

Task5

Step1 - 修改 rust_chrdev.rs 源码

Rust

```
1 // 写入字符设备
2 fn write(_this: &Self, _file: &file::File, _reader: &mut impl kernel::io
  _buffer::IoBufferReader, _offset: u64,) -> Result<usize> {
3     let len = _reader.len();
4     let data = &mut _this.inner.lock();
5     _reader.read_slice(&mut data[..len])?;
6     Ok(len)
7 }
8
9 // 读取字符设备
10 fn read(_this: &Self, _file: &file::File, _writer: &mut impl kernel::io_
  buffer::IoBufferWriter, _offset: u64,) -> Result<usize> {
11     let offset: usize = _offset.try_into()?;
12     let data = &mut *_this.inner.lock();
13     let len = data.len();
14     let len = core::cmp::min(_writer.len(), len.saturating_sub(offse
    t));
15     _writer.write_slice(&data[offset..][..len])?;
16     Ok(len)
17 }
```

Step2 - 关联 rust_chrdev 模块

```
.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 >
```

Step3 - 重新编译 Linux 内核

Bash

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

Step4 - 注册字符设备到 `Linux` 系统

1. 复制 `rust_chrdev.ko` 文件到 `src_e1000/rootfs`
2. 运行 `./build_image.sh`
3. 安装 `rust_chrdev` 模块: `insmod rust_chrdev.ko`
4. 输入字符: `echo "Hello" > /dev/cicv`
5. 输出字符: `cat /dev/cicv`

```

~ # insmod rust_chrdev.ko
[35.592374] rust_chrdev: Rust character device sample (init)
[35.594825] insmod (81) used greatest stack depth: 13864 bytes left
~ # echo "hello" > /dev/cicv
~ # cat /dev/cicv
hello

```

随后在 Linux shell 下使用 ls 命令，你将发现一份新的 rust_helloworld.ko 文件

Q & A

Q1: 字符设备 `/dev/cicv` 是怎么创建的？

1. 初始化 `cdev`

```

'''
/// You may call this once per device type, up to `N` times.
pub fn register<T: file::Operations<OpenData = ()>>(self: Pin<&mut Self>) -> Result {
    // SAFETY: We must ensure that we never move out of `this`.
    let this: &mut Registration<N> = unsafe { self.get_unchecked_mut() };
    if this.inner.is_none() {
        let mut dev: bindings::dev_t = 0;
        // SAFETY: Calling unsafe function. `this.name` has `static`
        // lifetime.
        let res: i32 = unsafe {
            bindings::alloc_chrdev_region(
                arg1: &mut dev,
                arg2: this.minors_start.into(),
                arg3: N.try_into()?,
                arg4: this.name.as_char_ptr(),
            )
        };
        if res != 0 {
            return Err(Error::from_kernel_errno(res));
        }
    }
}

```

2. 注册 `cdev`

```

// SAFETY: The adapter doesn't retrieve any state yet, so it's compatible with any
// registration.
let fops: &file_operations = unsafe { file::OperationsVtable::<Self, T>::build() };
let mut cdev: <Result<Cdev, Error> as Try>::Output = Cdev::alloc(fops, this.this_module)?;
cdev.add(inner.dev + inner.used as bindings::dev_t, 1)?;
inner.cdevs[inner.used].replace(cdev);
inner.used += 1;
Ok(())
fn register

```

Q2: 设备号是多少？

查看设备号: `ls -l /dev | grep cicv`

```
~ # ls -l /dev | grep cicv
crw-r--r-- 1 0 0 248, 0 Nov 15 07:37 cicv
```

主设备号 248, 次设备号 0

Q3: 如何和字符设备驱动关联上的?

1. 每当打开设备文件时(`/dev/cicv`), 可以根据设备文件对应的 `struct inode` 结构体描述的信息知道接下来操作的设备类型
2. 根据 `struct inode` 结构体里面记录的设备号, 可以找到对应的驱动程序(字符设备)
3. 字符设备对应的 `struct cdev` 结构体描述了字符设备的所有信息, 包括字符设备的操作函数接口
4. 找到 `struct cdev` 结构体之后, Linux 内核会将 `struct cdev` 结构体所在的内存空间首地址记录在 `struct inode` 结构体的 `i_cdev` 成员中, 将 `struct cdev` 结构体中记录的操作函数接口地址记录在 `struct file` 结构体的 `f_op` 成员中
5. 接下来上层可以通过文件描述符 `fd` 找到 `struct file`, 然后找到操作字符设备的函数接口

