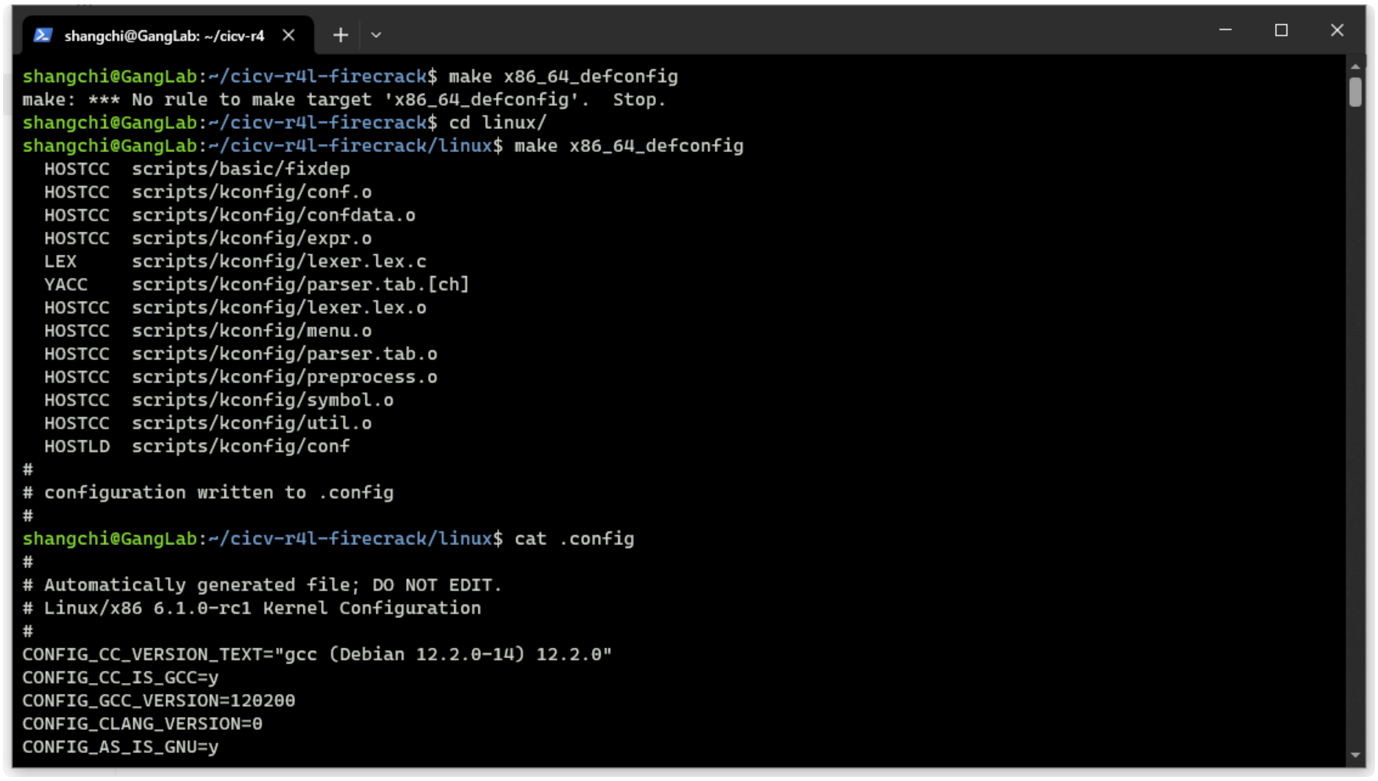
作业

作业一

1. `make x86_64_defconfig`

生成一个x86_64架构的默认配置文件，其包含了相关的内核配置选项的默认值。



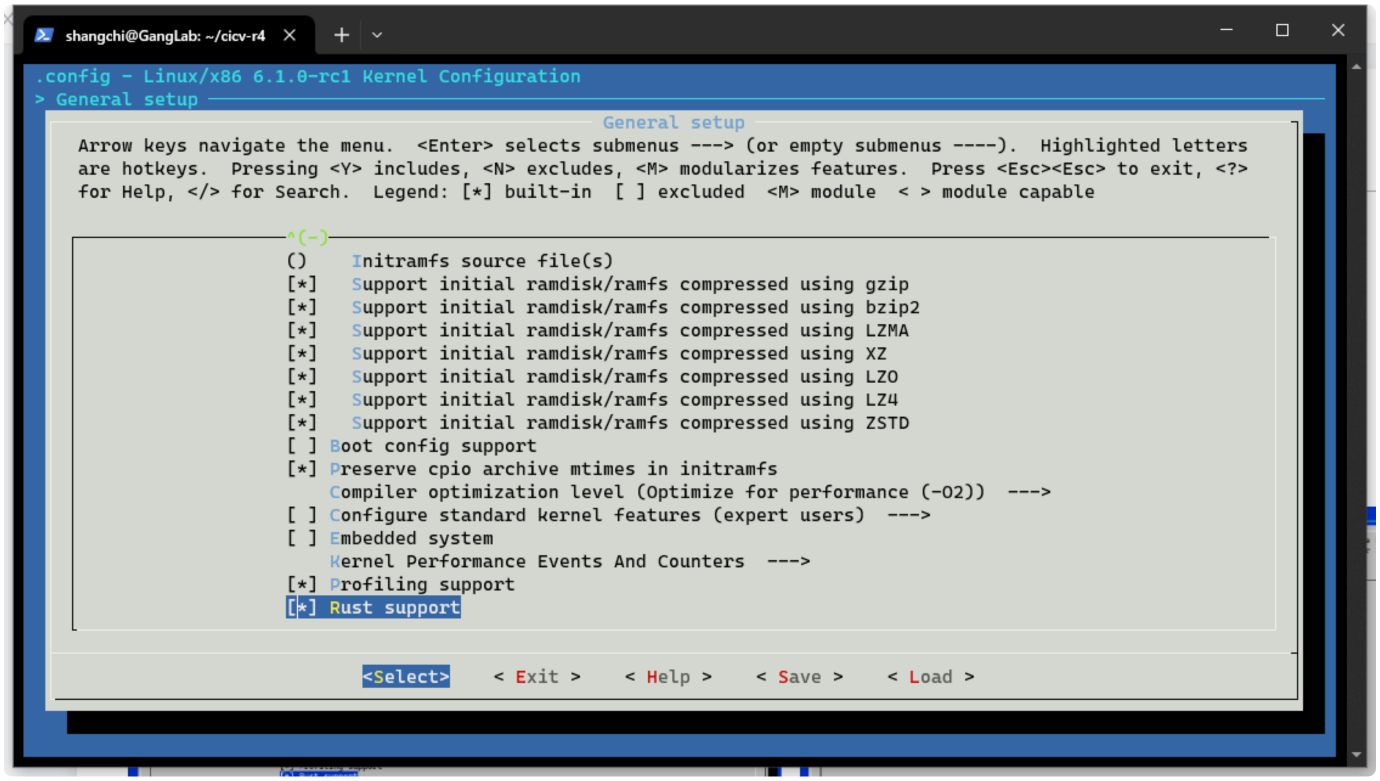
shangchi@GangLab: ~/cicv-r4

```
shangchi@GangLab:~/cicv-r4l-firecrack$ make x86_64_defconfig
make: *** No rule to make target 'x86_64_defconfig'.  Stop.
shangchi@GangLab:~/cicv-r4l-firecrack$ cd linux/
shangchi@GangLab:~/cicv-r4l-firecrack/linux$ make x86_64_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/menu.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
shangchi@GangLab:~/cicv-r4l-firecrack/linux$ cat .config
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86 6.1.0-rc1 Kernel Configuration
#
CONFIG_CC_VERSION_TEXT="gcc (Debian 12.2.0-14) 12.2.0"
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=120200
CONFIG_CLANG_VERSION=0
CONFIG_AS_IS_GNU=y
```

2. `makeLLVM=1 menuconfig`

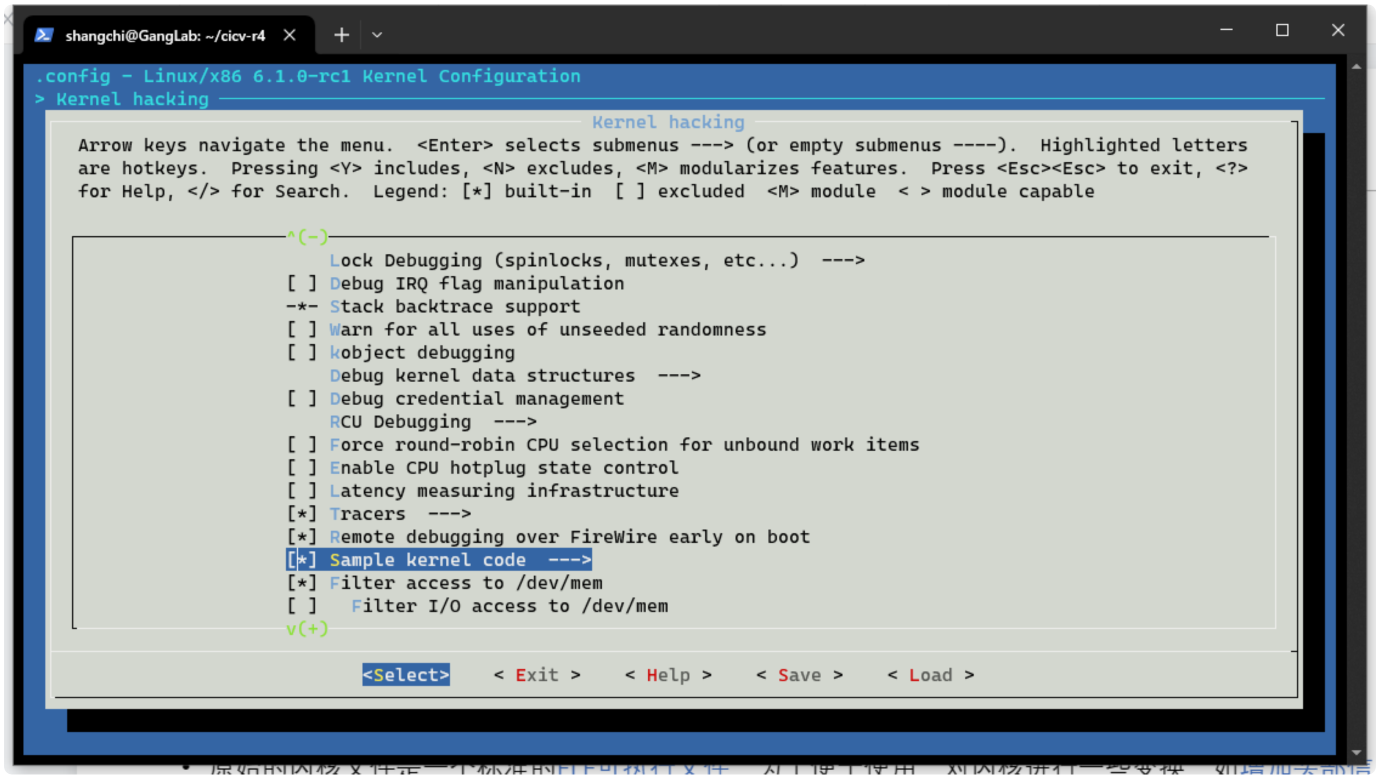
- `LLVM=1` 表示使用LLVM编译器来构建内核
- `menuconfig` 启动Linux内核配置工具的文本菜单界面

选中 `Rust support`



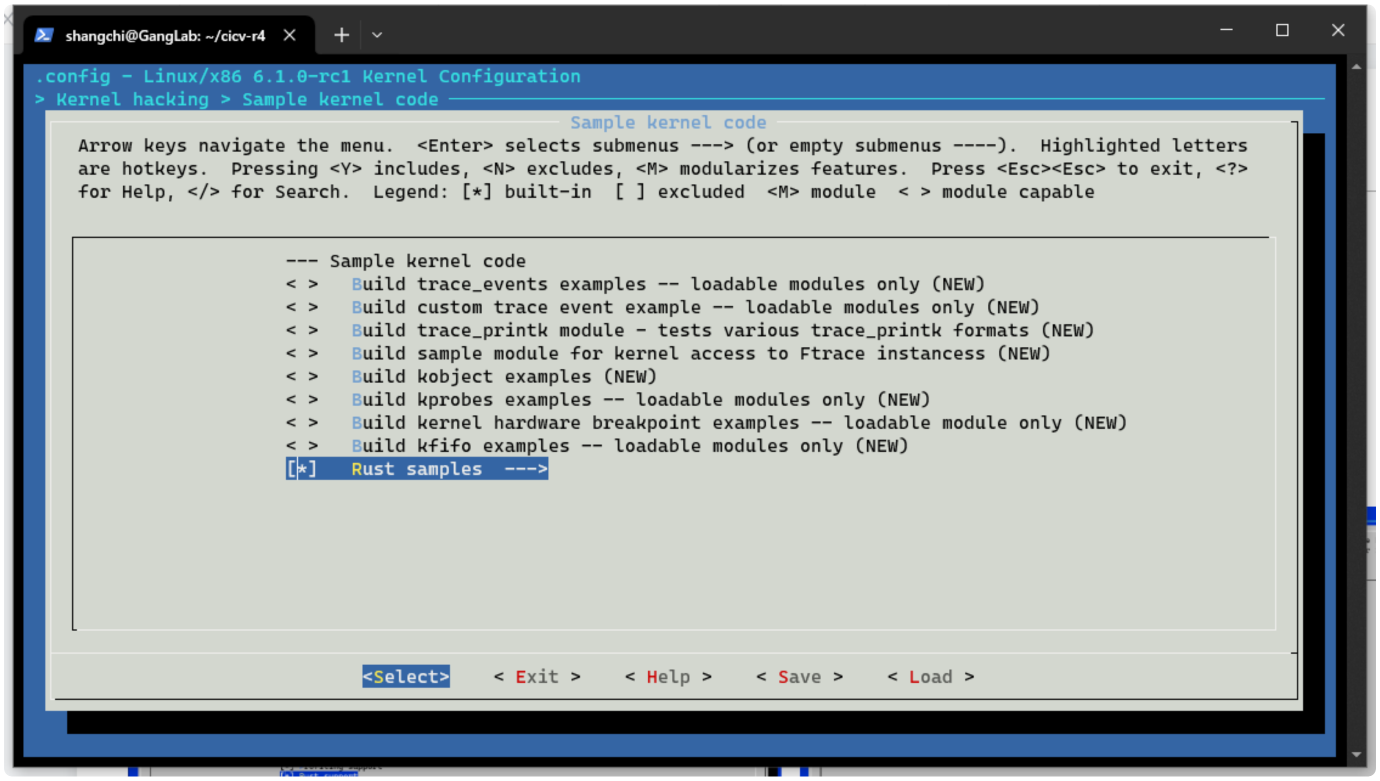
```
.config - Linux/x86 6.1.0-rc1 Kernel Configuration
> General setup ─────────────────────────────────────────────────────────────
                              General setup
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters
  are hotkeys.  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

                    ^(-)
                    ()    Initramfs source file(s)
                    [*]   Support initial ramdisk/ramfs compressed using gzip
                    [*]   Support initial ramdisk/ramfs compressed using bzip2
                    [*]   Support initial ramdisk/ramfs compressed using LZMA
                    [*]   Support initial ramdisk/ramfs compressed using XZ
                    [*]   Support initial ramdisk/ramfs compressed using LZO
                    [*]   Support initial ramdisk/ramfs compressed using LZ4
                    [*]   Support initial ramdisk/ramfs compressed using ZSTD
                    [ ] Boot config support
                    [*] Preserve cpio archive mtimes in initramfs
                        Compiler optimization level (Optimize for performance (-O2))  --->
                    [ ] Configure standard kernel features (expert users)  --->
                    [ ] Embedded system
                        Kernel Performance Events And Counters  --->
                    [*] Profiling support
                    [*] Rust support

                  <Select>    < Exit >    < Help >    < Save >    < Load >
```

选中 `Sample kernel code`



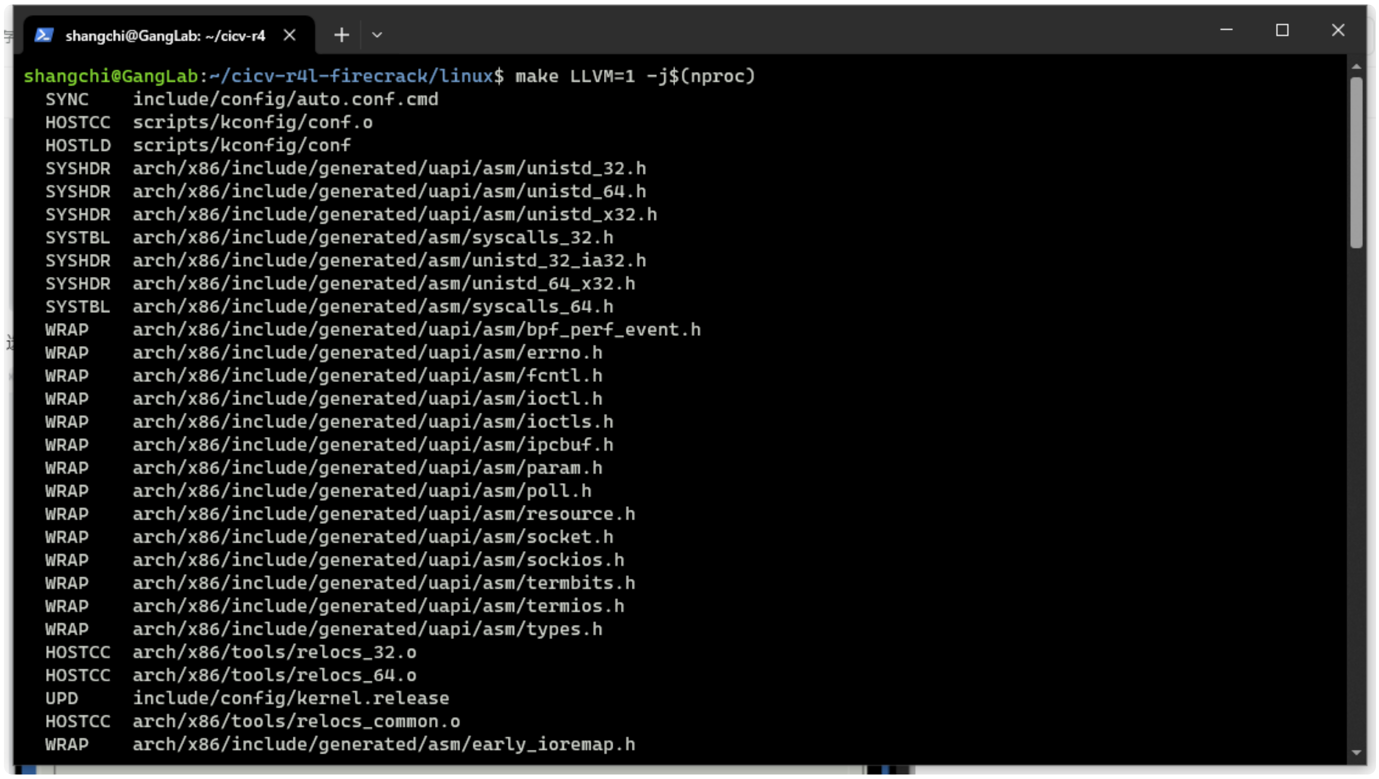
```
.config - Linux/x86 6.1.0-rc1 Kernel Configuration
> Kernel hacking ────────────────────────────────────────────────────────────
                              Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters
  are hotkeys.  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

                    ^(-)
                        Lock Debugging (spinlocks, mutexes, etc...)  --->
                    [ ] Debug IRQ flag manipulation
                    -*- Stack backtrace support
                    [ ] Warn for all uses of unseeded randomness
                    [ ] kobject debugging
                        Debug kernel data structures  --->
                    [ ] Debug credential management
                        RCU Debugging  --->
                    [ ] Force round-robin CPU selection for unbound work items
                    [ ] Enable CPU hotplug state control
                    [ ] Latency measuring infrastructure
                    [*] Tracers  --->
                    [*] Remote debugging over FireWire early on boot
                    [*] Sample kernel code  --->
                    [*] Filter access to /dev/mem
                    [ ]   Filter I/O access to /dev/mem
                    v(+)

                  <Select>    < Exit >    < Help >    < Save >    < Load >
```

选中 `Rust samples`



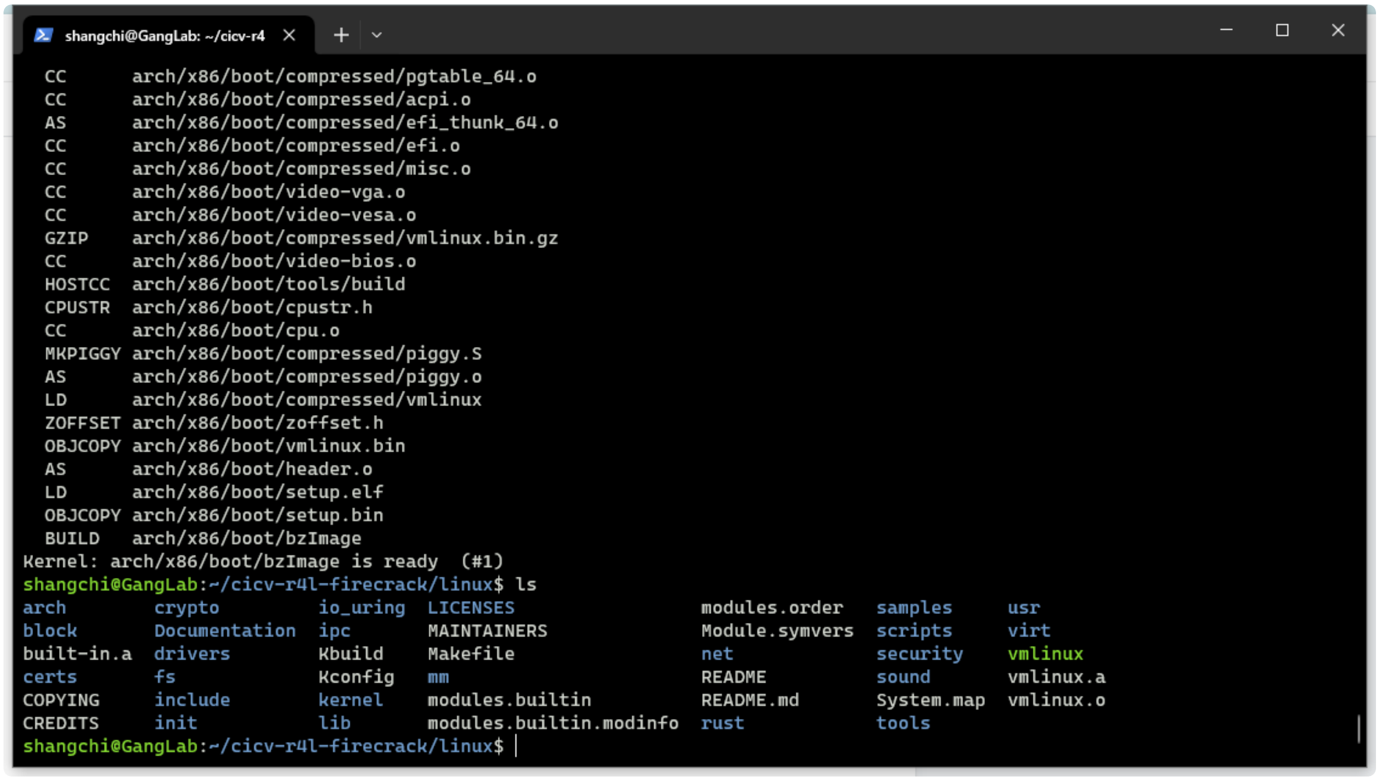
```
.config - Linux/x86 6.1.0-rc1 Kernel Configuration
> Kernel hacking > Sample kernel code ──────────────────────────────────────
                          Sample kernel code
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters
  are hotkeys.  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

                   --- Sample kernel code
                   < >   Build trace_events examples -- loadable modules only (NEW)
                   < >   Build custom trace event example -- loadable modules only (NEW)
                   < >   Build trace_printk module - tests various trace_printk formats (NEW)
                   < >   Build sample module for kernel access to Ftrace instances (NEW)
                   < >   Build kobject examples (NEW)
                   < >   Build kprobes examples -- loadable modules only (NEW)
                   < >   Build kernel hardware breakpoint examples -- loadable module only (NEW)
                   < >   Build kfifo examples -- loadable modules only (NEW)
                   [*]   Rust samples  --->

                     <Select>    < Exit >    < Help >    < Save >    < Load >
```

3. `make LLVM=1 -j$(proc)` 开始编译内核



```
shangchi@GangLab:~/cicv-r4l-firecrack/linux$ make LLVM=1 -j$(nproc)
  SYNC    include/config/auto.conf.cmd
  HOSTCC  scripts/kconfig/conf.o
  HOSTLD  scripts/kconfig/conf
  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
  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
  WRAP    arch/x86/include/generated/uapi/asm/bpf_perf_event.h
  WRAP    arch/x86/include/generated/uapi/asm/errno.h
  WRAP    arch/x86/include/generated/uapi/asm/fcntl.h
  WRAP    arch/x86/include/generated/uapi/asm/ioctl.h
  WRAP    arch/x86/include/generated/uapi/asm/ioctls.h
  WRAP    arch/x86/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/x86/include/generated/uapi/asm/param.h
  WRAP    arch/x86/include/generated/uapi/asm/poll.h
  WRAP    arch/x86/include/generated/uapi/asm/resource.h
  WRAP    arch/x86/include/generated/uapi/asm/socket.h
  WRAP    arch/x86/include/generated/uapi/asm/sockios.h
  WRAP    arch/x86/include/generated/uapi/asm/termbits.h
  WRAP    arch/x86/include/generated/uapi/asm/termios.h
  WRAP    arch/x86/include/generated/uapi/asm/types.h
  HOSTCC  arch/x86/tools/relocs_32.o
  HOSTCC  arch/x86/tools/relocs_64.o
  UPD     include/config/kernel.release
  HOSTCC  arch/x86/tools/relocs_common.o
  WRAP    arch/x86/include/generated/asm/early_ioremap.h
```

编译完成后会在linux目录中生成 `vmlinux` 文件

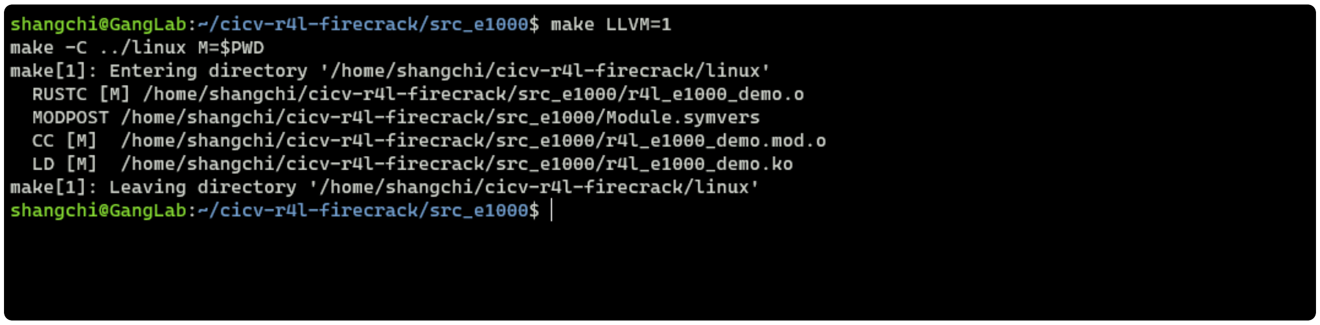


```
shangchi@GangLab: ~/cicv-r4
  CC      arch/x86/boot/compressed/pgtable_64.o
  CC      arch/x86/boot/compressed/acpi.o
  AS      arch/x86/boot/compressed/efi_thunk_64.o
  CC      arch/x86/boot/compressed/efi.o
  CC      arch/x86/boot/compressed/misc.o
  CC      arch/x86/boot/video-vga.o
  CC      arch/x86/boot/video-vesa.o
  GZIP    arch/x86/boot/compressed/vmlinux.bin.gz
  CC      arch/x86/boot/video-bios.o
  HOSTCC  arch/x86/boot/tools/build
  CPUSTR  arch/x86/boot/cpustr.h
  CC      arch/x86/boot/cpu.o
  MKPIGGY arch/x86/boot/compressed/piggy.S
  AS      arch/x86/boot/compressed/piggy.o
  LD      arch/x86/boot/compressed/vmlinux
  ZOFFSET arch/x86/boot/zoffset.h
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)
shangchi@GangLab:~/cicv-r4l-firecrack/linux$ ls
arch        crypto         io_uring  LICENSES                 modules.order   samples     usr
block       Documentation  ipc       MAINTAINERS              Module.symvers  scripts     virt
built-in.a  drivers        Kbuild    Makefile                 net             security    vmlinux
certs       fs             Kconfig   mm                       README          sound       vmlinux.a
COPYING     include        kernel    modules.builtin          README.md       System.map  vmlinux.o
CREDITS     init           lib       modules.builtin.modinfo  rust            tools
shangchi@GangLab:~/cicv-r4l-firecrack/linux$ |
```

作业二

- 构建网卡模块

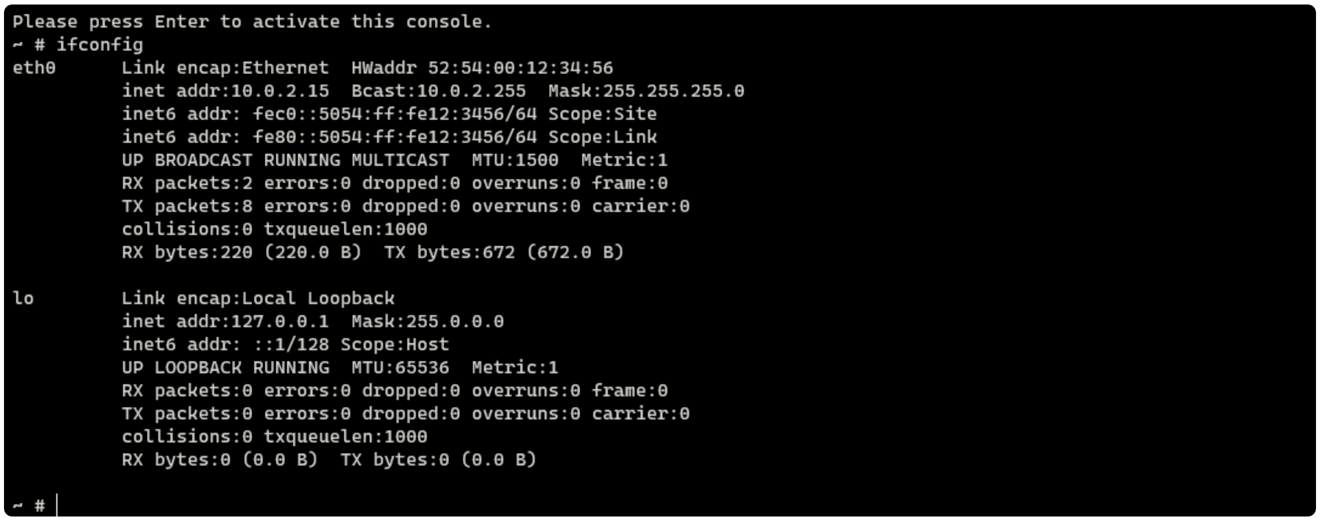
  在 `src_e1000` 文件夹中执行 `make LLVM=1` ，构建一个网卡驱动模块(.ko文件)

```
shangchi@GangLab:~/cicv-r4l-firecrack/src_e1000$ make LLVM=1
make -C ../linux M=$PWD
make[1]: Entering directory '/home/shangchi/cicv-r4l-firecrack/linux'
  RUSTC [M] /home/shangchi/cicv-r4l-firecrack/src_e1000/r4l_e1000_demo.o
  MODPOST /home/shangchi/cicv-r4l-firecrack/src_e1000/Module.symvers
  CC [M]  /home/shangchi/cicv-r4l-firecrack/src_e1000/r4l_e1000_demo.mod.o
  LD [M]  /home/shangchi/cicv-r4l-firecrack/src_e1000/r4l_e1000_demo.ko
make[1]: Leaving directory '/home/shangchi/cicv-r4l-firecrack/linux'
shangchi@GangLab:~/cicv-r4l-firecrack/src_e1000$ |
```

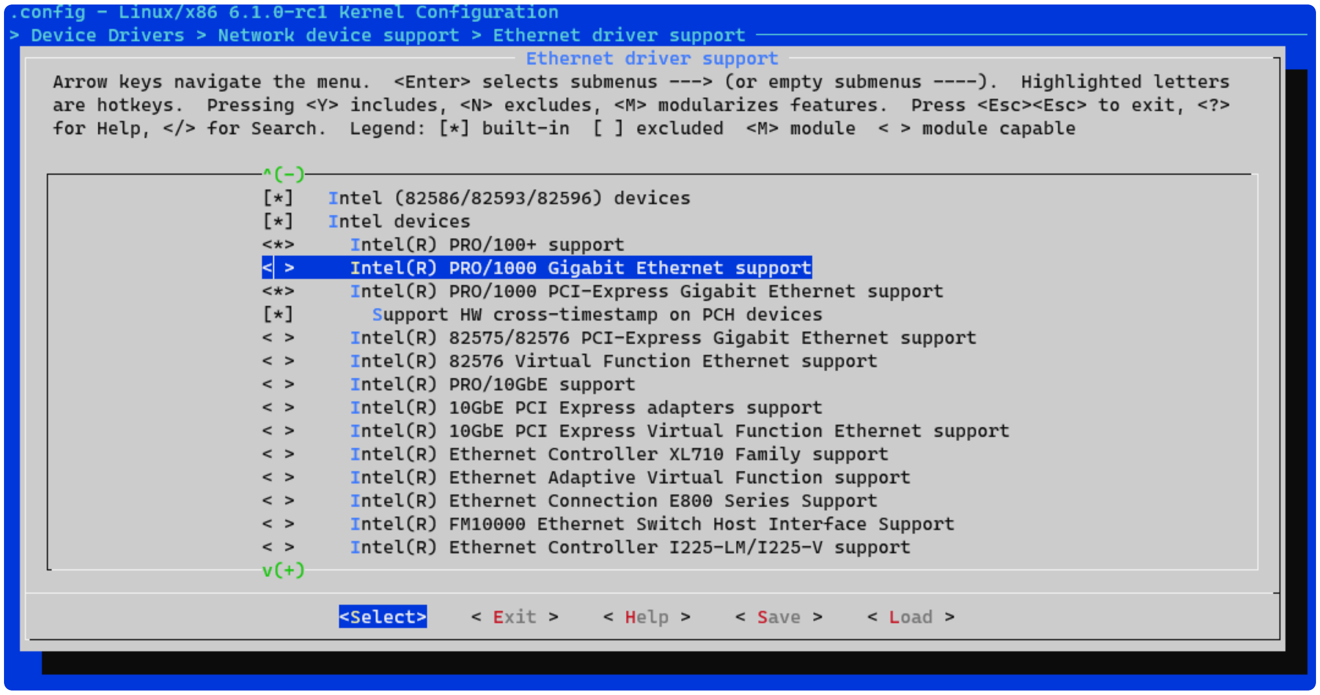
- 使用 `./build_image.sh` 脚本运行qemu

  `ifconfig` 查看网卡，这里启动的网卡驱动是Linux 内核本身具有的



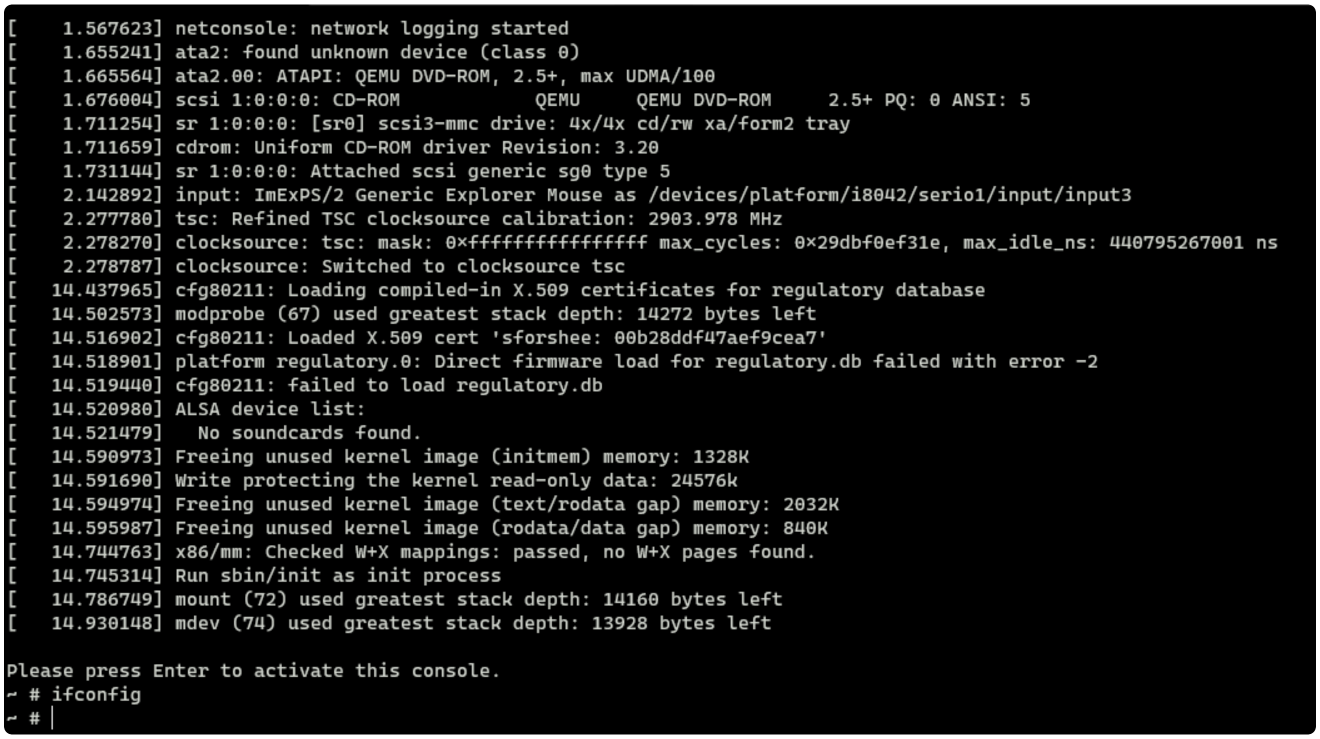
```
Please press Enter to activate this console.
~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fec0::5054:ff:fe12:3456/64 Scope:Site
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:220 (220.0 B)  TX bytes:672 (672.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  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)

~ #
```

- 禁用Linux默认的网卡驱动
  `Device Divers->Network device support->Ethernet driver support -> Intel (R) PRO/1000 Gigabit Ethernet support`



```
.config - Linux/x86 6.1.0-rc1 Kernel Configuration
> Device Drivers > Network device support > Ethernet driver support ─────────────
                              Ethernet driver support
  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

                  ^(-)
                  [*]   Intel (82586/82593/82596) devices
                  [*]   Intel devices
                  <*>     Intel(R) PRO/100+ support
                  < >     Intel(R) PRO/1000 Gigabit Ethernet support
                  <*>     Intel(R) PRO/1000 PCI-Express Gigabit Ethernet support
                  [*]       Support HW cross-timestamp on PCH devices
                  < >     Intel(R) 82575/82576 PCI-Express Gigabit Ethernet support
                  < >     Intel(R) 82576 Virtual Function Ethernet support
                  < >     Intel(R) PRO/10GbE support
                  < >     Intel(R) 10GbE PCI Express adapters support
                  < >     Intel(R) 10GbE PCI Express Virtual Function Ethernet support
                  < >     Intel(R) Ethernet Controller XL710 Family support
                  < >     Intel(R) Ethernet Adaptive Virtual Function support
                  < >     Intel(R) Ethernet Connection E800 Series Support
                  < >     Intel(R) FM10000 Ethernet Switch Host Interface Support
                  < >     Intel(R) Ethernet Controller I225-LM/I225-V support
                  v(+)

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

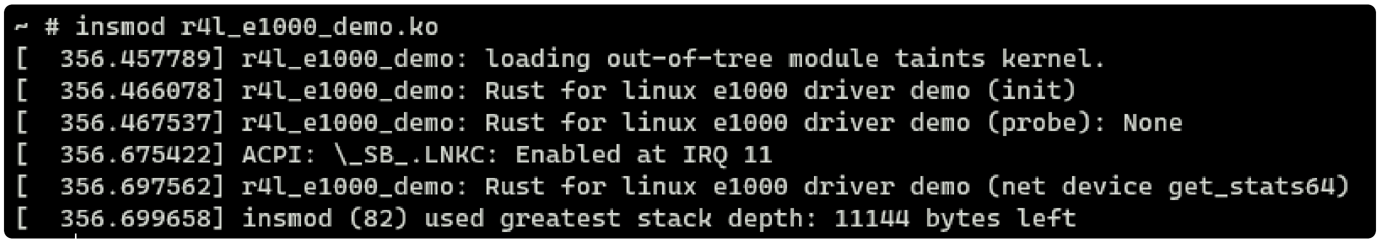
- 再次编译内核，并启动qemu，使用 `ifconfig` 命令，可以看到现在已经看不到网络接口了



```
[    1.567623] netconsole: network logging started
[    1.655241] ata2: found unknown device (class 0)
[    1.665564] ata2.00: ATAPI: QEMU DVD-ROM, 2.5+, max UDMA/100
[    1.676004] scsi 1:0:0:0: CD-ROM            QEMU     QEMU DVD-ROM     2.5+ PQ: 0 ANSI: 5
[    1.711254] sr 1:0:0:0: [sr0] scsi3-mmc drive: 4x/4x cd/rw xa/form2 tray
[    1.711659] cdrom: Uniform CD-ROM driver Revision: 3.20
[    1.731144] sr 1:0:0:0: Attached scsi generic sg0 type 5
[    2.142892] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3
[    2.277780] tsc: Refined TSC clocksource calibration: 2903.978 MHz
[    2.278270] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x29dbf0ef31e, max_idle_ns: 440795267001 ns
[    2.278787] clocksource: Switched to clocksource tsc
[   14.437965] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[   14.502573] modprobe (67) used greatest stack depth: 14272 bytes left
[   14.516902] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[   14.518901] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[   14.519440] cfg80211: failed to load regulatory.db
[   14.520980] ALSA device list:
[   14.521479]   No soundcards found.
[   14.590973] Freeing unused kernel image (initmem) memory: 1328K
[   14.591690] Write protecting the kernel read-only data: 24576k
[   14.594974] Freeing unused kernel image (text/rodata gap) memory: 2032K
[   14.595987] Freeing unused kernel image (rodata/data gap) memory: 840K
[   14.744763] x86/mm: Checked W+X mappings: passed, no W+X pages found.
[   14.745314] Run sbin/init as init process
[   14.786749] mount (72) used greatest stack depth: 14160 bytes left
[   14.930148] mdev (74) used greatest stack depth: 13928 bytes left

Please press Enter to activate this console.
~ # ifconfig
~ # 
```

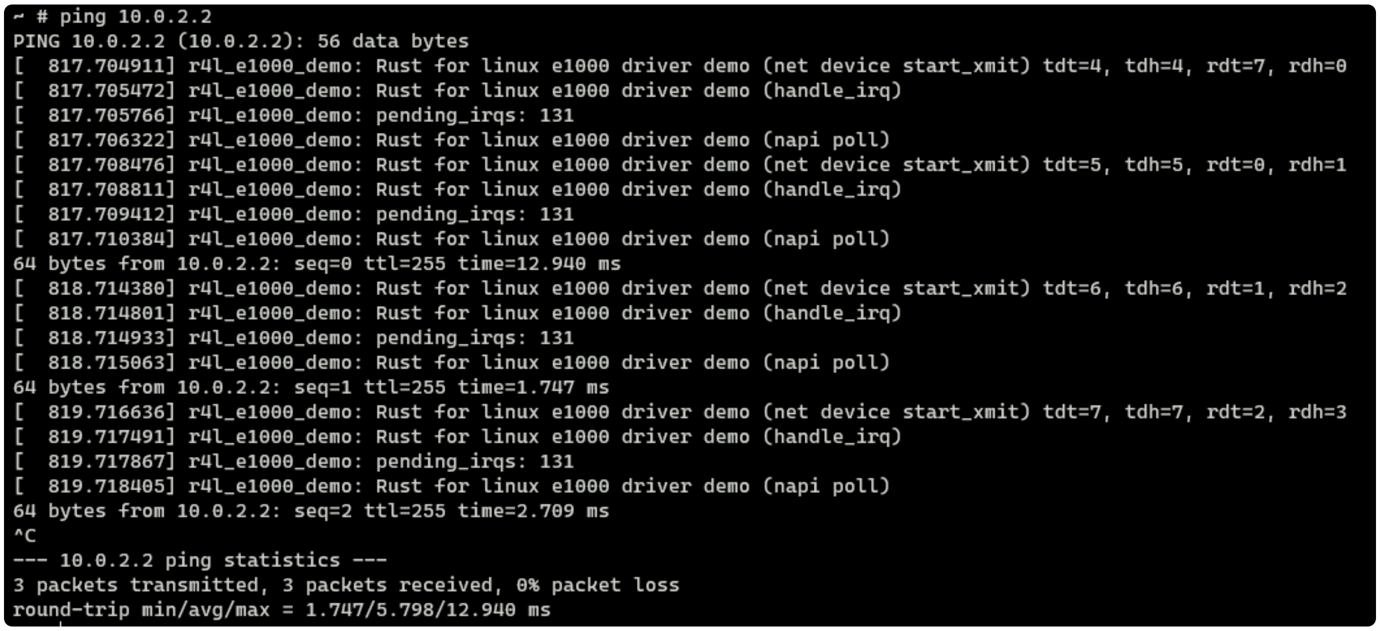
- 加载 `r4l_e1000_demo.ko` 模块，并配置网卡，在qemu启动的系统中输入：

```bash
# 加载内核模块
insmod r4l_e1000_demo.ko
# 启动名为eth0的网络接口
ip link set eth0 up
# 添加广播地址
ip addr add broadcast 10.0.2.255 dev eth0
#将 10.0.2.15 IP 地址分配给 eth0 网络接口，并将子网掩码设置为 255.255.255.0
ip addr add 10.0.2.15/255.255.255.0 dev eth0
# 添加默认路由网关
ip route add default via 10.0.2.1
```



```
~ # insmod r4l_e1000_demo.ko
[  356.457789] r4l_e1000_demo: loading out-of-tree module taints kernel.
[  356.466078] r4l_e1000_demo: Rust for linux e1000 driver demo (init)
[  356.467537] r4l_e1000_demo: Rust for linux e1000 driver demo (probe): None
[  356.675422] ACPI: \_SB_.LNKC: Enabled at IRQ 11
[  356.697562] r4l_e1000_demo: Rust for linux e1000 driver demo (net device get_stats64)
[  356.699658] insmod (82) used greatest stack depth: 11144 bytes left
```

`ping 10.0.2.2`



```
~ # ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2): 56 data bytes
[  817.704911] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=4, tdh=4, rdt=7, rdh=0
[  817.705472] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  817.705766] r4l_e1000_demo: pending_irqs: 131
[  817.706322] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
[  817.708476] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=5, tdh=5, rdt=0, rdh=1
[  817.708811] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  817.709412] r4l_e1000_demo: pending_irqs: 131
[  817.710384] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
64 bytes from 10.0.2.2: seq=0 ttl=255 time=12.940 ms
[  818.714380] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=6, tdh=6, rdt=1, rdh=2
[  818.714801] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  818.714933] r4l_e1000_demo: pending_irqs: 131
[  818.715063] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
64 bytes from 10.0.2.2: seq=1 ttl=255 time=1.747 ms
[  819.716636] r4l_e1000_demo: Rust for linux e1000 driver demo (net device start_xmit) tdt=7, tdh=7, rdt=2, rdh=3
[  819.717491] r4l_e1000_demo: Rust for linux e1000 driver demo (handle_irq)
[  819.717867] r4l_e1000_demo: pending_irqs: 131
[  819.718405] r4l_e1000_demo: Rust for linux e1000 driver demo (napi poll)
64 bytes from 10.0.2.2: seq=2 ttl=255 time=2.709 ms
^C
--- 10.0.2.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.747/5.798/12.940 ms
```

1. **编译成内核模块，是在哪个文件中以哪条语句定义的?**
   在src_e1000的目录中的Kbuild文件中的声明 `obj-m := r4l_e1000_demo.o` 告知构建系统要编译一个模块，构建系统会自动生成一个对应的 `.ko` 文件
2. **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**
   `obj-m := r4l_e1000_demo.o` 声明了要构建的模块
   Makefile中的 `$(MAKE) -C $(KDIR) M=$$PWD`
   a. -C 选项告诉构建系统到内核源代码的路径去查找内核头文件和构建规则。
   b. M=$$PWD 选项告诉构建系统去当前模块源代码目录中查找Makefile

作业三 使用rust编写一个简单的内核模块并运行

- 在 `samples/rust` 目录下添加一个 `rust_helloworld.rs`，内容如下

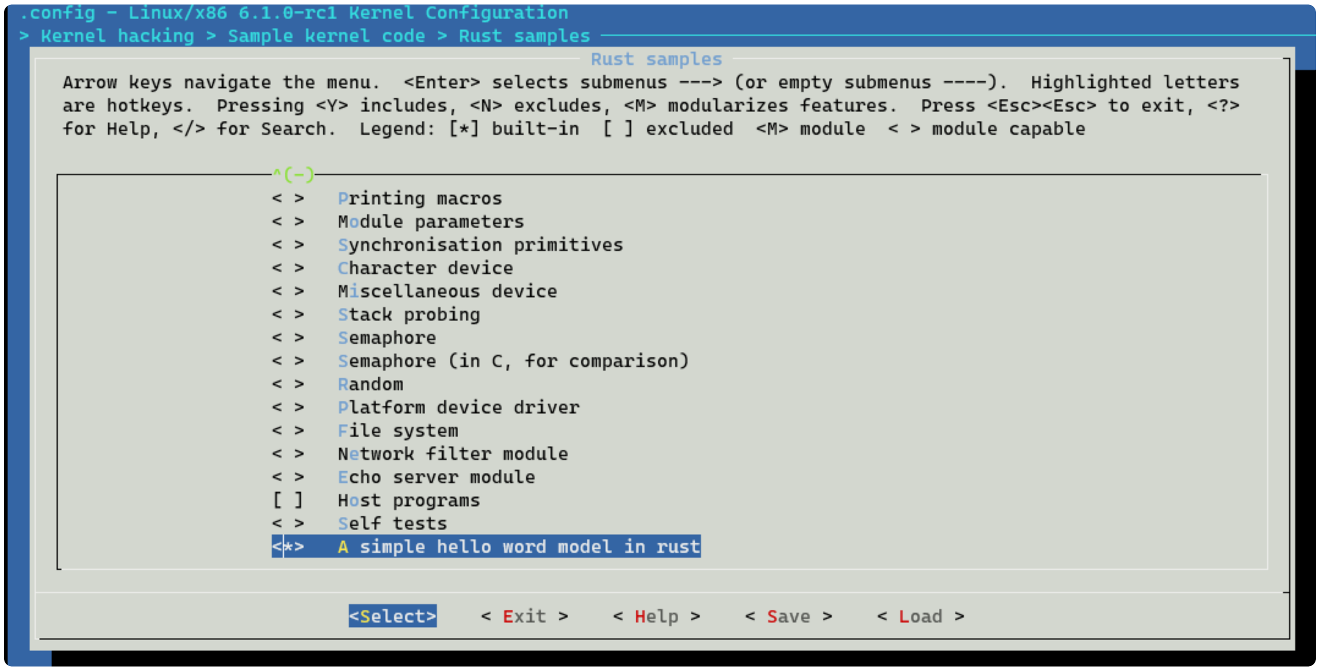
```bash
// SPDX-License-Identifier: GPL-2.0
//! Rust minimal sample.

use kernel::prelude::*;

module! {
  type: RustHelloWorld,
  name: "rust_helloworld",
  author: "whocare",
  description: "hello world module in rust",
  license: "GPL",
}

struct RustHelloWorld {}

impl kernel::Module for RustHelloWorld {
  fn init(_name: &'static CStr, _module: &'static ThisModule) -> Result<Self> {
      pr_info!("Hello World from Rust module");
      Ok(RustHelloWorld {})
  }
}
```

- 修改 `Kconfig` 文件，在最后一行 `endif # SAMPLES_RUST` 前插入以下代码

```bash
config SAMPLE_RUST_HELLOWORLD
  tristate "A simple hello word model in rust"
  default M
  help
    This option enables the Rust Hello World module.

    Say 'Y' to build the Rust Hello World module into the kernel,
    'M' to compile it as a loadable module, or 'N' to exclude it.

    The Rust Hello World module provides a simple example of a Rust module
    for the Linux kernel.

    If unsure, say 'M' to compile it as a loadable module.
```

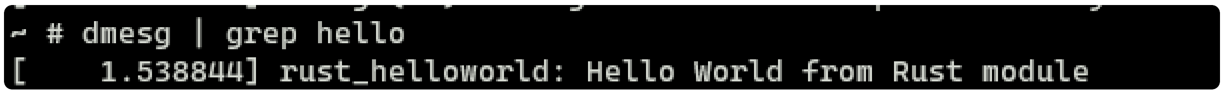
  - config SAMPLE_RUST_HELLOWORLD：这是配置选项的名称，名为 SAMPLE_RUST_HELLOWORLD。这个选项用于控制是否构建 Rust 编写的自测试案例
  - tristate：这个配置选项的类型是 tristate，表示它具有三种可能的状态：Y（启用，编译

到内核中）、N（禁用）和M（编译成一个可加载模块，`.ko` 文件）。
    - default M：这个选项的默认值为 "M"
- 修改 `Makefile` 文件，模仿文件中的语句添加 `obj-$(CONFIG_SAMPLE_RUST_HELLOWORLD)		+= rust_helloworld.o`
- 调用 `make LLVM=1 menuconfig` 配置该模块，这里选built-in试一下



```
.config - Linux/x86 6.1.0-rc1 Kernel Configuration
> Kernel hacking > Sample kernel code > Rust samples
                          Rust samples
  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

                    ^(-)
                    < >   Printing macros
                    < >   Module parameters
                    < >   Synchronisation primitives
                    < >   Character device
                    < >   Miscellaneous device
                    < >   Stack probing
                    < >   Semaphore
                    < >   Semaphore (in C, for comparison)
                    < >   Random
                    < >   Platform device driver
                    < >   File system
                    < >   Network filter module
                    < >   Echo server module
                    [ ]   Host programs
                    < >   Self tests
                    <*>   A simple hello word model in rust

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

- `make LLVM=1 -j$(nproc)` 重新编译内核
- 在 `src_e1000` 目录中用qemu 启动内核，用 `dmesg | grep` 查看内核日志



```
~ # dmesg | grep hello
[    1.538844] rust_helloworld: Hello World from Rust module
```

- 同样可以在 `menuconfig` 中配置为构建一个 `.ko` 模块文件，在系统中通过 `insmod` 来加载模块

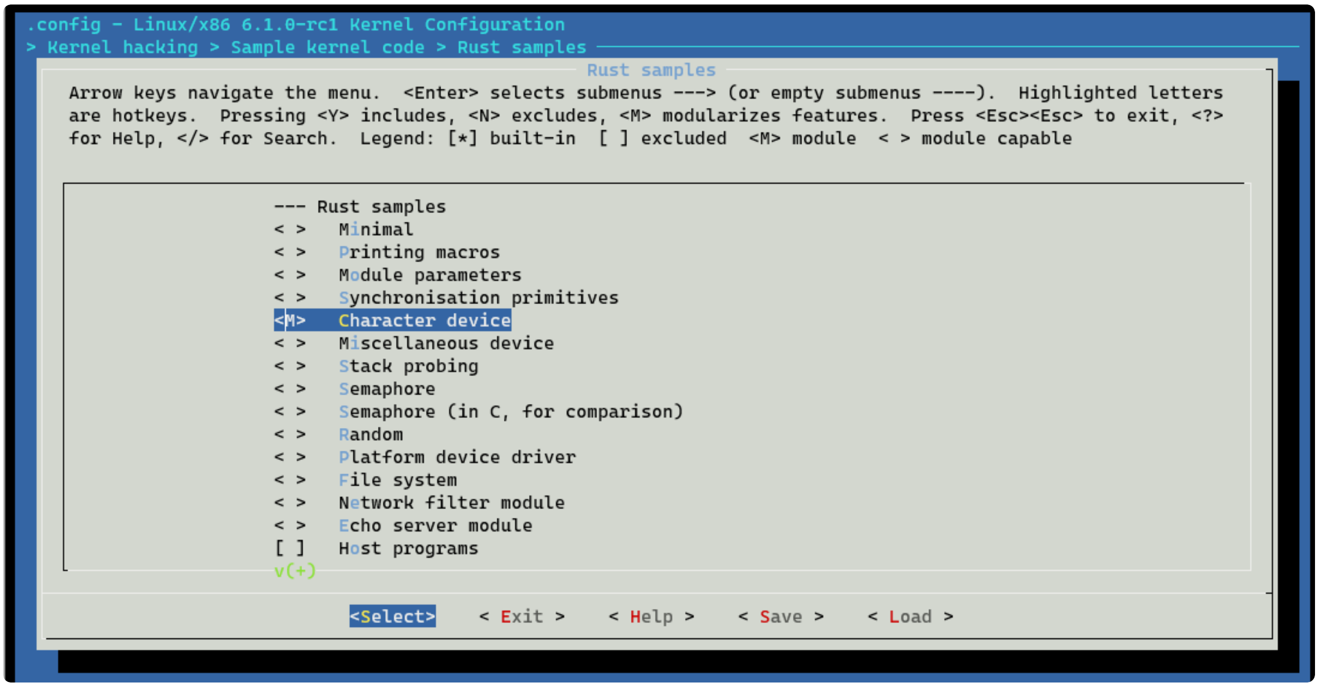
作业四

作业五 注册字符设备

- 修改 `rust_chrdev.rs`

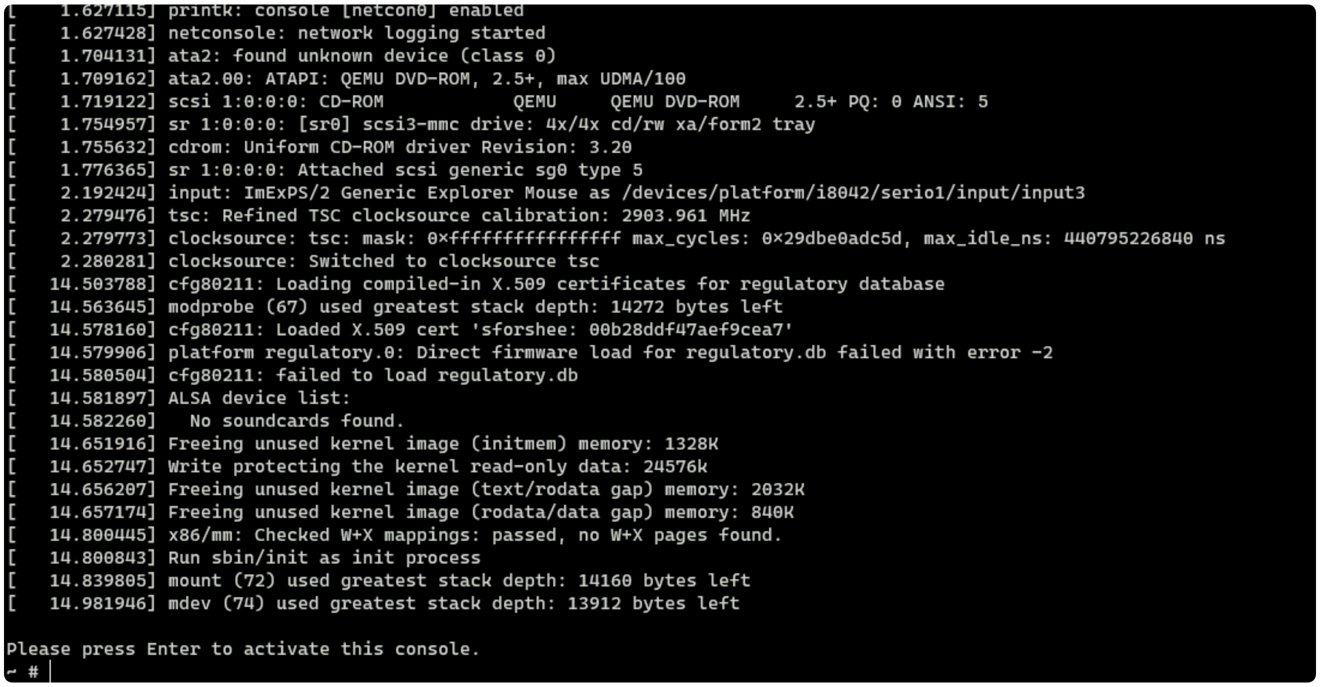
```rust
fn write(_this: &Self,_file: &file::File,_reader: &mut impl kernel::io_buffer::IoBufferReader,_offset:u64,) -> Result<usize> {
    if _reader.is_empty() {
        return Ok(0);
    }

    let mut data_buf = _this.inner.lock();

    let read_len = if _reader.len() > GLOBALMEM_SIZE {
        GLOBALMEM_SIZE
    } else {
        _reader.len()
    };

    _reader.read_slice(&mut data_buf[..read_len])?;
    Ok(read_len)
}

fn read(_this: &Self,_file: &file::File,_writer: &mut impl kernel::io_buffer::IoBufferWriter,_offset:u64,) -> Result<usize> {
    // 加锁
    let data = _this.inner.lock();

    // 计算写入长度
    let len = core::cmp::min(_writer.len(), data.len().saturating_sub(_offset as usize));

    _writer.write_slice(&data[_offset as usize..][..len])?;

    Ok(len)

}
```

- 配置 `menuconfig` ，然后重新编译



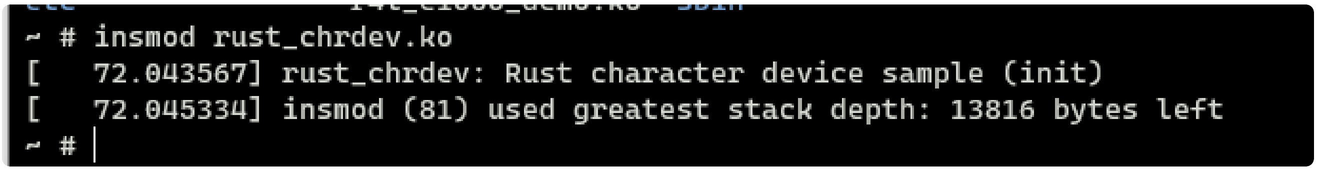
```
.config - Linux/x86 6.1.0-rc1 Kernel Configuration
> Kernel hacking > Sample kernel code > Rust samples ─────────────────────
                               Rust samples
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters
  are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>
  for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                    --- Rust samples
                    < >   Minimal
                    < >   Printing macros
                    < >   Module parameters
                    < >   Synchronisation primitives
                    <M>   Character device
                    < >   Miscellaneous device
                    < >   Stack probing
                    < >   Semaphore
                    < >   Semaphore (in C, for comparison)
                    < >   Random
                    < >   Platform device driver
                    < >   File system
                    < >   Network filter module
                    < >   Echo server module
                    [ ]   Host programs
                    v(+)

                   <Select>    < Exit >    < Help >    < Save >    < Load >
```

- 将 `rust_chrdev.ko` 复制到 `src_e1000/rootfs` 目录下，启动qemu



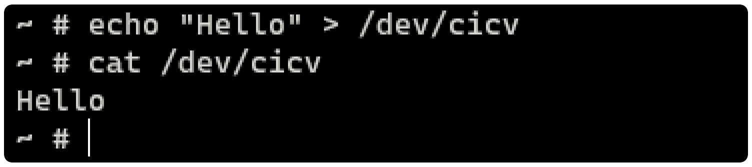
```
[    1.627115] printk: console [netcon0] enabled
[    1.627428] netconsole: network logging started
[    1.704131] ata2: found unknown device (class 0)
[    1.709162] ata2.00: ATAPI: QEMU DVD-ROM, 2.5+, max UDMA/100
[    1.719122] scsi 1:0:0:0: CD-ROM            QEMU     QEMU DVD-ROM     2.5+ PQ: 0 ANSI: 5
[    1.754957] sr 1:0:0:0: [sr0] scsi3-mmc drive: 4x/4x cd/rw xa/form2 tray
[    1.755632] cdrom: Uniform CD-ROM driver Revision: 3.20
[    1.776365] sr 1:0:0:0: Attached scsi generic sg0 type 5
[    2.192424] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3
[    2.279476] tsc: Refined TSC clocksource calibration: 2903.961 MHz
[    2.279773] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x29dbe0adc5d, max_idle_ns: 440795226840 ns
[    2.280281] clocksource: Switched to clocksource tsc
[   14.503788] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[   14.563645] modprobe (67) used greatest stack depth: 14272 bytes left
[   14.578160] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[   14.579906] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[   14.580504] cfg80211: failed to load regulatory.db
[   14.581897] ALSA device list:
[   14.582260]   No soundcards found.
[   14.651916] Freeing unused kernel image (initmem) memory: 1328K
[   14.652747] Write protecting the kernel read-only data: 24576k
[   14.656207] Freeing unused kernel image (text/rodata gap) memory: 2032K
[   14.657174] Freeing unused kernel image (rodata/data gap) memory: 840K
[   14.800445] x86/mm: Checked W+X mappings: passed, no W+X pages found.
[   14.800843] Run sbin/init as init process
[   14.839805] mount (72) used greatest stack depth: 14160 bytes left
[   14.981946] mdev (74) used greatest stack depth: 13912 bytes left

Please press Enter to activate this console.
~ #
```

- `insmod rust_chrdev.ko` 加载模块



```
~ # insmod rust_chrdev.ko
[   72.043567] rust_chrdev: Rust character device sample (init)
[   72.045334] insmod (81) used greatest stack depth: 13816 bytes left
~ #
```

- 测试



```
~ # echo "Hello" > /dev/cicv
~ # cat /dev/cicv
Hello
~ #
```

- **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？**
  通过build_image.sh创建了一个脚步/etc/init.d/rsC，其中一条命令为 `mknod /dev/cicv c 248 0` 创建一个字符设备节点 /dev/cicv，其主设备号为 248，次设备号为 0
- **它是如何与我们写的字符设备驱动关联上的？**
  `chrdev_reg.as_mut().register::<RustFile>()?;` 会调用 `alloc_chrdev_region` 函数分配一个

```Rust
/**
 * alloc_chrdev_region() - register a range of char device numbers
 * @dev: output parameter for first assigned number
 * @baseminor: first of the requested range of minor numbers
 * @count: the number of minor numbers required
 * @name: the name of the associated device or driver
 *
 * Allocates a range of char device numbers.  The major number will be
 * chosen dynamically, and returned (along with the first minor number)
 * in @dev.  Returns zero or a negative error code.
 */
int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count,
            const char *name)
{
    struct char_device_struct *cd;
    cd = __register_chrdev_region(0, baseminor, count, name);
    if (IS_ERR(cd))
        return PTR_ERR(cd);
    *dev = MKDEV(cd->major, cd->baseminor);
    return 0;
}
```

这里会动态分配一个主设备号

动态分配规则，会先从254-234开始分配

Rust

```
static int find_dynamic_major(void)
{
    int i;
    struct char_device_struct *cd;

    for (i = ARRAY_SIZE(chrdevs)-1; i >= CHRDEV_MAJOR_DYN_END; i--) {
        if (chrdevs[i] == NULL)
            return i;
    }

    for (i = CHRDEV_MAJOR_DYN_EXT_START;
         i >= CHRDEV_MAJOR_DYN_EXT_END; i--) {
        for (cd = chrdevs[major_to_index(i)]; cd; cd = cd->next)
            if (cd->major == i)
                break;

        if (cd == NULL)
            return i;
    }

    return -EBUSY;
}
```