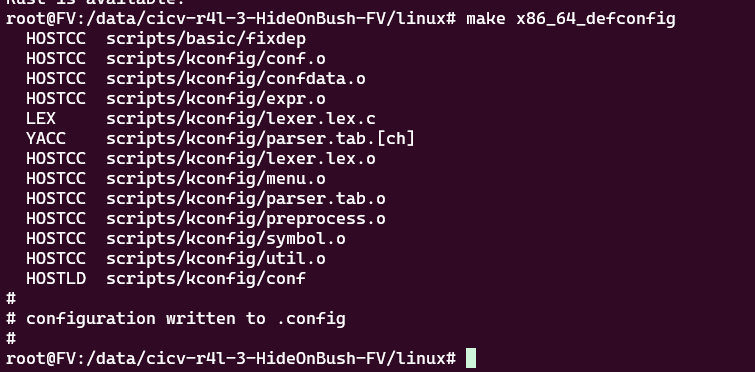
# 作业报告

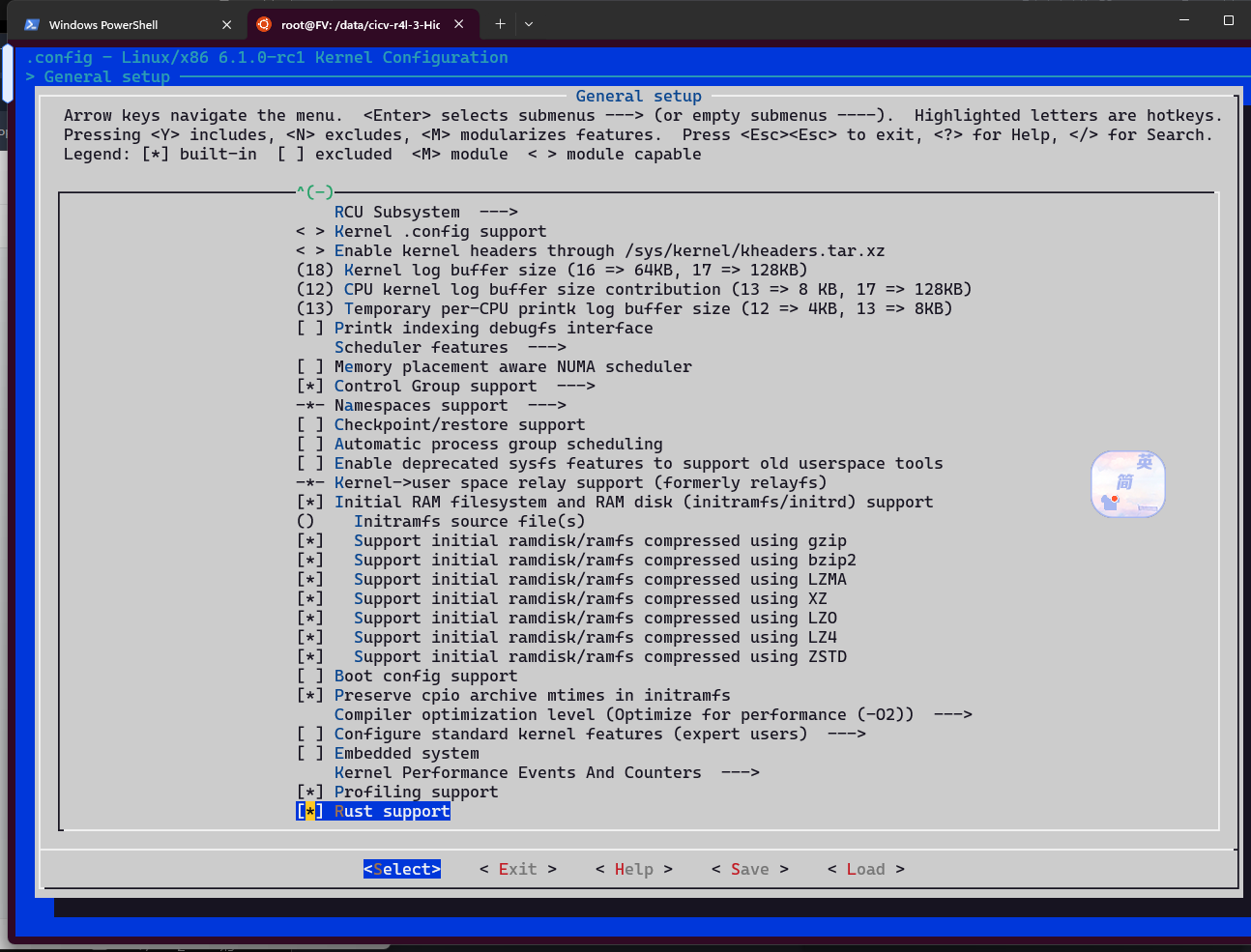
## 作业1：

### 步骤及截图

make x86\_64\_defconfig



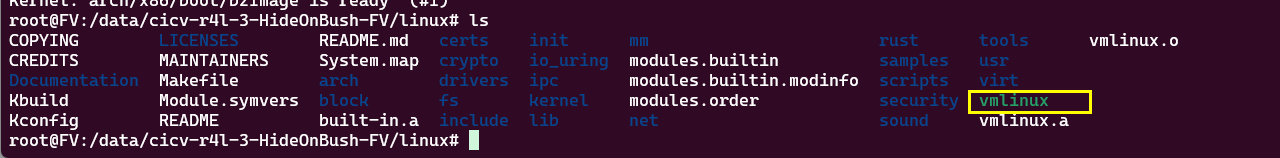
make LLVM=1 menuconfig



在 General setup 里选中 Rust Support

make LLVM=1 -j$(nproc)

### 结果

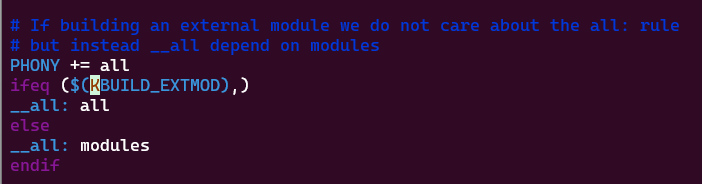


## 作业2

### 问答题

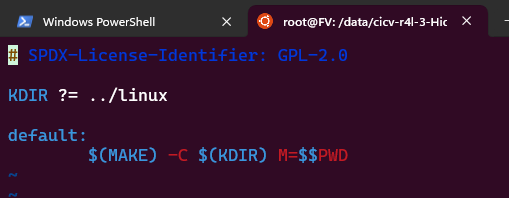
#### 第一题

1. 编译成内核模块，是在哪个文件中以哪条语句定义的？

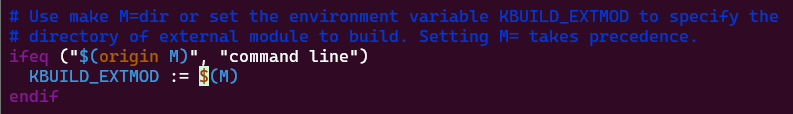
答：

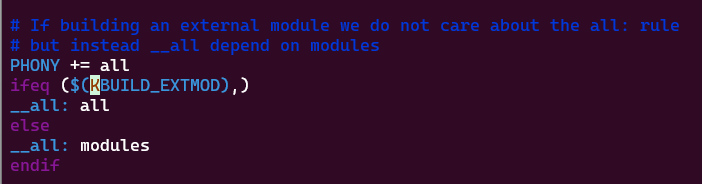
整体流程：

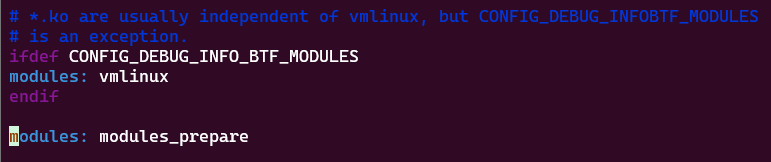
1. Src\_e1000/build\_image.sh 里 make 走Makefile

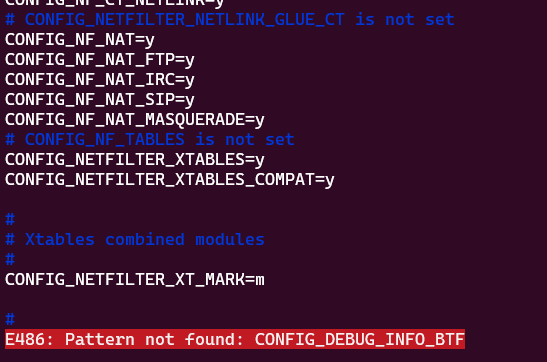


1. 进入 ../linux 查看 Makefile





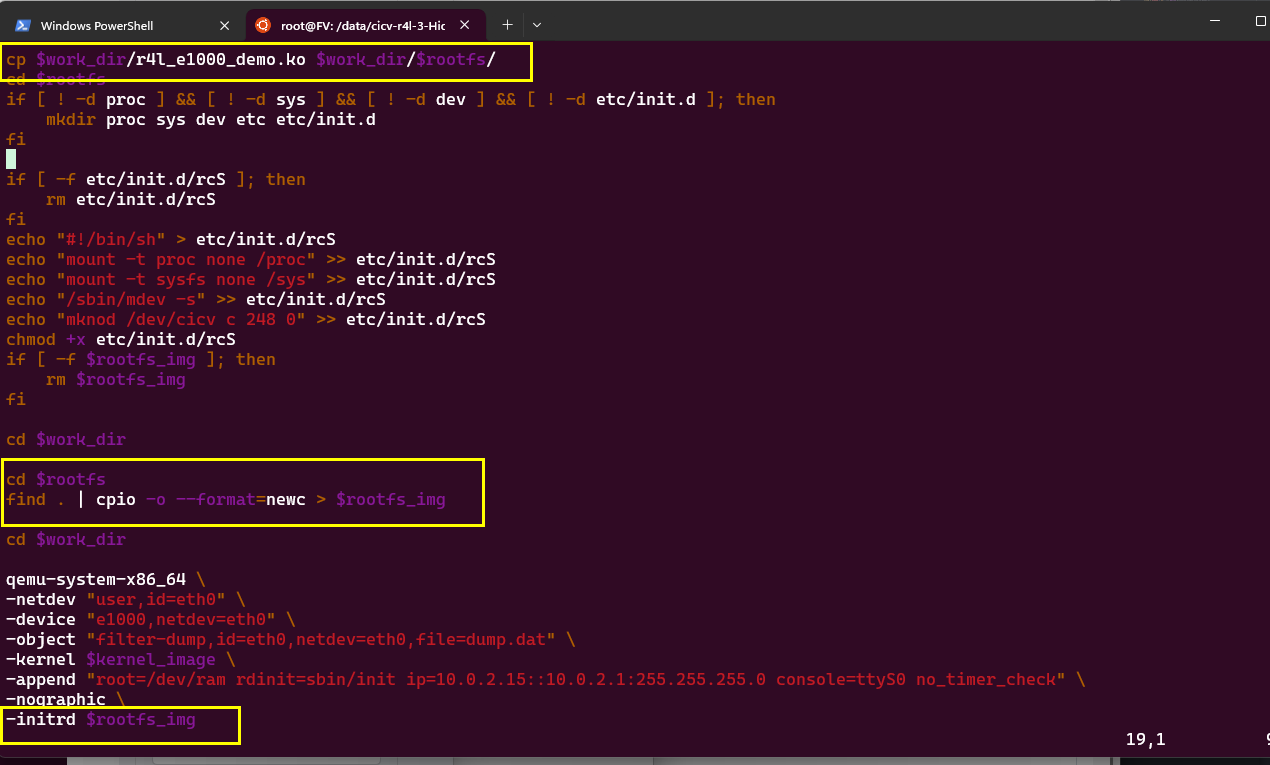


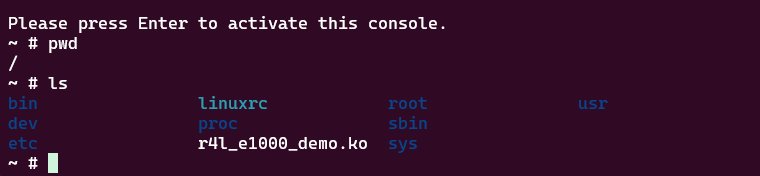
 .config文件里没有这个编译选项，所以使用的是 modules\_prepare

#### 第二题

1. 该模块位于独立的文件夹内，却能编译成Linux内核模块，这叫做out-of-tree module，请分析它是如何与内核代码产生联系的？

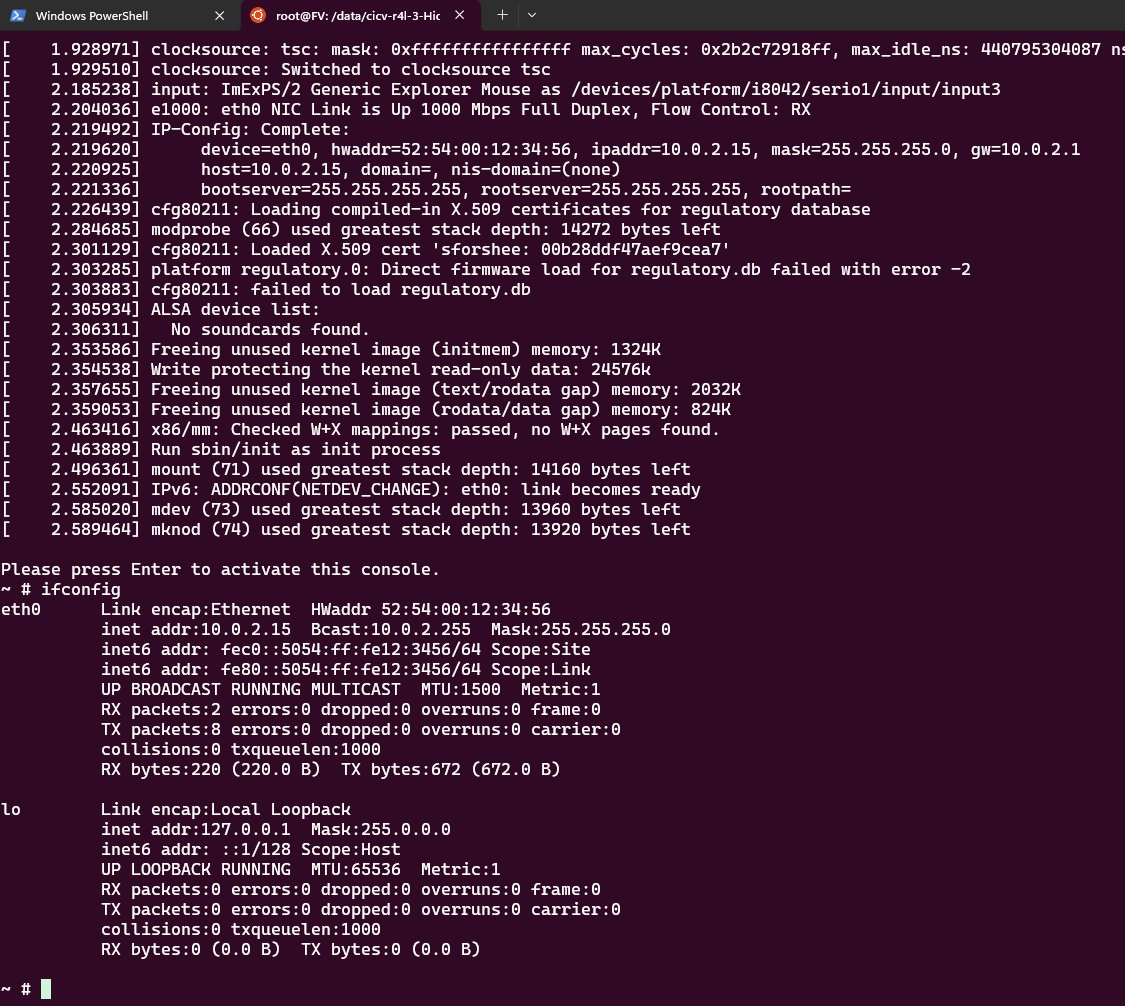
答：在 src\_e1000/build\_image.sh 里，把编译出来的.ko cp到 rootfs 下 打包进 rootfs\_image 里面，执行qemu时用 -initrd 参数加载文件系统，进入qemu就可以看到在根目录下



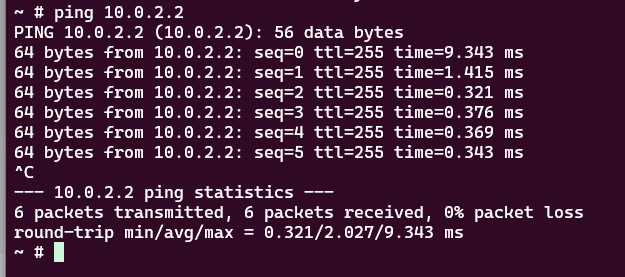


### 步骤

执行 src\_e1000/build\_image.sh



默认情况下就已经能正常连网



加载模块并配置网络：

insmod r4l\_e1000\_demo.ko

ip link set eth0 up

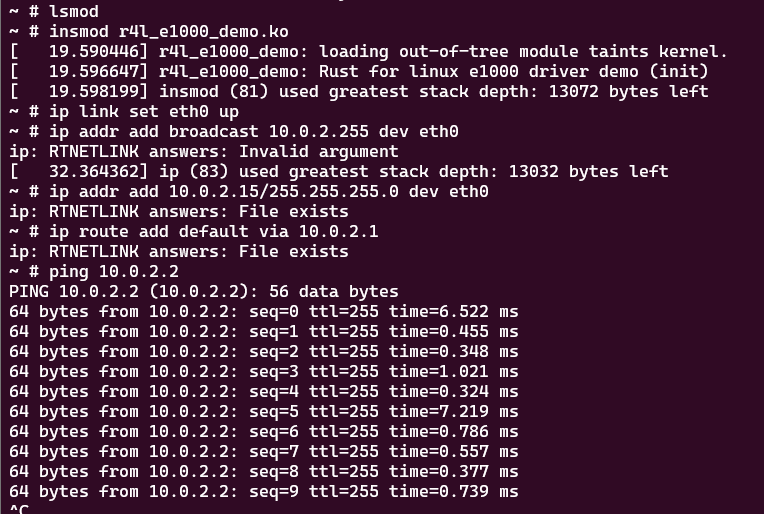
ip addr add broadcast 10.0.2.255 dev eth0

ip addr add 10.0.2.15/255.255.255.0 dev eth0

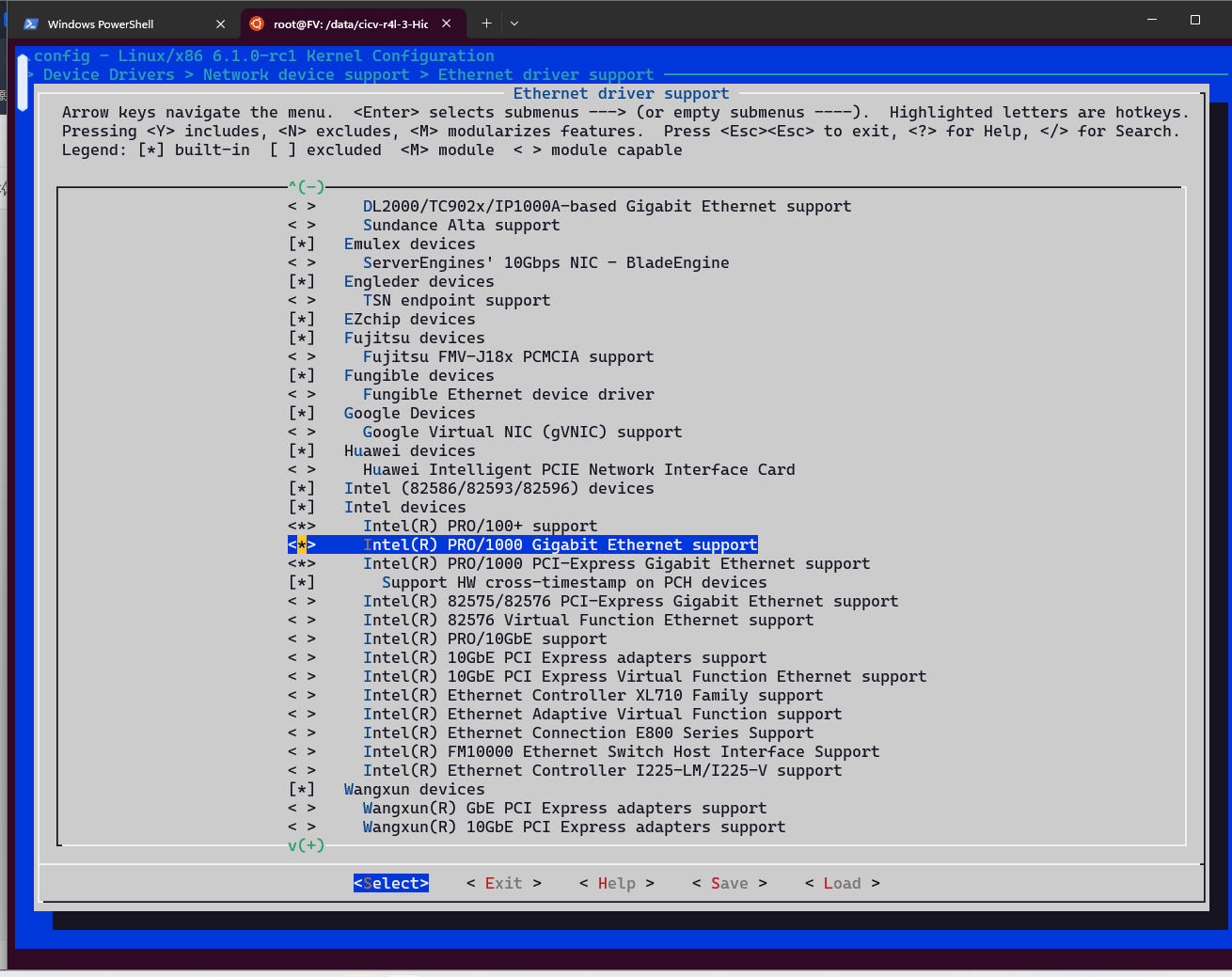
ip route add default via 10.0.2.1

ping 10.0.2.2

还是能连，并且 insmod 没有报错

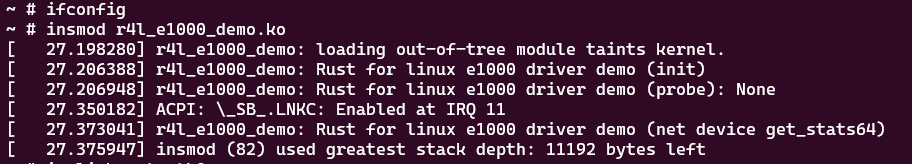


退回到 linux/ 下 make menuconfig 检查 没有被禁用

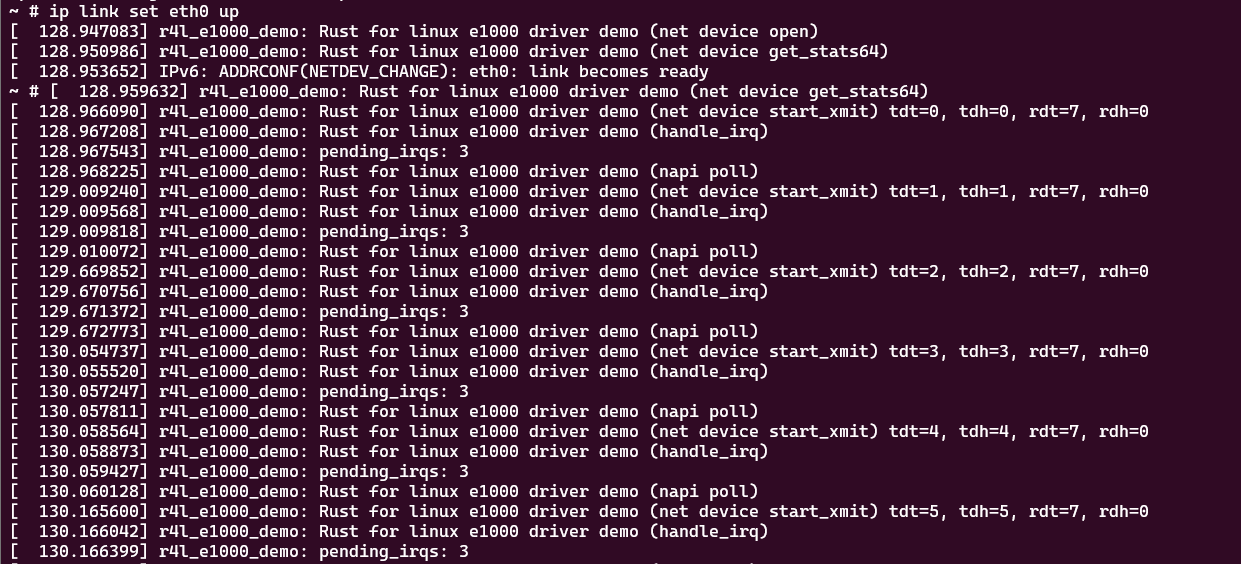


禁用掉保存退出，重新编译内核再启动qemu，看到没有自动配置网络

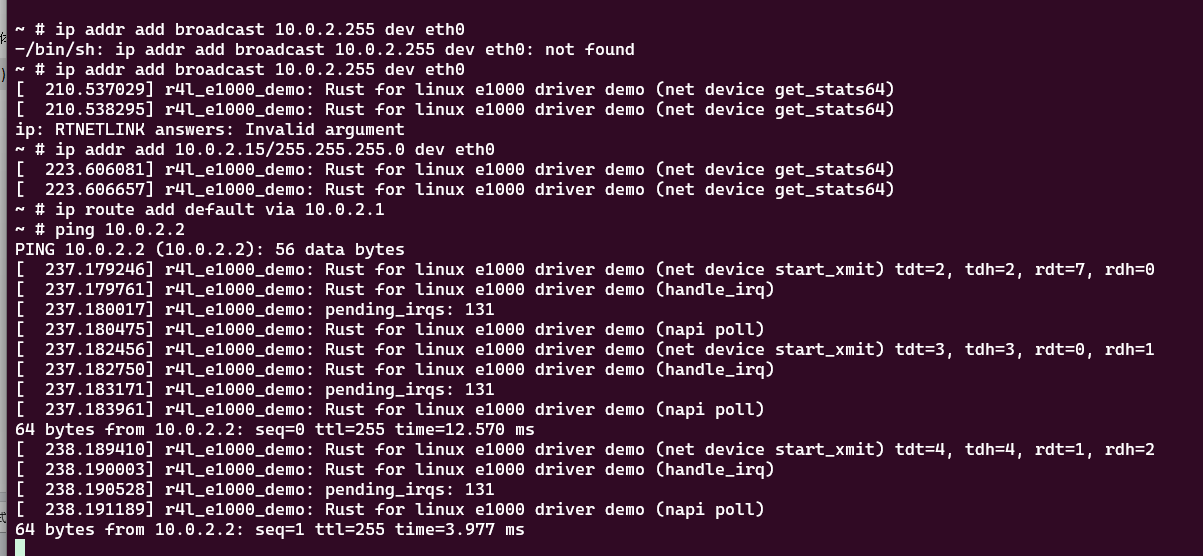
手动加载模块



配置网络



### 结果

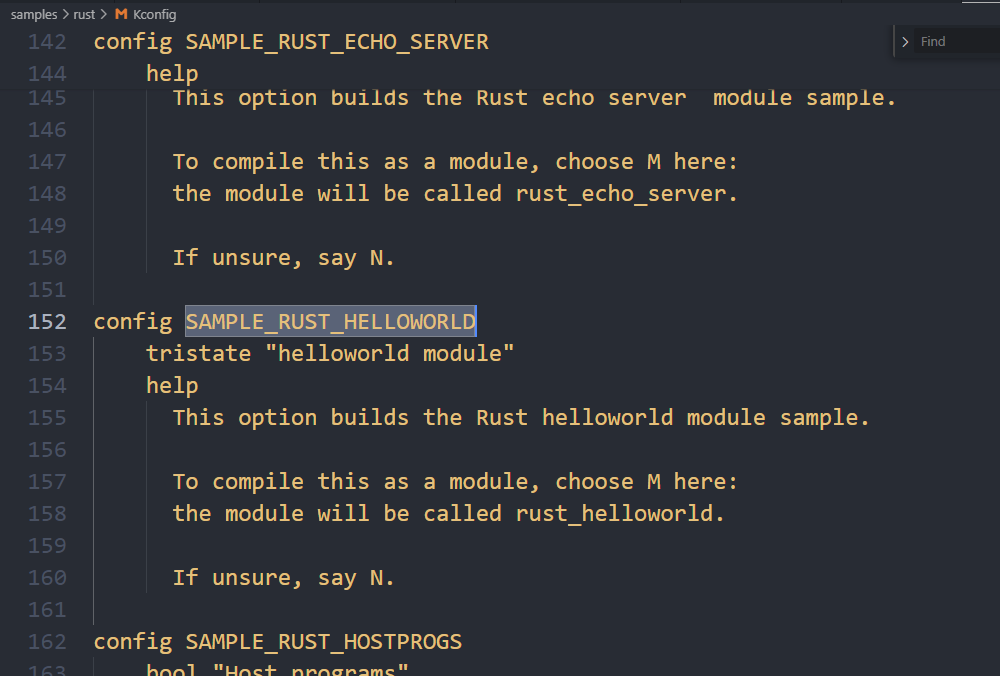


能连网，达到实验预期

## 作业3

### 步骤

按照实验文档在linux/samples/rust/ 下添加 rust\_helloworld.rs之后修改 Kconfig 和Makefile

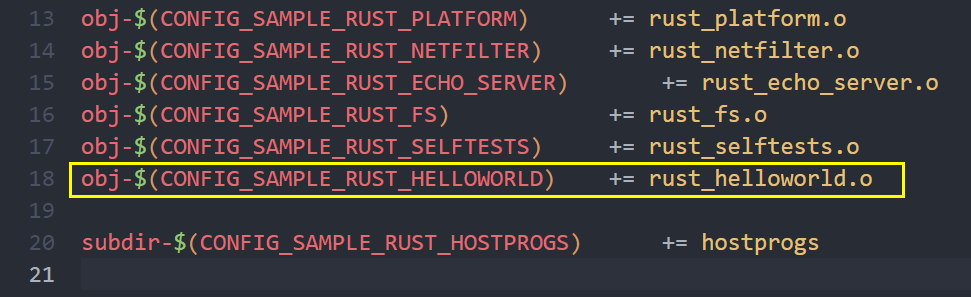


Kconfig里copy其他选项改个名字，实验文档里要求的是



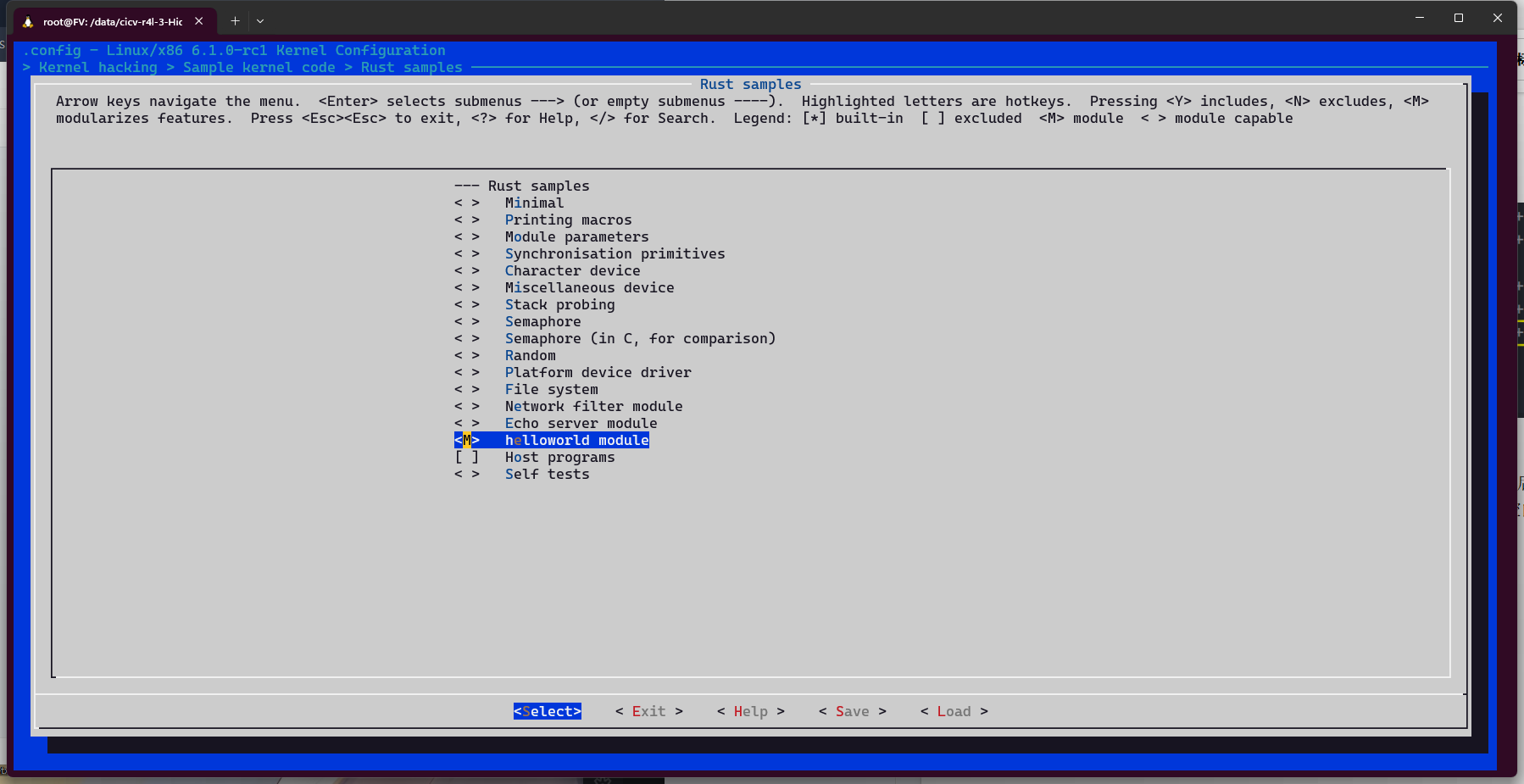
所以copy 带 To compile this as a module 的选项

Makefile里照抄别的，加一行

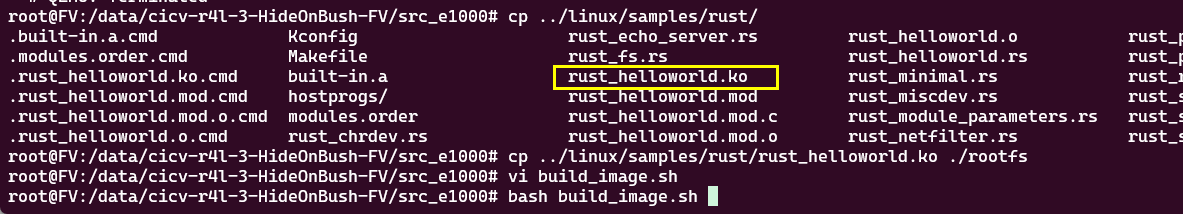


在 linux下执行 make LLVM=1 menuconfig

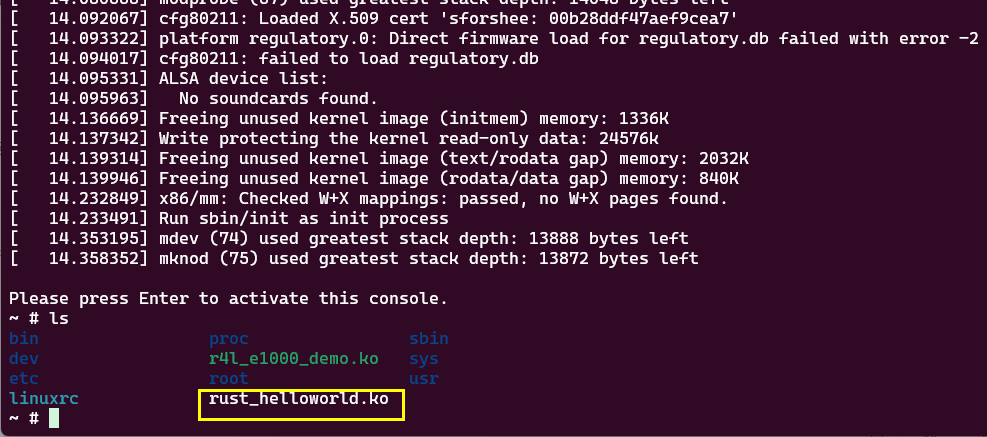
进入 kernel hacking，光标在 Sample kernel code 上停留之后按y或空格，前面[]里出现星号，回车进去之后才有各个sample，不然进去里面只有一个空的 Sample kernel code没有选项。回车进去之后选 Rust samples，选中刚才加的 helloworld，保存退出



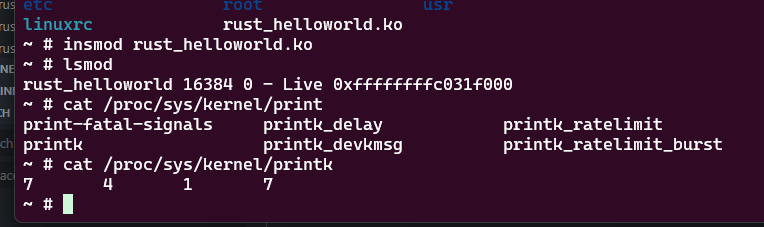
重新 make LLVM=1 -j$(nproc) 编译内核镜像，之后把生成的 samples/rust/rust\_helloworld.ko复制到 src\_e1000/rootfs/ 下面，重新跑 bash build\_image.sh



进入qemu发现里面有了这个模块

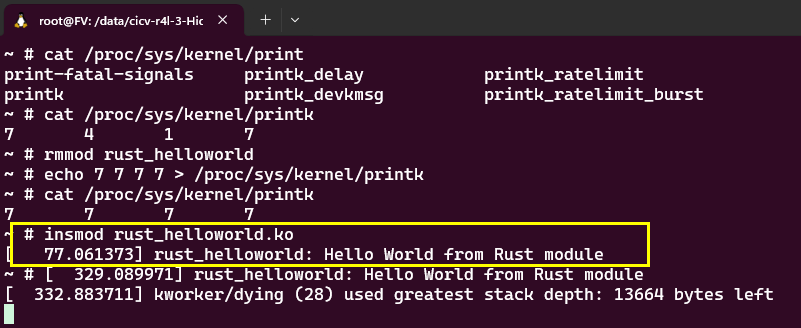


insmod 安装之后发现没有打印，检查打印级别



由于 rust\_helloworld.rs 里面用的函数是 pr\_info，猜测是主要控制台这列的 4 级别高于 info的6，而pr\_info 函数是在 init 函数里调用的，猜测只在加载时运行一次，所以先卸载模块，修改打印等级，再重新加载模块

### 结果



符合预期

### Kconfig

*# SPDX-License-Identifier: GPL-2.0*

menuconfig SAMPLES\_RUST

    bool "Rust samples"

    depends on RUST

    help

      You can build sample Rust kernel code here.

      If unsure, say N.

if SAMPLES\_RUST

config SAMPLE\_RUST\_MINIMAL

    tristate "Minimal"

    help

      This option builds the Rust minimal module sample.

      To compile this as a module, choose M here:

      the module will be called rust\_minimal.

      If unsure, say N.

config SAMPLE\_RUST\_PRINT

    tristate "Printing macros"

    help

      This option builds the Rust printing macros sample.

      To compile this as a module, choose M here:

      the module will be called rust\_print.

      If unsure, say N.

config SAMPLE\_RUST\_MODULE\_PARAMETERS

    tristate "Module parameters"

    help

      This option builds the Rust module parameters sample.

      To compile this as a module, choose M here:

      the module will be called rust\_module\_parameters.

      If unsure, say N.

config SAMPLE\_RUST\_SYNC

    tristate "Synchronisation primitives"

    help

      This option builds the Rust synchronisation primitives sample.

      To compile this as a module, choose M here:

      the module will be called rust\_sync.

      If unsure, say N.

config SAMPLE\_RUST\_CHRDEV

    tristate "Character device"

    help

      This option builds the Rust character device sample.

      To compile this as a module, choose M here:

      the module will be called rust\_chrdev.

      If unsure, say N.

config SAMPLE\_RUST\_MISCDEV

    tristate "Miscellaneous device"

    help

      This option builds the Rust miscellaneous device sample.

      To compile this as a module, choose M here:

      the module will be called rust\_miscdev.

      If unsure, say N.

config SAMPLE\_RUST\_STACK\_PROBING

    tristate "Stack probing"

    help

      This option builds the Rust stack probing sample.

      To compile this as a module, choose M here:

      the module will be called rust\_stack\_probing.

      If unsure, say N.

config SAMPLE\_RUST\_SEMAPHORE

    tristate "Semaphore"

    help

      This option builds the Rust semaphore sample.

      To compile this as a module, choose M here:

      the module will be called rust\_semaphore.

      If unsure, say N.

config SAMPLE\_RUST\_SEMAPHORE\_C

    tristate "Semaphore (in C, for comparison)"

    help

      This option builds the Rust semaphore sample (in C, for comparison).

      To compile this as a module, choose M here:

      the module will be called rust\_semaphore\_c.

      If unsure, say N.

config SAMPLE\_RUST\_RANDOM

    tristate "Random"

    help

      This option builds the Rust random sample.

      To compile this as a module, choose M here:

      the module will be called rust\_random.

      If unsure, say N.

config SAMPLE\_RUST\_PLATFORM

    tristate "Platform device driver"

    help

      This option builds the Rust platform device driver sample.

      To compile this as a module, choose M here:

      the module will be called rust\_platform.

config SAMPLE\_RUST\_FS

    tristate "File system"

    help

      This option builds the Rust file system sample.

      To compile this as a module, choose M here:

      the module will be called rust\_fs.

      If unsure, say N.

config SAMPLE\_RUST\_NETFILTER

    tristate "Network filter module"

    depends on NETFILTER

    help

      This option builds the Rust netfilter module sample.

      To compile this as a module, choose M here:

      the module will be called rust\_netfilter.

      If unsure, say N.

config SAMPLE\_RUST\_ECHO\_SERVER

    tristate "Echo server module"

    help

      This option builds the Rust echo server  module sample.

      To compile this as a module, choose M here:

      the module will be called rust\_echo\_server.

      If unsure, say N.

config SAMPLE\_RUST\_HELLOWORLD

    tristate "helloworld module"

    help

      This option builds the Rust helloworld module sample.

      To compile this as a module, choose M here:

      the module will be called rust\_helloworld.

      If unsure, say N.

config SAMPLE\_RUST\_HOSTPROGS

    bool "Host programs"

    help

      This option builds the Rust host program samples.

      If unsure, say N.

config SAMPLE\_RUST\_SELFTESTS

    tristate "Self tests"

    help

      This option builds the self test cases for Rust.

      If unsure, say N.

endif *# SAMPLES\_RUST*

### Makefile

*# SPDX-License-Identifier: GPL-2.0*

obj-$(CONFIG\_SAMPLE\_RUST\_MINIMAL)       += rust\_minimal.o

obj-$(CONFIG\_SAMPLE\_RUST\_PRINT)         += rust\_print.o

obj-$(CONFIG\_SAMPLE\_RUST\_MODULE\_PARAMETERS) += rust\_module\_parameters.o

obj-$(CONFIG\_SAMPLE\_RUST\_SYNC)          += rust\_sync.o

obj-$(CONFIG\_SAMPLE\_RUST\_CHRDEV)        += rust\_chrdev.o

obj-$(CONFIG\_SAMPLE\_RUST\_MISCDEV)       += rust\_miscdev.o

obj-$(CONFIG\_SAMPLE\_RUST\_STACK\_PROBING)     += rust\_stack\_probing.o

obj-$(CONFIG\_SAMPLE\_RUST\_SEMAPHORE)     += rust\_semaphore.o

obj-$(CONFIG\_SAMPLE\_RUST\_SEMAPHORE\_C)       += rust\_semaphore\_c.o

obj-$(CONFIG\_SAMPLE\_RUST\_RANDOM)        += rust\_random.o

obj-$(CONFIG\_SAMPLE\_RUST\_PLATFORM)      += rust\_platform.o

obj-$(CONFIG\_SAMPLE\_RUST\_NETFILTER)     += rust\_netfilter.o

obj-$(CONFIG\_SAMPLE\_RUST\_ECHO\_SERVER)       += rust\_echo\_server.o

obj-$(CONFIG\_SAMPLE\_RUST\_FS)            += rust\_fs.o

obj-$(CONFIG\_SAMPLE\_RUST\_SELFTESTS)     += rust\_selftests.o

obj-$(CONFIG\_SAMPLE\_RUST\_HELLOWORLD)    += rust\_helloworld.o

subdir-$(CONFIG\_SAMPLE\_RUST\_HOSTPROGS)      += hostprogs

## 作业5

### 问答题

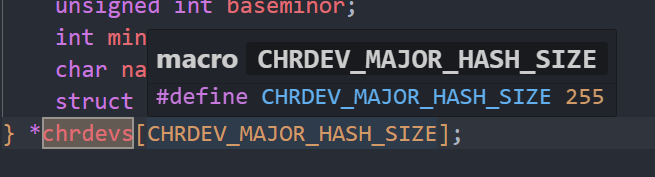
**Q：作业5中的字符设备/dev/cicv是怎么创建的？它的设备号是多少？它是如何与我们写的字符设备驱动关联上的？**

答：设备号是 248。在 build\_image.sh 里往 /etc/init.d/rcS 里写了 mknod命令创建了 /dev/cicv 设备，samples/rust/rust\_chrdev.rs 里面调用

chrdev\_reg.as\_mut().register::<RustFile>()?; 里面调用

alloc\_chrdev\_region 把模块绑定设备号，里面调用

find\_dynamic\_major

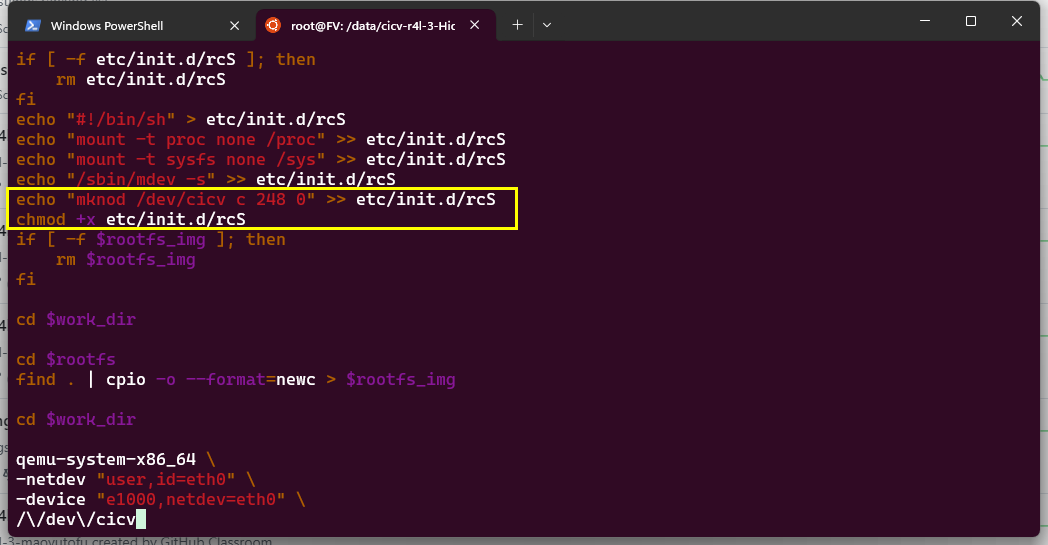


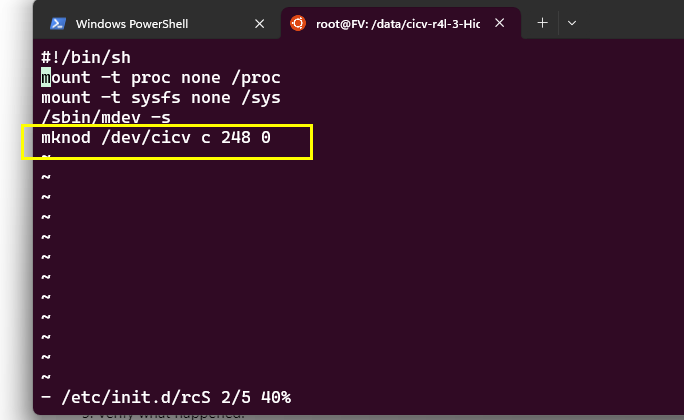
从255 个里，最后一个开始，逐个查找未被使用的设备号，返回给 chrdev



可以看到 255以下到248之间的设备号都已被其他驱动占用，所以分给 rust\_chrdev 的就是 248 了







/dev/cicv 与248号绑定，248 号分给了 rust\_chrdev，所以 rust\_chrdev 可以绑定 /dev/cicv，如果修改 build\_image 里的命令，/dev/cicv分配其他编号比如247，就会 insmod 也无法使用，因为rust\_chrdev 分到的设备号依然是248

### 步骤

改写 samples/rust/rust\_chrdev.rs 里的 write, read 函数

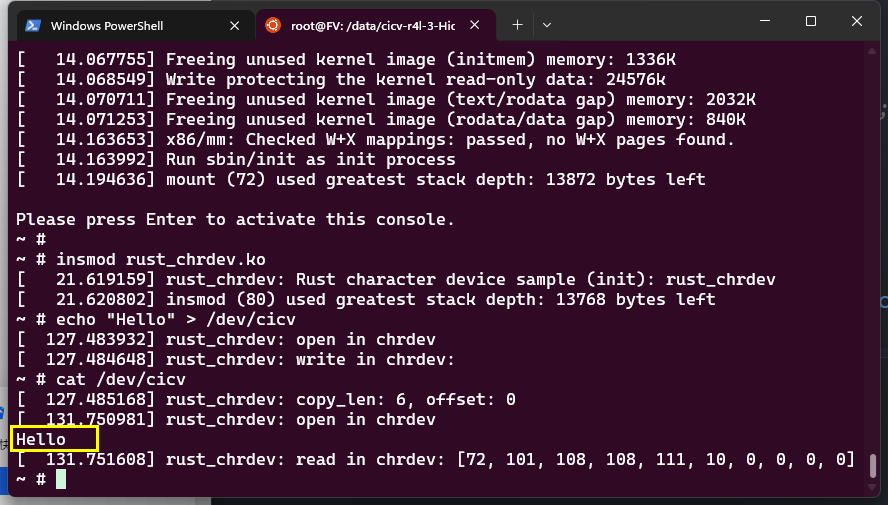
write时把buffer里的东西保存进静态数组，

read时把静态数组里的东西写进buffer



### 结果

可以正常存储和读出

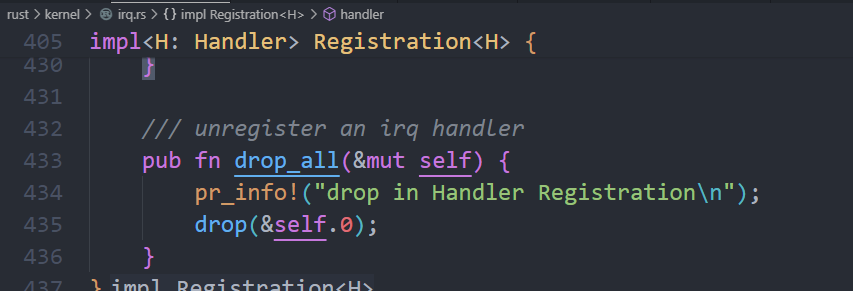


## 作业4

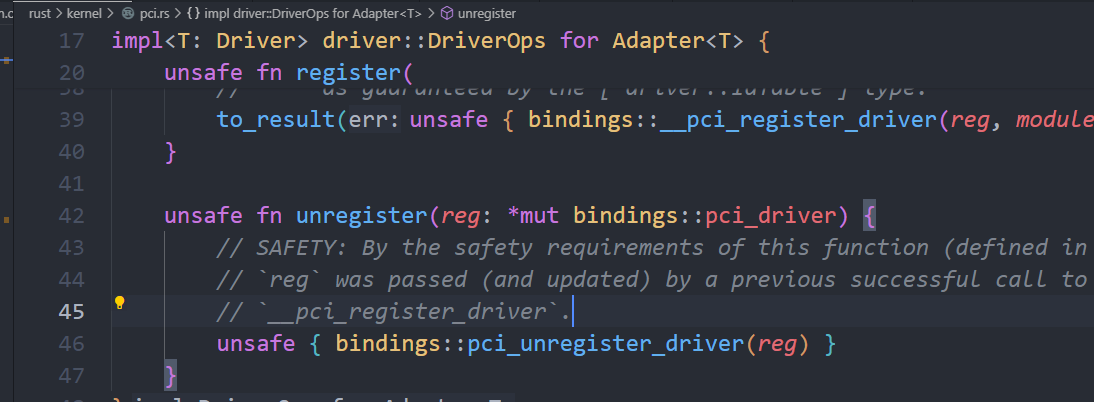
### 步骤

1. 梳理C代码，搞清楚各个结构体成员对应的具体内容、互相之间的关系、相关联的函数
2. 梳理rust代码，跟C和具体设备及函数对应起来
3. 补充缺失的函数，重点是 rust/kernel 里的不全，需要自己去 binding\_generate 下找C对应的函数并在kernel里补全 pci, net, irq 等需要用到的组件，并在demo里调用

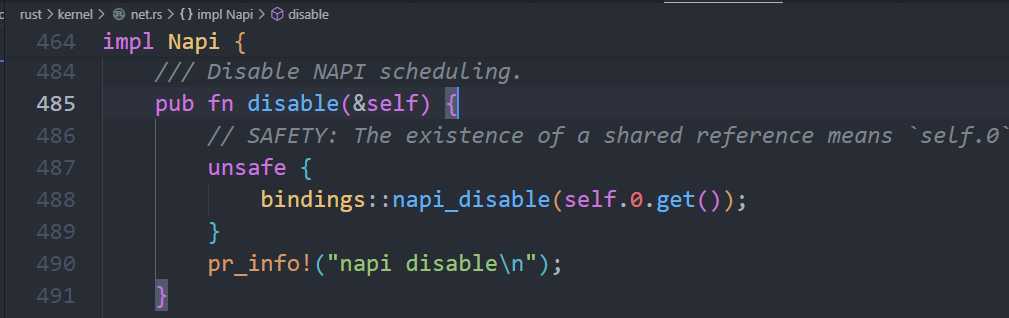
irq里改动：InternalRegistration 的drop已经实现了 free\_irq，并且是Registration的成员，所以只加一句话调用一下就可以



pci里加上unregister：

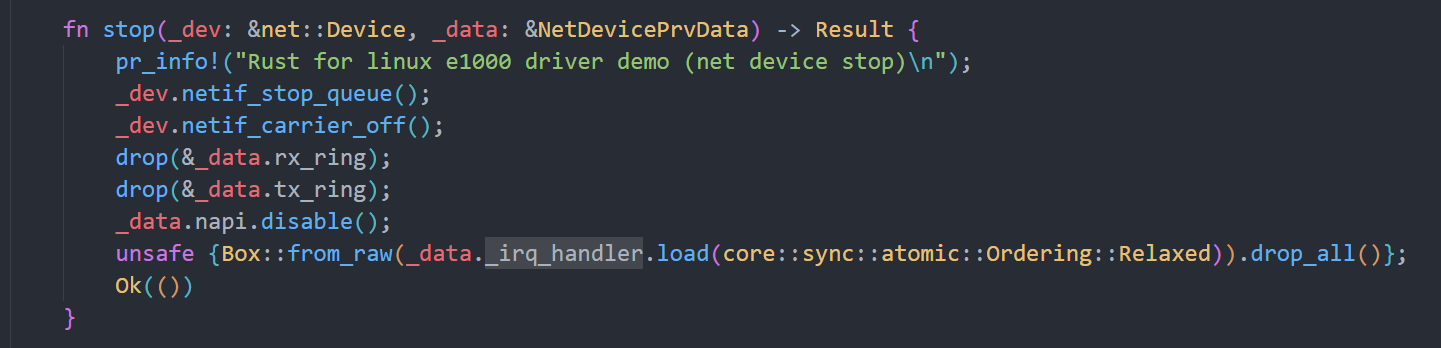


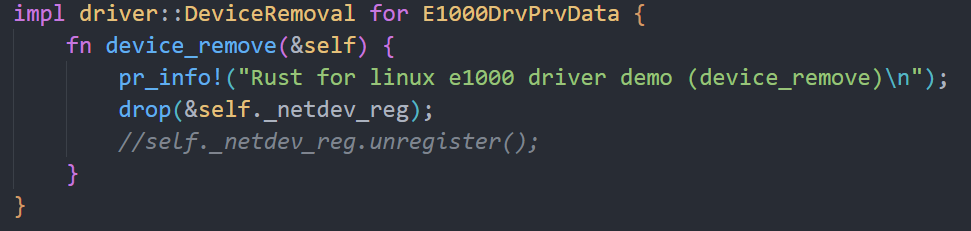
net加上disable\_napi

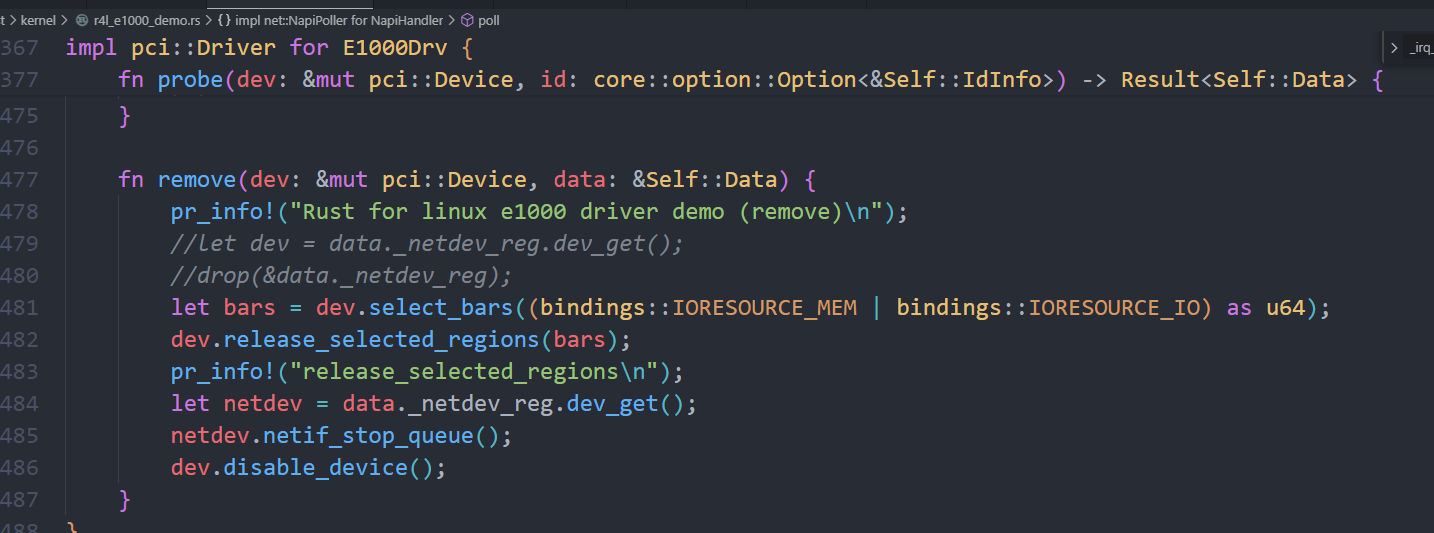


**PS：pci和net的具体改动是前几天的，记不全了=。=可能有遗漏**

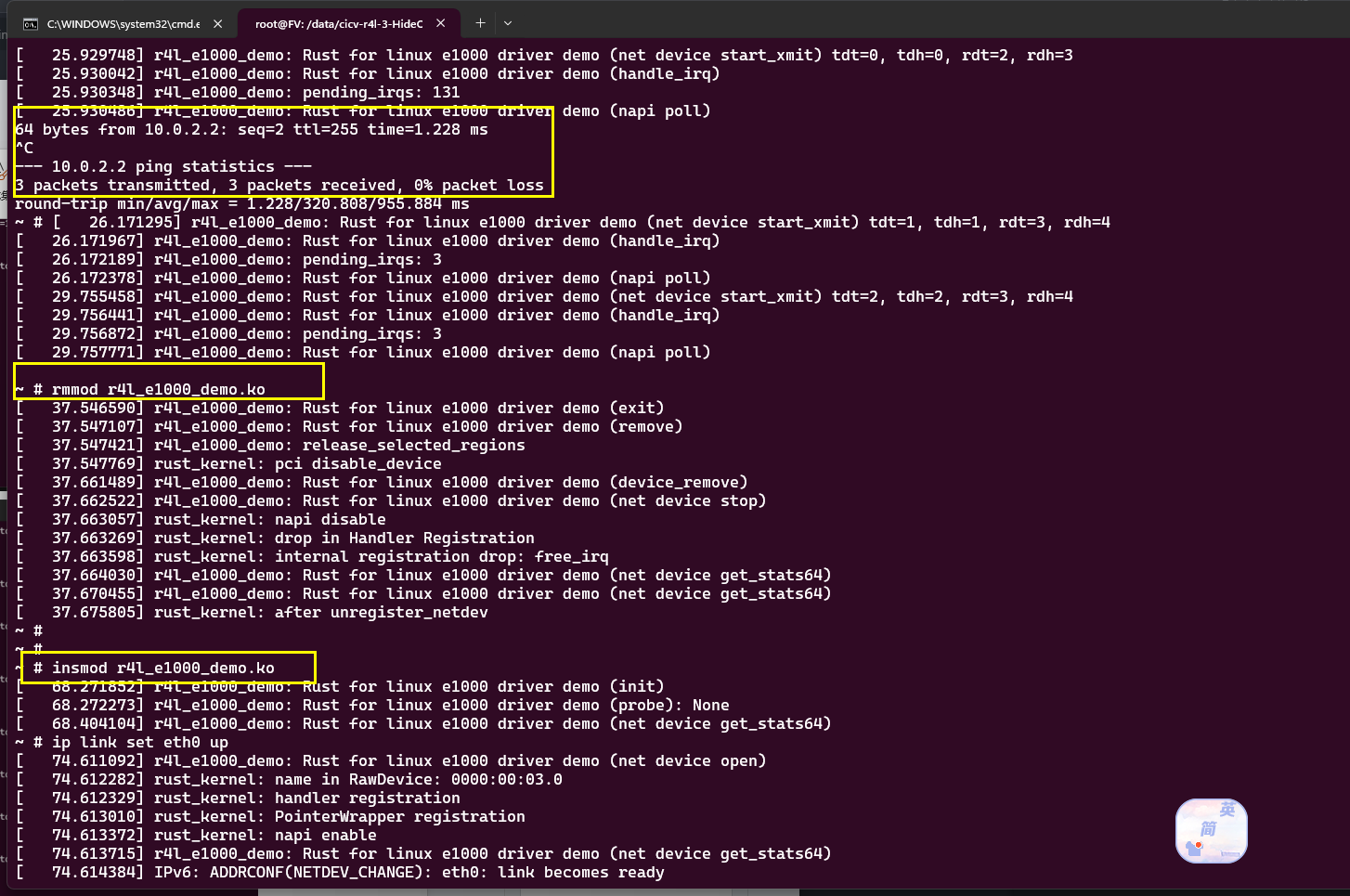
r4l\_e1000\_demo.rs 里改动：





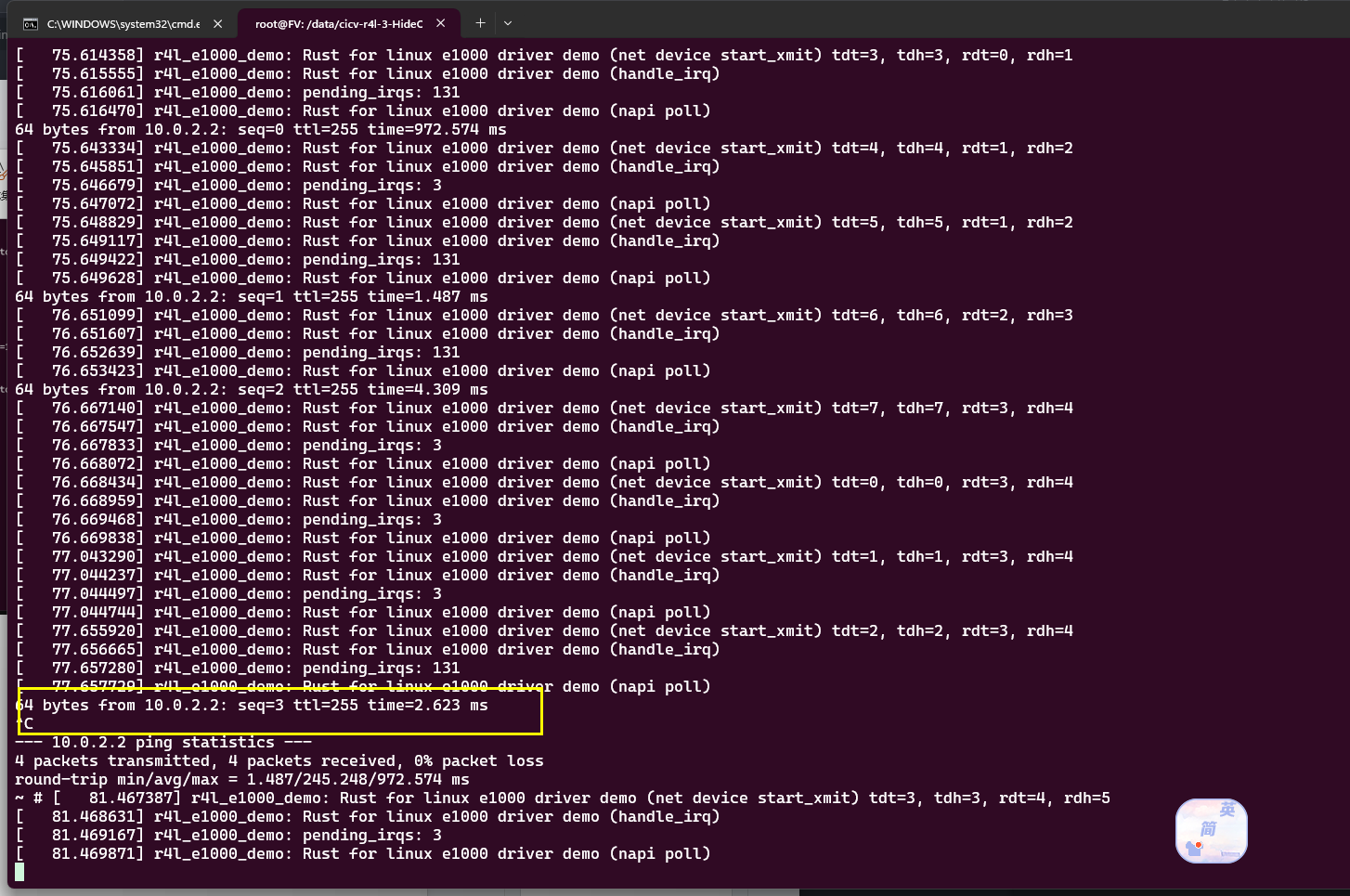


### 结果



insmod 并配置网络能ping通，之后rmmod 并再次insmod + 配置网络，

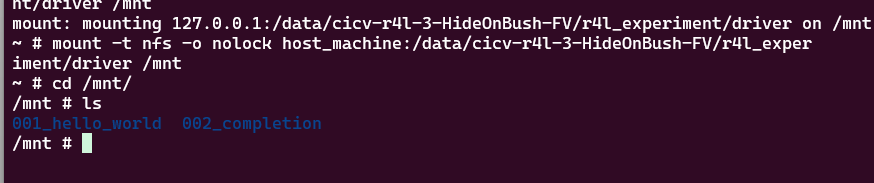
依旧能ping通



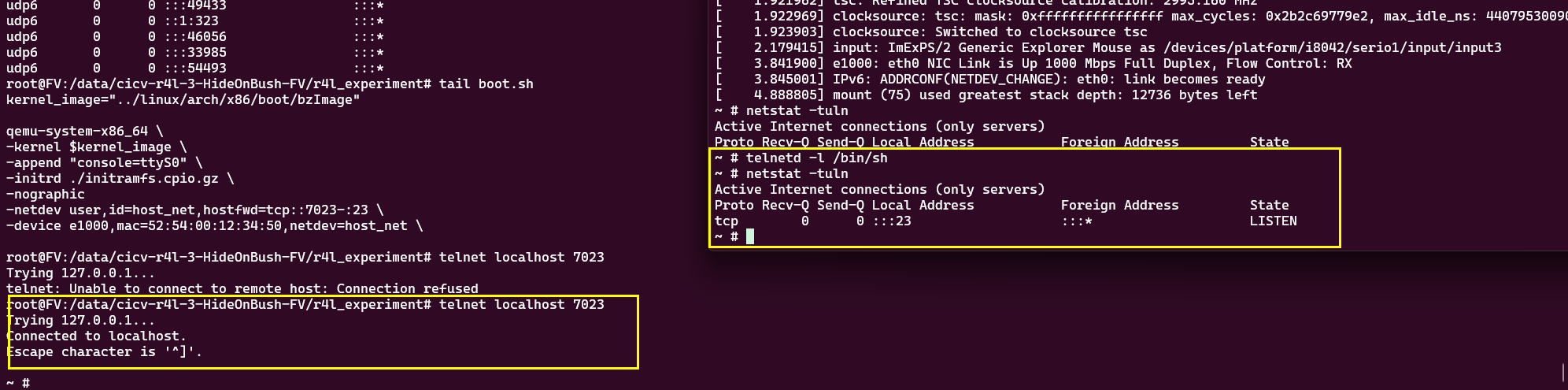
## 项目小试验

### 步骤

支持NFS：



支持telnet



启动监听，telnet 可以连接上

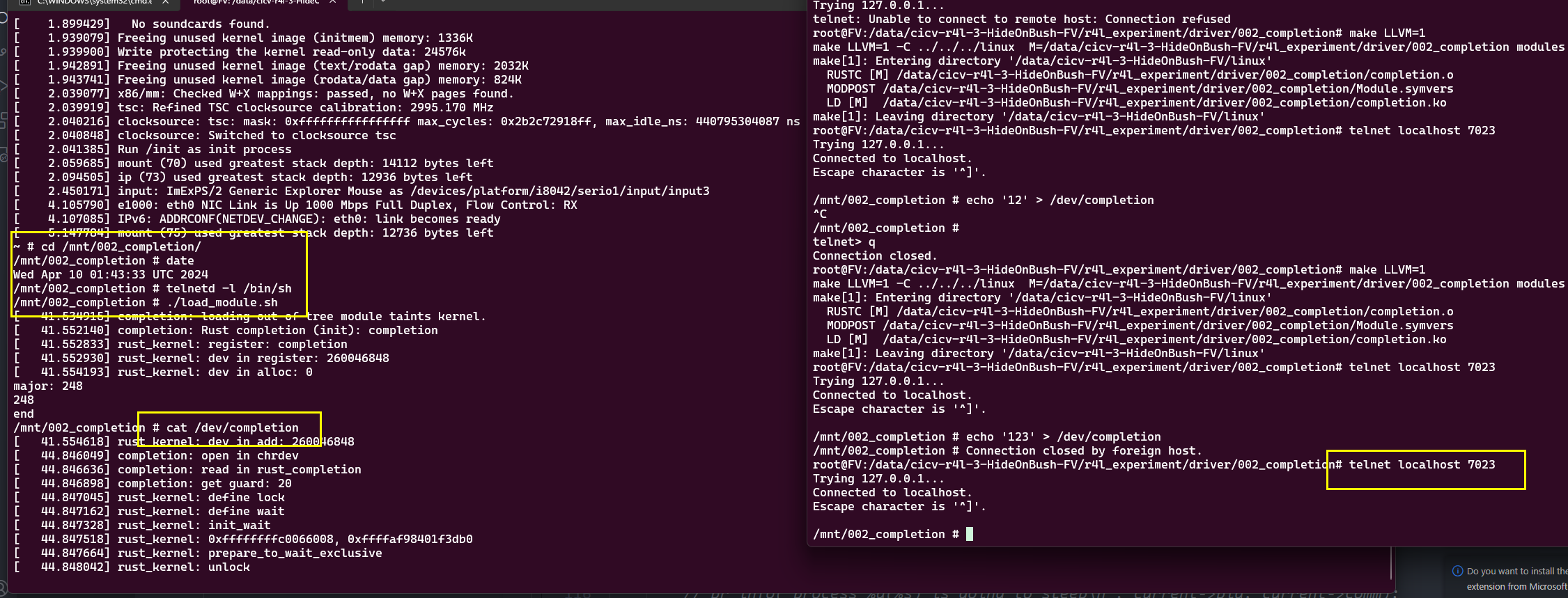
### completion

代码：照抄 samples/rust/rust\_sync.rs，使用静态Mutex + CondVar 即可=。= 一开始没看到静态的，自己写的静态在 open 里返回了，运行就在wait里一直报对空指针解引用

结果：

qemu端init里自动挂载，进入挂载目录之后启动telnetd并加载模块，

cat /dev/completion 进入等待



主机端 telnet 进入qemu的系统，echo xxx > /dev/completion condvar唤醒qemu端

