Task5

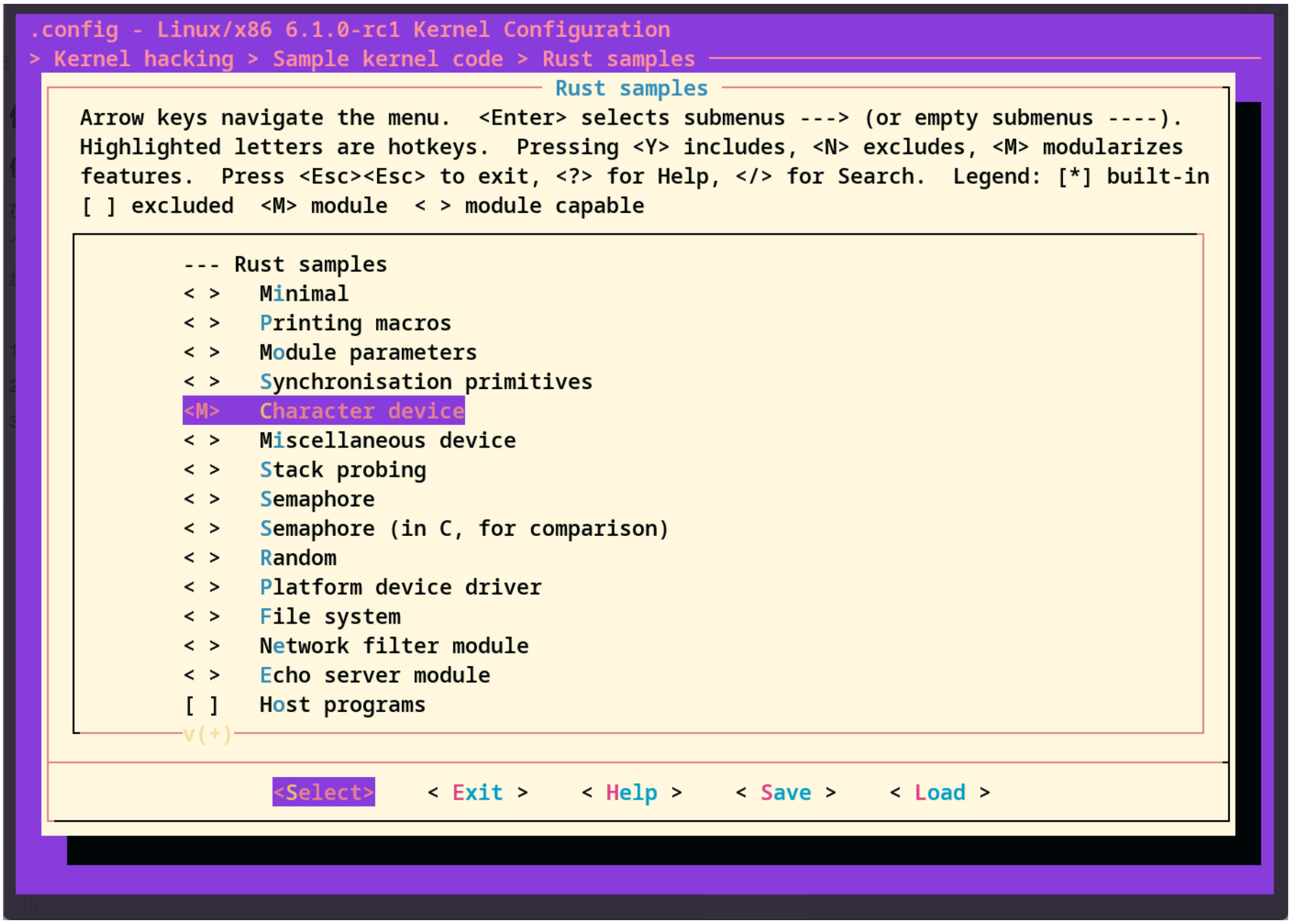
Step1 - 修改 rust_chrdev.rs 源码

Rust

```rust
// 写入字符设备
fn write(_this: &Self,_file: &file::File,_reader: &mut impl kernel::io_buffer::IoBufferReader,_offset:u64,) -> Result<usize> {
    let len = _reader.len();
    let data = &mut _this.inner.lock();
    _reader.read_slice(&mut data[..len])?;
    Ok(len)
}

// 读取字符设备
fn read(_this: &Self,_file: &file::File,_writer: &mut impl kernel::io_buffer::IoBufferWriter,_offset:u64,) -> Result<usize> {
    let offset: usize = _offset.try_into()?;
    let data = &mut *_this.inner.lock();
    let len = data.len();
    let len = core::cmp::min(_writer.len(), len.saturating_sub(offset));
    _writer.write_slice(&data[offset..][..len])?;
    Ok(len)
}
```

Step2 - 关联 rust_chrdev 模块



Step3 - 重新编译 Linux 内核

```Bash
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
```

Q & A

Q1: 字符设备 `/dev/cicv` 是怎么创建的？

1. 初始化 `cdev`

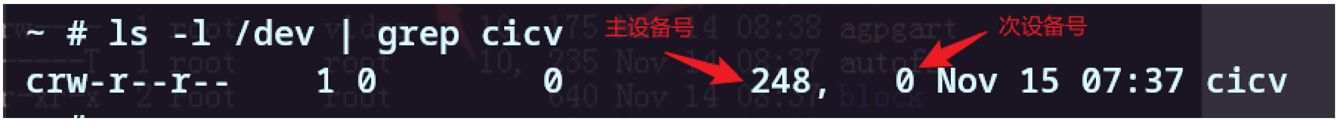
```rust
/// You may call this once per device type, up to `N` times.
pub fn register<T: file::Operations<OpenData = ()>>(self: Pin<&mut Self>) -> Result {
    // SAFETY: We must ensure that we never move out of `this`.
    let this: &mut Registration<N> = unsafe { self.get_unchecked_mut() };
    if this.inner.is_none() {
        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`

```rust
// 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`



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