✌️

导学第二阶段作业

Page • 1 backlink • Tag

作业1

1.安装依赖：

安装 `flex` 工具。 `flex` 是一个生成词法分析器（lexical analyzer）的工具，它在编译 Linux 内核时是必需的。

Bash Unwrap Copy

```
sudo apt-get install flex
```

安装 `bison` 工具。 `bison` 是一个生成语法分析器（parser）的工具，它在编译 Linux 内核时是必需的。

Bash

```
sudo apt-get install bison
```

安装 LLD 工具。 `ld.lld` 是 LLVM 的链接器，它在使用 LLVM/Clang 编译时是必需的。

Bash

```
sudo apt-get install lld
```

安装 `libelf` 库

Bash

```
sudo apt-get install libelf-dev
```

安装 OpenSSL 库

```bash
sudo apt-get install libssl-dev
```

`bc` 是一个基本计算器工具，内核编译过程中需要使用。

```bash
sudo apt-get install bc
```

2.指定 Rust 版本

安装指定版本的 Rust 工具链（包括标准库源代码）：

```bash
rustup toolchain install 1.62.0 --component rust-src
```

3.进入Linux文件夹，使用如下命令进行编译：

```bash
make x86_64_defconfig
```

用来生成一个基于 x86_64 架构的默认配置文件。这个配置文件包含了适用于大多数 x86_64 系统的默认选项。执行这个命令后，会在当前目录下生成一个 `.config` 文件。

```bash
make LLVM=1 menuconfig
```

这一行命令使用 `menuconfig` 进行内核配置，允许用户通过图形界面（基于 ncurses 的终端界面）来配置内核选项。其中 `LLVM=1` 表示使用 LLVM/Clang 作为编译器，而不是默认的 GCC。

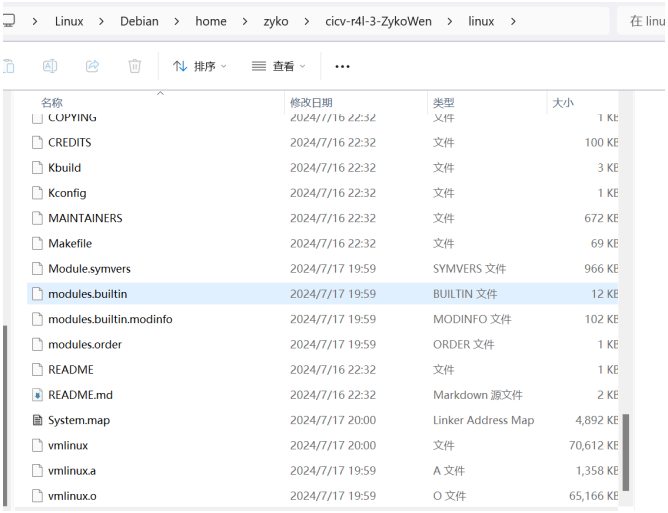
```bash
#set the following config to yes
General setup
        ---> [*] Rust support
```

启用“Rust support”选项。这将会在内核中启用对 Rust 编程语言的支持。

保证rustc的version: 1.62.0和bindgen —version 0.56.0

```Bash
make LLVM=1 -j$(nproc)
```

这行命令用于编译内核，`LLVM=1` 再次指定使用 LLVM/Clang 作为编译器。`-j$(nproc)` 选项表示使用所有可用的 CPU 核心进行并行编译，以加快编译速度。`$(nproc)` 会自动替换为当前系统中的 CPU 核心数。



作业2

编译内核模块

进入src_e1000目录，执行以下命令，该文件夹内的代码将编译成一个内核模块

```Rust
$make LLVM=1
make -C ../linux M=$PWD
make[1]: Entering directory '/home/zyko/cicv-r4l-3-ZykoWen/linux'
  RUSTC [M] /home/zyko/cicv-r4l-3-ZykoWen/src_e1000/r4l_e1000_demo.o
  MODPOST /home/zyko/cicv-r4l-3-ZykoWen/src_e1000/Module.symvers
  CC [M]  /home/zyko/cicv-r4l-3-ZykoWen/src_e1000/r4l_e1000_demo.mod.o
```

```
  LD [M]  /home/zyko/cicv-r4l-3-ZykoWen/src_e1000/r4l_e1000_demo.ko
make[1]: Leaving directory '/home/zyko/cicv-r4l-3-ZykoWen/linux'
```

> Q: 在该文件夹中调用make LLVM=1，该文件夹内的代码将编译成一个内核模块。请结合你学到的知识，回答以下两个问题：

禁用e1000网卡

启动menuconfig进行配置

```Rust
make menuconfig
```

（配置路径Device Drivers > Network device support > Ethernet driver support > Intel devices, Intel(R) PRO/1000 Gigabit Ethernet support）并禁用

重新编译内核

进入 Linux 内核文件夹：

```Bash
make LLVM=1 -j$(nproc)
```

运行build_image.sh脚本

```Bash
./build_image.sh
```

进入 qemu 环境，加载驱动：

```Bash
insmod r4l_e1000_demo.ko
```

```
Please press Enter to activate this console.
~ # insmod r4l_e1000_demo.ko
[   31.091980] r4l_e1000_demo: loading out-of-tree module taints kernel.
[   31.098214] r4l_e1000_demo: Rust for linux e1000 driver demo (init)
[   31.098865] r4l_e1000_demo: Rust for linux e1000 driver demo (probe): None
[   31.318077] ACPI: \_SB_.LNKC: Enabled at IRQ 11
[   31.340800] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
[   31.342962] insmod (80) used greatest stack depth: 10904 bytes left
```

使用以下命令验证模块是否正确加载：

```Bash
lsmod | grep r4l_e1000_demo
```

```
~ # lsmod | grep r4l_e1000_demo
r4l_e1000_demo 40960 0 - Live 0xffffffffc0145000 (O)
```

配置联网：

```bash
ip link set eth0 up
ip addr add broadcast 10.0.2.255 dev eth0
ip addr add 10.0.2.15/255.255.255.0 dev eth0
ip route add default via 10.0.2.1
ping 10.0.2.2
```

```
~ # ip link set eth0 up
~ # ip addr add broadcast 10.0.2.255 dev eth0
[  294.096883] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
[  294.097340] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
ip: RTNETLINK answers: Invalid argument
~ # ip addr add 10.0.2.15/255.255.255.0 dev eth0
[  307.862007] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
[  307.862519] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
~ # ip route add default via 10.0.2.1
```

```
~ # ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2): 56 data bytes
[  322.888354] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=3, tdh=3, rd
[  322.889089] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  322.889348] r4l_e1000_demo: pending_irqs: 131
[  322.889944] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
[  322.892294] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=4, tdh=4, rd
[  322.892985] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  322.893219] r4l_e1000_demo: pending_irqs: 131
[  322.894158] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
64 bytes from 10.0.2.2: seq=0 ttl=255 time=12.782 ms
[  323.898260] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=5, tdh=5, rd
[  323.899109] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  323.899729] r4l_e1000_demo: pending_irqs: 131
[  323.900076] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
64 bytes from 10.0.2.2: seq=1 ttl=255 time=2.819 ms
[  324.901408] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=6, tdh=6, rd
[  324.902101] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  324.902500] r4l_e1000_demo: pending_irqs: 131
[  324.902846] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
```

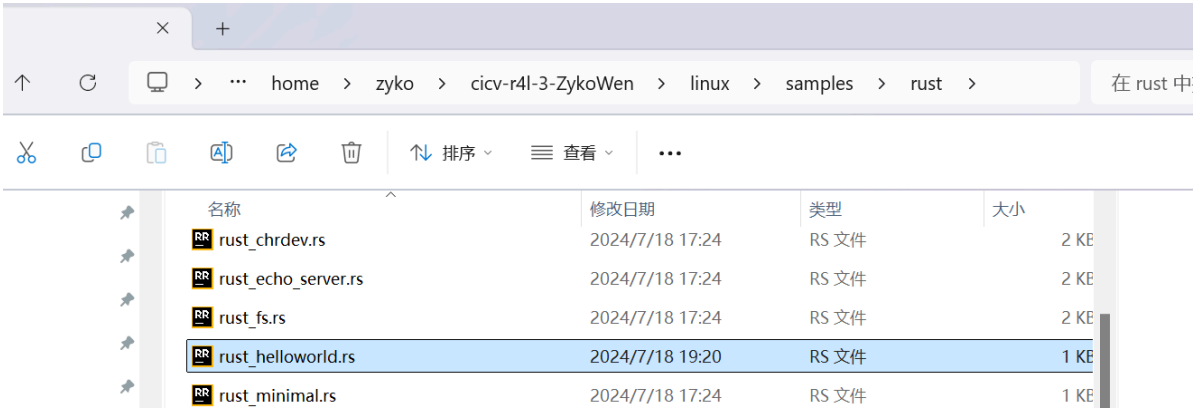
```
--- 10.0.2.2 ping statistics ---
39 packets transmitted, 39 packets received, 0% packet loss
round-trip min/avg/max = 1.759/2.547/12.782 ms
```

验证

```
~ # ifconfig
[  420.976280] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

~ # ping
BusyBox v1.36.1 (2024-07-18 17:26:27 CST) multi-call binary.

Usage: ping [OPTIONS] HOST

Send ICMP ECHO_REQUESTs to HOST

        -4,-6           Force IP or IPv6 name resolution
        -c CNT          Send only CNT pings
        -s SIZE         Send SIZE data bytes in packets (default 56)
        -i SECS         Interval
        -A              Ping as soon as reply is received
        -t TTL          Set TTL
        -I IFACE/IP     Source interface or IP address
        -W SEC          Seconds to wait for the first response (default 10)
                        (after all -c CNT packets are sent)
        -w SEC          Seconds until ping exits (default:infinite)
                        (can exit earlier with -c CNT)
        -q              Quiet, only display output at start/finish
        -p HEXBYTE      Payload pattern
~ # [  578.657671] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=4, tdh=4, rdt=7, rdh=0
[  578.658499] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  578.658852] r4l_e1000_demo: pending_irqs: 3
[  578.659271] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
```

作业3

将rust_helloworld放入rust文件下



设置Kconfig

```
Rust
config SAMPLE_RUST_HELLOWORLD
    tristate "Print Helloworld in Rust"
    help
          This option enables the My Rust Module. It is a minimal
sample that
          prints "Hello World from Rust module" on initialization.

          If unsure, say N.
```

设置Makefile

```Rust
obj-$(CONFIG_SAMPLE_RUST_HELLOWORLD)     += rust_helloworld.o
```

代码编译

如果你添加的配置正确，那么可以运行

```JavaScript
make LLVM=1 menuconfig
```

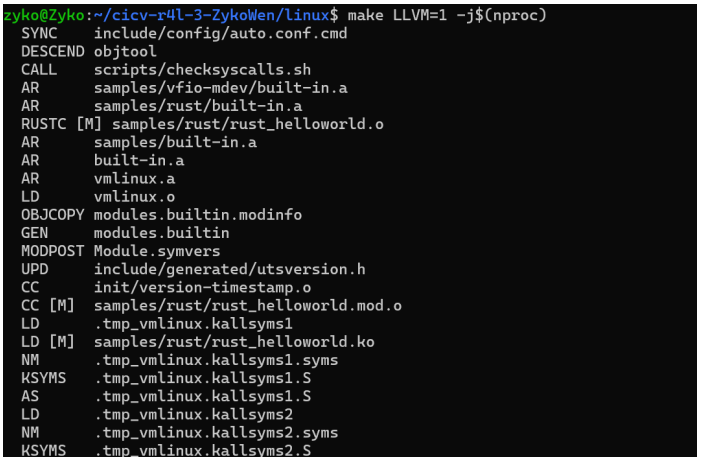
更改该模块的配置，使之编译成模块

```JavaScript
Kernel hacking
  ---> Sample Kernel code
      ---> Rust samples
              ---> <M>Print Helloworld in Rust (NEW)
```

重新编译内核

进入 Linux 内核文件夹：

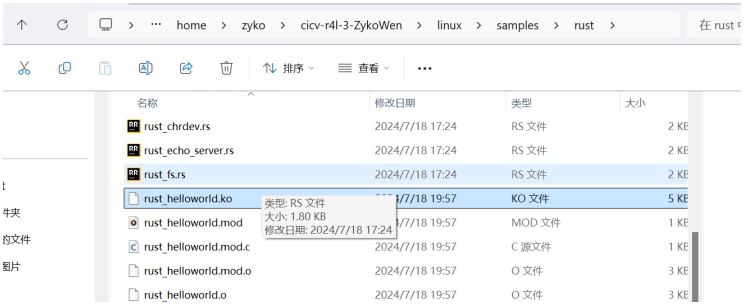
```Bash
make LLVM=1 -j$(nproc)
```

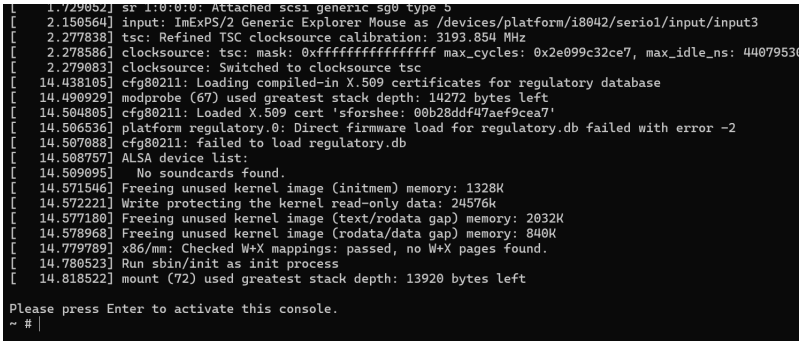


运行src_e1000/build_image.sh

```bash
./build_image.sh
```

测试样例






```
~ # ls
bin                                 root
dev                                 rust_helloworld.ko
etc                                 rust_helloworld.ko:Zone.Identifier
linuxrc                             sbin
proc                                sys
r4l_e1000_demo.ko                   usr
```

```
~ # insmod rust_helloworld.ko
[  360.784605] rust_helloworld: Hello World from Rust module
```

作业4

修改 rust for linux 库函数

- `linux/rust/kernel/net.rs`
- `linux/rust/kernel/pci.rs`

配置ping通

```
64 bytes from 10.0.2.2: seq=46 ttl=255 time=0.420 ms
64 bytes from 10.0.2.2: seq=47 ttl=255 time=0.428 ms
64 bytes from 10.0.2.2: seq=48 ttl=255 time=0.459 ms
64 bytes from 10.0.2.2: seq=49 ttl=255 time=0.400 ms
64 bytes from 10.0.2.2: seq=50 ttl=255 time=0.358 ms
64 bytes from 10.0.2.2: seq=51 ttl=255 time=0.428 ms
64 bytes from 10.0.2.2: seq=52 ttl=255 time=0.890 ms
^C
--- 10.0.2.2 ping statistics ---
53 packets transmitted, 53 packets received, 0% packet loss
round-trip min/avg/max = 0.353/0.718/8.338 ms
```

移除模块

```Rust
rmmod r4l_e1000_demo.ko
```

```
~ # rmmod r4l_e1000_demo.ko
[  182.934760] r4l_e1000_demo: Rust for linux e1000 driver demo (exit)
~ # rmmod r4l_e1000_demo.ko
rmmod: remove 'r4l_e1000_demo': No such file or directory
```

重新安装模块

```Rust
insmod r4l_e1000_demo.ko
```

重新配置ping通

```
64 bytes from 10.0.2.2: seq=22 ttl=255 time=0.482 ms
64 bytes from 10.0.2.2: seq=23 ttl=255 time=0.441 ms
64 bytes from 10.0.2.2: seq=24 ttl=255 time=0.428 ms
^C
--- 10.0.2.2 ping statistics ---
25 packets transmitted, 25 packets received, 0% packet loss
round-trip min/avg/max = 0.380/0.505/0.894 ms
```

作业5

修改函数

```Rust
fn write(this: &Self, _file: &file::File, reader: &mut impl
IoBufferReader, offset: u64) -> Result<usize> {
    let offset = offset.try_into()?;
    let mut vec = this.inner.lock();
    let len = core::cmp::min(reader.len(),
vec.len().saturating_sub(offset));
    reader.read_slice(&mut vec[offset..][..len])?;
    Ok(len)
}

fn read(this: &Self, _file: &file::File, writer: &mut impl
IoBufferWriter, offset: u64) -> Result<usize> {
    let offset = offset.try_into()?;
    let vec = this.inner.lock();
    let len = core::cmp::min(writer.len(),
vec.len().saturating_sub(offset));
    writer.write_slice(&vec[offset..][..len])?;
    Ok(len)
}
```

修改配置

```JavaScript
make LLVM=1 menuconfig
```

更改该模块的配置，使之编译成模块

```JavaScript
Kernel hacking
  ---> Sample Kernel code
      ---> Rust samples
              ---> <*>Character device (NEW)
```

重新编译内核

进入 Linux 内核文件夹：

```Bash
make LLVM=1 -j$(nproc)
```

测试

```
root@Zyko:~/cicv-r4l-3-ZykoWen/src_e1000# cd ..
root@Zyko:~/cicv-r4l-3-ZykoWen# cd linux
root@Zyko:~/cicv-r4l-3-ZykoWen/linux# make menuconfig
  HOSTCC  scripts/basic/fixdep
configuration written to .config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

root@Zyko:~/cicv-r4l-3-ZykoWen/linux# echo "Hello" > /dev/cicv
root@Zyko:~/cicv-r4l-3-ZykoWen/linux# cat /dev/cicv
Hello
```

Question:

- **作业5中的字符设备 `/dev/cicv` 是怎么创建的？它的设备号是多少？它是如何与我们写的字符设备驱动关联上的？**

  设备文件 `/dev/cicv` 通过设备号与字符设备驱动关联。当应用程序对 `/dev/cicv` 进行读写操作时，内核会通过设备号将这些操作路由到对应的字符设备驱动。在驱动代码中，使用 `chrdev::Registration` 注册字符设备，并指定了设备名称和设备号。

  ```rust
  let mut chrdev_reg = chrdev::Registration::new_pinned(name, 0, module)?;
  ```

  这段代码注册了一个名为 `name` 的字符设备，并指定了次设备号为 0。内核会为该设备分配一个主设备号。当加载驱动模块后，通过 `mknod` 命令创建的设备文件 `/dev/cicv` 就会与该驱动关联。当应用程序对 `/dev/cicv` 进行读写操作时，内核会调用驱动中的 `read` 和 `write` 函数处理这些操作。

小测验

添加环境变量

```rust
export R4L_EXP=/root/cicv-r4l-3-ZykoWen/r4l_experiment
//验证 R4L_EXP 环境变量是否正确设置：
echo $R4L_EXP
```

创建initramfs镜像

```Rust
mkdir -p cicv-r4l-3-ZykoWen/r4l_experiment/initramfs
cd $R4L_EXP/initramfs
# Create necessary directories
mkdir -p {bin,dev,etc,lib,lib64,mnt,proc,root,sbin,sys,tmp}

# Set Permission
chmod 1777 tmp

# Copy necessary device files from host, root privilege maybe
needed.
sudo cp -a /dev/{null,console,tty,ttyS0} dev/
```

将之前静态编译的busybox拷贝到initramf/bin下

```Rust
cd $R4L_EXP/initramfs

cp cp /root/cicv-r4l-3-ZykoWen/busybox-1.36.1/busybox  ./bin/
chmod +x bin/busybox
# Install busybox
bin/busybox --install bin
bin/busybox --install sbin
```

编写init脚本

```Rust
cd $R4L_EXP/initramfs

cat << EOF > init
#!/bin/busybox sh

# Mount the /proc and /sys filesystems.
mount -t proc none /proc
mount -t sysfs none /sys

# Boot real things.

# NIC up
ip link set eth0 up
ip addr add 10.0.2.15/24 dev eth0
ip link set lo up

# Wait for NIC ready
```

```
sleep 0.5

# Make the new shell as a login shell with -l option
# Only login shell read /etc/profile
setsid sh -c 'exec sh -l </dev/ttyS0 >/dev/ttyS0 2>&1'

EOF

chmod +x init
```

更多设置

```Rust
cd $R4L_EXP/initramfs

# name resolve
cat << EOF > etc/hosts
127.0.0.1    localhost
10.0.2.2     host_machine
EOF

# common alias
cat << EOF > etc/profile
alias ll='ls -l'
EOF

# busybox saves password in /etc/passwd directly, no /etc/shadow is
needed.
cat << EOF > etc/passwd
root:x:0:0:root:/root:/bin/bash
EOF

# group file
cat << EOF > etc/group
root:x:0:
EOF
```

构建initramfs镜像

```Rust
cd $R4L_EXP/initramfs

find . -print0 | cpio --null -ov --format=newc | gzip -9 >
../initramfs.cpio.gz
```

通过boot.sh脚本启动

```Rust
cd $R4L_EXP

# 以下是boot.sh的内容:
#!/bin/sh
kernel_image="../linux/arch/x86/boot/bzImage"

qemu-system-x86_64 \
-kernel $kernel_image \
-append "console=ttyS0" \
-initrd ./initramfs.cpio.gz \
-nographic

# 然后执行以下命令启动
chmod +x boot.sh
./boot.sh  # Press <C-A> x to terminate QEMU.
```

支持NFC

在主机上设置NFS服务器

注意$R4L_EXP

```Rust
sudo apt-get install nfs-kernel-server
sudo bash -c "echo \
'$R4L_EXP/driver     127.0.0.1(insecure,rw,sync,no_root_squash)' \
    >> /etc/exports"
sudo /etc/init.d/rpcbind restart
sudo /etc/init.d/nfs-kernel-server restart
```

在qemu执行

```Rust
# Add this line in init script. Put it just after the line of sleep
0.5.
mount -t nfs -o nolock host_machine:/host/cicv-r4l-3-
ZykoWen/r4l_experiment/driver /mnt

# 然后rebuild initramfs
cd $R4L_EXP/initramfs
```

```
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
../initramfs.cpio.gz
```

支持telnet server

首先设置 pts device node

```Rust
cd $R4L_EXP/initramfs

mkdir dev/pts
mknod -m 666 dev/ptmx c 5 2
# 同样在init脚本中设置自动挂载，在NFS设置后面加入
mount -t devpts devpts  /dev/pts
# 然后rebuild initramfs
cd $R4L_EXP/initramfs
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
../initramfs.cpio.gz
```

在boot.sh中加入一下qemu启动参数：

```Rust
-netdev user,id=host_net,hostfwd=tcp::7023-:23 \
-device e1000,mac=52:54:00:12:34:50,netdev=host_net \
```

开启telnet server：

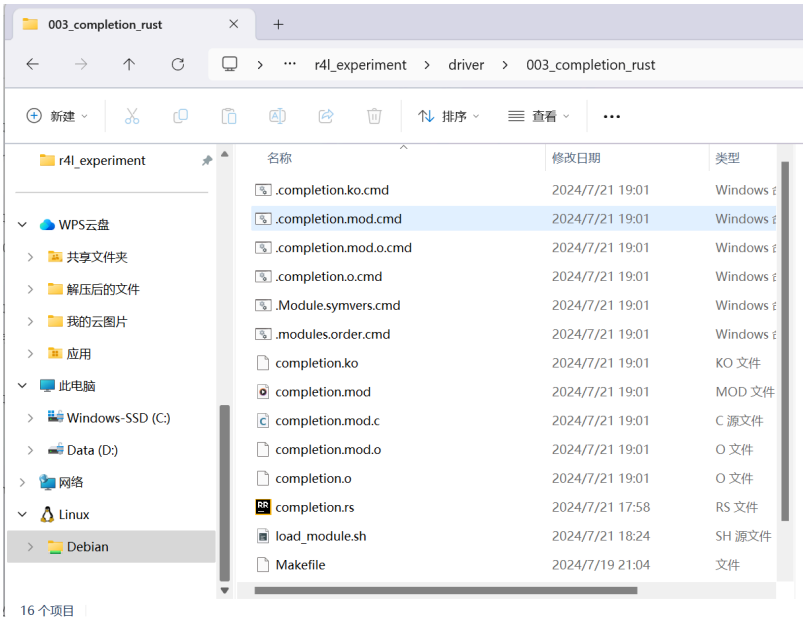
```Rust
# 同样在init脚本中设置自动启动，在telnetserver设置后面加入
telnetd -l /bin/sh
# 然后rebuild initramfs
cd $R4L_EXP/initramfs
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
../initramfs.cpio.gz
```

在本机通过telnet server连接qemu控制台

```Rust
telnet localhost 7023
```

重构测试

根据002_complement重构代码新建文件为003_complement_rust



将该代码块编译

JavaScript

```
make LLVM=1 -j$(nproc)
```

通过 `telnet` 链接虚拟环境：

需要首先在另一个命令行中执行./boot.sh脚本

Bash

```
telnet localhost 7023
```

加载模块

Bash

```
cd /mnt/003_completion_rust
```

执行脚本

Bash

```
./load_module.sh
```

```
~ # [   35.096735] completion: loading out-of-tree module taints kernel.
[   35.105163] completion: Rust character device sample (init)
```

测试

```
~ # echo "hello,world" > /dev/completion
~ # cat /dev/completion
hello,world
```