可证，使用GPL以避免加载时的内核“污染”警告

#define BUF_SIZE 20           // 定义缓冲区大小

// 定义一个指向内核线程任务结构的指针，用于保存线程的句柄
static struct task_struct *myThread = NULL;

// 内核线程的主函数
static int print(void *data)
{
    // 当未收到线程终止信号时持续执行
    while (!kthread_should_stop()) {
        // 打印信息到内核日志
        printk("Dzh is printing something.");
        // 睡眠2秒
        msleep(2000);
    }

    // 接收到终止信号后，线程正常退出
    return 0;
}

// 模块初始化函数，模块加载时执行
static int __init kthread_init(void)
{
    printk("Create kernel thread!\n");

    // 创建并启动一个内核线程，执行 print 函数
    // 参数说明：
    // - print: 指定上文编写的线程主函数
    // - NULL: 传递给线程函数的数据
    // - "dzh_print_thread": 线程名称
    myThread = kthread_run(print, NULL, "dzh_print_thread");

    return 0; // 返回0表示成功
}

// 模块退出函数，模块卸载时执行
static void __exit kthread_exit(void)
{
    printk("Kill new kthread.\n");

    // 如果线程已经创建，则发送终止信号并停止线程
    if (myThread)
        kthread_stop(myThread);  // kthread_stop 会阻塞等待线程退出
}
```

```
module_init(kthread_init);    // 指定模块加载时执行的函数
module_exit(kthread_exit);    // 指定模块卸载时执行的函数
```

```
[root@localhost manage_process]# cat Makefile
ifneq ($(KERNELRELEASE),)
        obj-m :=k_thread.o
else
        KERNELDIR ?=/usr/lib/modules/$(shell uname -r)/build
        PWD :=$(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
.PHONY:clean
clean:
        -rm *.mod.c *.o *.order *.symvers *.ko
[root@localhost manage_process]# ls
k_thread.c  Makefile
[root@localhost manage_process]# make
make -C /usr/lib/modules/4.19.90/build M=/home/dzh/manage_process modules
make[1]: 进入目录“/home/dzh/kernel-4.19.90-2405.5.0”
  CC [M]  /home/dzh/manage_process/k_thread.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/dzh/manage_process/k_thread.mod.o
  LD [M]  /home/dzh/manage_process/k_thread.ko
make[1]: 离开目录“/home/dzh/kernel-4.19.90-2405.5.0”
[root@localhost manage_process]# ls
k_thread.c  k_thread.ko  k_thread.mod.c  k_thread.mod.o  k_thread.o  Makefile  modules.order  Module.symvers
[root@localhost manage_process]#
```

```
[root@localhost manage_process]# insmod k_thread.ko
[root@localhost manage_process]# lsmod | grep k_thread
k_thread               16384  0
[root@localhost manage_process]# rmmod k_thread
[root@localhost manage_process]# dmesg -T | tail -n 10
[六 5月 17 10:33:39 2025] Kill new kthread.
[六 5月 17 10:35:17 2025] Create kernel thread!
[六 5月 17 10:35:17 2025] Dzh is printing something.
[六 5月 17 10:35:19 2025] Dzh is printing something.
[六 5月 17 10:35:21 2025] Dzh is printing something.
[六 5月 17 10:35:23 2025] Dzh is printing something.
[六 5月 17 10:35:25 2025] Dzh is printing something.
[六 5月 17 10:35:28 2025] Dzh is printing something.
[六 5月 17 10:35:30 2025] Dzh is printing something.
[六 5月 17 10:35:31 2025] Kill new kthread.
[root@localhost manage_process]#
```

2. 打印输出当前系统CPU负载情况，编译并加载模块，查看打印信息；

```c
#include <linux/module.h>     // 提供模块相关的宏和函数
#include <linux/fs.h>         // 提供文件操作函数，如 filp_open、filp_close

MODULE_LICENSE("GPL");        // 声明模块使用GPL许可证，避免内核模块污染警告

// 定义一个全局字符数组，用于保存读取到的 CPU 负载信息（前4个字符）
char tmp_cpu_load[5] = {'\0'};  // 初始化为空字符串

// 获取 /proc/loadavg 文件中的前4个字符（表示1分钟内的CPU负载平均值）
static int get_loadavg(void)
{
    struct file *fp_cpu;        // 用于指向打开的文件
    loff_t pos = 0;             // 文件读写位置（偏移量）
    char buf_cpu[10];           // 存储读取到的数据（最多读10字节）

    // 打开 /proc/loadavg 文件，O_RDONLY表示只读
    fp_cpu = filp_open("/proc/loadavg", O_RDONLY, 0);
    if (IS_ERR(fp_cpu)) {       // 判断打开是否失败
        printk("Failed to open loadavg file!\n");
        return -1;
    }

    // 从文件中读取内容到 buf_cpu 缓冲区
    kernel_read(fp_cpu, buf_cpu, sizeof(buf_cpu), &pos);

    // 将前4个字符（比如“0.15”）拷贝到 tmp_cpu_load，用于后续打印
    strncpy(tmp_cpu_load, buf_cpu, 4);

    // 关闭文件
    filp_close(fp_cpu, NULL);

    return 0;  // 成功返回0
}

// 模块加载时调用的函数
static int __init cpu_loadavg_init(void)
{
    printk("Start cpu_loadavg!\n");

    // 调用 get_loadavg 读取 CPU 负载，如果失败则返回错误
    if (0 != get_loadavg()) {
        printk("Failed to read loadavg file!\n");
        return -1;
    }

    // 打印读取到的CPU负载值
    printk("The cpu loadavg in one minute is: %s\n", tmp_cpu_load);

    return 0;  // 成功加载模块
}

// 模块卸载时调用的函数
```

```c
static void __exit cpu_loadavg_exit(void)
{
    printk("Exit cpu_loadavg!\n");  // 模块卸载时打印信息
}

// 指定模块的加载与卸载函数
module_init(cpu_loadavg_init);   // 加载模块时执行 cpu_loadavg_init
module_exit(cpu_loadavg_exit);   // 卸载模块时执行 cpu_loadavg_exit
```

```
[root@localhost cpu_load]# ls
cpu_load.c  Makefile
[root@localhost cpu_load]# cat Makefile
ifneq ($(KERNELRELEASE),)
        obj-m :=cpu_load.o
else
        KERNELDIR ?=/usr/lib/modules/$(shell uname -r)/build
        PWD :=$(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
.PHONY:clean
clean:
        -rm *.mod.c *.o *.order *.symvers *.ko
[root@localhost cpu_load]# make
make -C /usr/lib/modules/4.19.90/build M=/home/dzh/manage_process/cpu_load modules
make[1]: 进入目录“/home/dzh/kernel-4.19.90-2405.5.0”
  CC [M]  /home/dzh/manage_process/cpu_load/cpu_load.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/dzh/manage_process/cpu_load/cpu_load.mod.o
  LD [M]  /home/dzh/manage_process/cpu_load/cpu_load.ko
make[1]: 离开目录“/home/dzh/kernel-4.19.90-2405.5.0”
[root@localhost cpu_load]# ls
cpu_load.c  cpu_load.ko  cpu_load.mod.c  cpu_load.mod.o  cpu_load.o  Makefile  modules.order  Module.symvers
[root@localhost cpu_load]# insmod cpu_load.ko
[root@localhost cpu_load]# lsmod | grep cpu_load
cpu_load               16384  0
[root@localhost cpu_load]# rmmod c
cdrom            cpu_load         crc32c_intel     crc32_pclmul     crct10dif_pclmul
[root@localhost cpu_load]# rmmod cpu_load
[root@localhost cpu_load]# dmesg | tail -n 3
[ 1923.014282] Start cpu_loadavg!
[ 1923.014307] The cpu loadavg in one minute is: 0.16
[ 1945.731100] Exit cpu_loadavg!
[root@localhost cpu_load]#
```

3. 打印输出当前处于运行状态的进程的PID和名字，编译，加载，查看；

```c
#include <linux/module.h>          // 提供模块初始化/卸载的相关接口
#include <linux/sched/signal.h>    // 提供 for_each_process 宏定义，用于遍历进程
#include <linux/sched.h>           // 提供 task_struct 结构体定义和进程状态等信息

MODULE_LICENSE("GPL");             // 声明模块许可证为 GPL，防止内核警告

// 定义一个指向进程描述符的全局指针，用于遍历进程链表
struct task_struct *p;

// 模块初始化函数，模块加载时调用
static int __init process_info_init(void)
{
    printk("Start process_info!\n");

    // 遍历所有进程（使用 for_each_process 宏，底层其实是通过 init_task->tasks 链表遍历）
    for_each_process(p) {
        // 仅打印状态为运行态（state == 0，表示 TASK_RUNNING）的进程
        if (p->state == 0)
            printk("1) name: %s  2) pid: %d  3) state: %ld\n", p->comm, p->pid, p->state);
            // p->comm 是进程的名字
            // p->pid 是进程的 ID
            // p->state 是进程的状态（0 表示正在运行）
    }

    return 0; // 返回 0 表示模块加载成功
}

// 模块退出函数，模块卸载时调用
static void __exit process_info_exit(void)
{
    printk("Exit process_info!\n");
}

// 注册模块初始化与退出函数
module_init(process_info_init);  // 指定加载模块时调用的函数
module_exit(process_info_exit);  // 指定卸载模块时调用的函数
```

```
[root@localhost pid_info]# ls
Makefile  pid_info.c
[root@localhost pid_info]# cat Makefile
ifneq ($(KERNELRELEASE),)
        obj-m :=pid_info.o
else
        KERNELDIR ?=/usr/lib/modules/$(shell uname -r)/build
        PWD :=$(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
.PHONY:clean
clean:
        -rm *.mod.c *.o *.order *.symvers *.ko
[root@localhost pid_info]# make
make -C /usr/lib/modules/4.19.90/build M=/home/dzh/manage_process/pid_info modules
make[1]: 进入目录“/home/dzh/kernel-4.19.90-2405.5.0”
  CC [M]  /home/dzh/manage_process/pid_info/pid_info.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/dzh/manage_process/pid_info/pid_info.mod.o
  LD [M]  /home/dzh/manage_process/pid_info/pid_info.ko
make[1]: 离开目录“/home/dzh/kernel-4.19.90-2405.5.0”
[root@localhost pid_info]# ls
Makefile  modules.order  Module.symvers  pid_info.c  pid_info.ko  pid_info.mod.c  pid_info.mod.o  pid_info.o
[root@localhost pid_info]# insmod pid_info.ko
[root@localhost pid_info]# lsmod |grep pid_info
pid_info               16384  0
[root@localhost pid_info]# rmmod pid_info
[root@localhost pid_info]# dmesg | tail -n 3
[ 2252.763558] 1) name: rcu_sched  2) pid: 10  3) state: 0
[ 2252.763622] 1) name: insmod  2) pid: 9208  3) state: 0
[ 2270.786993] Exit process_info!
[root@localhost pid_info]# dmesg | tail -n 6
[ 1923.014307] The cpu loadavg in one minute is: 0.16
[ 1945.731100] Exit cpu loadavg!
[ 2252.763550] Start process_info!
[ 2252.763558] 1) name: rcu_sched  2) pid: 10  3) state: 0
[ 2252.763622] 1) name: insmod  2) pid: 9208  3) state: 0
[ 2270.786993] Exit process_info!
[root@localhost pid_info]#
```

4. 使用 cgroup 实现限制CPU核数；
   1. cgroup(Control Groups) 是Linux提供的一种功能，用于对进程的资源使用进行限制、控制与监视，比如 CPU、内存、磁盘、网络等。
   2. 挂载tmpfs格式的cgroup文件夹，以及cpuset管理子系统，然后设置cpu核数（此处只使用CPU0核）；

```
[root@localhost cgroup_for_cpus]# mkdir cgroup
[root@localhost cgroup_for_cpus]# mount -t tmpfs tmpfs cgroup
[root@localhost cgroup_for_cpus]# cd cgroup/
[root@localhost cgroup]# mkdir cpuset
[root@localhost cgroup]# mount -t cgroup -o cpuset cpuset /cgroup/cpuset
mount: /cgroup/cpuset: 挂载点不存在.
[root@localhost cgroup]# mount -t cgroup -o cpuset cpuset cpuset
[root@localhost cgroup]# ls
cpuset
[root@localhost cgroup]# cd cpuset/
[root@localhost cpuset]# mkdir mycpuset
[root@localhost cpuset]# cd mycpuset/
[root@localhost mycpuset]# ls
cgroup.clone_children  cpuset.cpu_exclusive  cpuset.effective_cpus  cpuset.mem_exclusive  cpuset.memory_migrate   cpuset.memory_spread_page  cpuset.mems           cpuset.sched_load_balance       notify_on_release
cgroup.procs           cpuset.cpus           cpuset.effective_mems  cpuset.mem_hardwall   cpuset.memory_pressure  cpuset.memory_spread_slab  cpuset.preferred_cpus  cpuset.sched_relax_domain_level  tasks
[root@localhost mycpuset]# cat cat cpuset.mems
cat: cat: No such file or directory

[root@localhost mycpuset]# echo 0 > cpuset.mems
[root@localhost mycpuset]# echo 0 > cpuset.cpus
[root@localhost mycpuset]# cat cpuset.mems
0
[root@localhost mycpuset]# cat cpuset.cpus
0
```

   3. 安装cgroup（yum install libcgroup），编译死循环程序，指定在cpuset子系统的mycpuset控制组中运行，运行后通过 top 查看PID，利用 taskset -pc <PID> 查看可用核。

```
[root@localhost while]# gcc while.c -o while
[root@localhost while]# ls
while  while.c
[root@localhost while]# cd ../cgroup/cpuset/mycpuset/
[root@localhost mycpuset]# cgexec -g cpuset:mycpuset ../../../while
/while
^Z
[5]+  已停止               cgexec -g cpuset:mycpuset ../../../while
/while
[root@localhost mycpuset]# cgexec -g cpuset:mycpuset ../../../while
/while
```

```
[root@localhost dzh]# taskset -p 14741
pid 14741 的当前亲和力掩码: 1
[root@localhost dzh]# taskset -pc 14741
pid 14741 的当前亲和力列表: 0
[root@localhost dzh]#
```

```
top - 11:47:54 up  1:31,  1 user,  load average: 1.26, 0.57, 0.31
Tasks: 354 total,   3 running, 346 sleeping,   5 stopped,   0 zombie
%Cpu(s): 25.2 us,  0.6 sy,  0.0 ni, 73.4 id,  0.1 wa,  0.7 hi,  0.1 si,  0.0 st
MiB Mem :   3404.1 total,   1474.8 free,   1314.4 used,    904.3 buff/cache
MiB Swap:   4020.0 total,   4020.0 free,      0.0 used.   2089.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  14741 root      20   0    2436    900    812 R  99.0   0.0   1:30.80 while
   2883 dzh       20   0   52.4g 614468  48248 S   0.7  17.6   1:16.90 node
     12 root      rt   0       0      0      0 S   0.3   0.0   0:00.04 migration/0
   1203 root      20   0  356588  33000  16056 S   0.3   0.9   0:14.14 tuned
   2514 dzh       20   0 1210080  75564  42240 S   0.3   2.2   0:28.80 node
   2853 dzh       20   0   39380  20800  14332 S   0.3   0.6   0:19.34 code-848b80+
```

内存管理

1. 使用 `kmalloc` 分配不同大小的内存，并打印指针地址，同时根据内核的内存分配也可以知道使用 `kmalloc` 分配的内存是内核空间的一部分；

```c
#include <linux/module.h>     // 提供模块宏，如 module_init, module_exit 等
#include <linux/slab.h>       // 提供 kmalloc 和 kfree 动态内存分配函数

MODULE_LICENSE("GPL");        // 指定模块许可证为 GPL，避免内核模块污染警告

// 定义两个全局指针，用于保存分配得到的内存地址
unsigned char *kmallocmem1;
unsigned char *kmallocmem2;

// 模块加载时执行的函数
static int __init mem_module_init(void)
{
    printk("Start kmalloc!\n");

    // 使用 kmalloc 分配 1024 字节的内核内存，GFP_KERNEL 表示普通分配（可睡眠）
    kmallocmem1 = (unsigned char *)kmalloc(1024, GFP_KERNEL);
    if (kmallocmem1 != NULL){
        // 打印分配到的内存地址
        printk(KERN_ALERT "kmallocmem1 addr = %lx\n", (unsigned
long)kmallocmem1);
    } else {
        // 内存分配失败
        printk("Failed to allocate kmallocmem1!\n");
    }

    // 再次使用 kmalloc 分配 8192 字节内存
    kmallocmem2 = (unsigned char *)kmalloc(8192, GFP_KERNEL);
    if (kmallocmem2 != NULL){
        // 打印分配到的内存地址
        printk(KERN_ALERT "kmallocmem2 addr = %lx\n", (unsigned
long)kmallocmem2);
    } else {
        printk("Failed to allocate kmallocmem2!\n");
    }

    return 0;  // 模块初始化成功
}

// 模块卸载时执行的函数
static void __exit mem_module_exit(void)
{
    // 释放之前通过 kmalloc 分配的内存
    kfree(kmallocmem1);
    kfree(kmallocmem2);

    printk("Exit kmalloc!\n");
}

// 注册模块的初始化和退出函数
module_init(mem_module_init);   // 指定加载模块时调用的函数
module_exit(mem_module_exit);   // 指定卸载模块时调用的函数
```

```
[root@localhost kmalloc]# ls
kmalloc.c  Makefile
[root@localhost kmalloc]# cat Makefile
ifneq ($(KERNELRELEASE),)
        obj-m :=kmalloc.o
else
        KERNELDIR ?=/usr/lib/modules/$(shell uname -r)/build
        PWD :=$(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
.PHONY:clean
clean:
        -rm *.mod.c *.o *.order *.symvers *.ko
[root@localhost kmalloc]# make
make -C /usr/lib/modules/4.19.90/build M=/home/dzh/manage_memory/kmalloc modules
make[1]: 进入目录“/home/dzh/kernel-4.19.90-2405.5.0”
  CC [M]  /home/dzh/manage_memory/kmalloc/kmalloc.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/dzh/manage_memory/kmalloc/kmalloc.mod.o
  LD [M]  /home/dzh/manage_memory/kmalloc/kmalloc.ko
make[1]: 离开目录“/home/dzh/kernel-4.19.90-2405.5.0”
[root@localhost kmalloc]# insmod kmalloc.ko
[root@localhost kmalloc]# dmesg | tail -n 4
[ 4313.386670] systemd-rc-local-generator[10565]: /etc/rc.d/rc.local is not marked executable, skipping.
[ 6016.577384] Start kmalloc!
[ 6016.577446] kmallocmem1 addr = ffff8e9451a52c00
[ 6016.579617] kmallocmem2 addr = ffff8e9441c40000
[root@localhost kmalloc]# rmmod kmalloc
[root@localhost kmalloc]# dmesg | tail -n 4
[ 6016.577384] Start kmalloc!
[ 6016.577446] kmallocmem1 addr = ffff8e9451a52c00
[ 6016.579617] kmallocmem2 addr = ffff8e9441c40000
[ 6043.757817] Exit kmalloc!
[root@localhost kmalloc]#
```

```
[root@localhost kmalloc]# cd /usr/lib/modules/4.19.90/
[root@localhost 4.19.90]# ls
build   modules.alias      modules.builtin          modules.builtin.bin      modules.dep      modules.devname  modules
kernel  modules.alias.bin  modules.builtin.alias.bin  modules.builtin.modinfo  modules.dep.bin  modules.order    modules
[root@localhost 4.19.90]# cd build/
[root@localhost build]# ls
arch        certs    crypto         firmware  init  Kbuild   lib          Makefile         modules.builtin.modinfo  net
block       COPYING  Documentation  fs        ipc   Kconfig  LICENSES     mm               modules.order            READ
built-in.a  CREDITS  drivers        include   kabi  kernel   MAINTAINERS  modules.builtin  Module.symvers           samp
[root@localhost build]# cd Documentation/x86/x86_64
[root@localhost x86_64]# ls
00-INDEX  5level-paging.txt  boot-options.txt  cpu-hotplug-spec  fake-numa-for-cpusets  machinecheck  mm.txt  uefi.txt
[root@localhost x86_64]# cat mm.txt

Virtual memory map with 4 level page tables:

0000000000000000 - 00007fffffffffff (=47 bits) user space, different per mm
hole caused by [47:63] sign extension
ffff800000000000 - ffff87ffffffffff (=43 bits) guard hole, reserved for hypervisor
ffff880000000000 - ffff887fffffffff (=39 bits) LDT remap for PTI
ffff888000000000 - ffffc87fffffffff (=64 TB) direct mapping of all phys. memory
ffffc88000000000 - ffffc8ffffffffff (=39 bits) hole
ffffc90000000000 - ffffe8ffffffffff (=45 bits) vmalloc/ioremap space
ffffe90000000000 - ffffe9ffffffffff (=40 bits) hole
ffffea0000000000 - ffffeaffffffffff (=40 bits) virtual memory map (1TB)
... unused hole ...
ffffec0000000000 - fffffbffffffffff (=44 bits) kasan shadow memory (16TB)
... unused hole ...
                                vaddr_end for KASLR
fffffe0000000000 - fffffe7fffffffff (=39 bits) cpu_entry_area mapping
fffffe8000000000 - fffffeffffffffff (=39 bits) LDT remap for PTI
ffffff0000000000 - ffffff7fffffffff (=39 bits) %esp fixup stacks
... unused hole ...
ffffffef00000000 - ffffffffffffffff (=64 GB) EFI region mapping space
... unused hole ...
ffffffff80000000 - ffffffff9fffffff (=512 MB)  kernel text mapping, from phys 0
ffffffffa0000000 - ffffffffffffffff (1520 MB) module mapping space
```

```
[fixmap start]   - ffffffffff5fffff kernel-internal fixmap range
ffffffffff600000 - ffffffffff600fff (=4 kB) legacy vsyscall ABI
ffffffffffe00000 - ffffffffffffffff (=2 MB) unused hole
```

2. 使用 `vmalloc` 分别分配不同大小的内存，打印指针地址，同时根据内核的内存分配也可以知道使用 `vmalloc` 分配的内存依然是内核空间的一部分（与 `kmalloc` 处于同一块区域）；

```c
#include <linux/module.h>       // 提供模块初始化、退出相关宏
#include <linux/vmalloc.h>      // 提供 vmalloc 和 vfree 函数

MODULE_LICENSE("GPL");          // 使用GPL许可证，防止加载时的内核警告

// 定义三个全局指针变量，用于保存分配得到的内存地址
unsigned char *vmallocmem1;
unsigned char *vmallocmem2;
unsigned char *vmallocmem3;

// 模块加载时调用的初始化函数
static int __init mem_module_init(void)
{
    printk("Start vmalloc!\n");

    // 使用 vmalloc 分配 8192 字节（8 KB）虚拟内存
    vmallocmem1 = (unsigned char *)vmalloc(8192);
    if (vmallocmem1 != NULL) {
        printk("vmallocmem1 addr = %lx\n", (unsigned long)vmallocmem1);
    } else {
        printk("Failed to allocate vmallocmem1!\n");
    }

    // 分配 1048576 字节（1 MB）内存
    vmallocmem2 = (unsigned char *)vmalloc(1048576);
    if (vmallocmem2 != NULL) {
        printk("vmallocmem2 addr = %lx\n", (unsigned long)vmallocmem2);
    } else {
        printk("Failed to allocate vmallocmem2!\n");
    }

    // 分配 67108864 字节（64 MB）内存
    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;  // 模块加载成功
}

// 模块卸载时调用的退出函数
static void __exit mem_module_exit(void)
{
    // 释放之前分配的虚拟内存（必须与 vmalloc 配对使用）
    vfree(vmallocmem1);
    vfree(vmallocmem2);
    vfree(vmallocmem3);

    printk("Exit vmalloc!\n");
}
```

```
// 注册模块加载与卸载函数
module_init(mem_module_init);
module_exit(mem_module_exit);
```

```
[root@localhost vmalloc]# ls
Makefile  vmalloc.c
[root@localhost vmalloc]# cat Makefile
ifneq ($(KERNELRELEASE),)
        obj-m :=vmalloc.o
else
        KERNELDIR ?=/usr/lib/modules/$(shell uname -r)/build
        PWD :=$(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
.PHONY:clean
clean:
        -rm *.mod.c *.o *.order *.symvers *.ko
[root@localhost vmalloc]# make
make -C /usr/lib/modules/4.19.90/build M=/home/dzh/manage_memory/vmalloc modules
make[1]: 进入目录“/home/dzh/kernel-4.19.90-2405.5.0”
  CC [M]  /home/dzh/manage_memory/vmalloc/vmalloc.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/dzh/manage_memory/vmalloc/vmalloc.mod.o
  LD [M]  /home/dzh/manage_memory/vmalloc/vmalloc.ko
make[1]: 离开目录“/home/dzh/kernel-4.19.90-2405.5.0”
[root@localhost vmalloc]# insmod vmalloc.ko
[root@localhost vmalloc]# lsmod | grep vmalloc
vmalloc                16384  0
[root@localhost vmalloc]# rmmod vmalloc
[root@localhost vmalloc]# dmesg | tail -n 5
[ 6913.952570] Start vmalloc!
[ 6913.952578] vmallocmem1 addr = ffffafe680677000
[ 6913.952644] vmallocmem2 addr = ffffafe6825f1000
[ 6913.968336] vmallocmem3 addr = ffffafe690569000
[ 6931.610713] Exit vmalloc!
[root@localhost vmalloc]#
```

3. 既然分配的内存都处于内核空间，那么这两种方法有什么不同呢？经过查找，`kmalloc` 分配的连续的物理地址，而 `vmalloc` 分配的则是连续的虚拟地址即物理上不一定连续，因此这种分配方式导致了前者适合分配较小的内存块，由于物理地址连续，访问时开销必然比较小，而后者适用于分配较大的内存块（因为前者很难找到较大的连续的物理内存）。

实验总结

本次实验中，我从头开始安装了OpenEuler系统，并通过下载源代码并编译的方式更新内核，同时在新的内核上完成了内核模块编程、进程管理、内存管理三部分基础实验，更深刻的理解了OS的各种设计与应用，同时学会了现代操作系统中与内核相关的常用命令，帮助我更好地将理论与实践相结合，加强了实践应用能力。