操作系统实验1

> 邓人嘉 21301032

一、实验步骤

1.1 创建一个Rust项目

- 进入mnt目录，执行"cargo new os --bin"

```
[root@dcea8e5aca3c mnt]# cargo new os --bin
     Created binary (application) `os` package
[root@dcea8e5aca3c mnt]#
```

- cd os
  cargo build
  cargo run

```
[root@dcea8e5aca3c mnt]# cd os
[root@dcea8e5aca3c os]# ls
Cargo.toml  src
[root@dcea8e5aca3c os]# cargo build
   Compiling os v0.1.0 (/mnt/os)
    Finished dev [unoptimized + debuginfo] target(s) in 2.06s
[root@dcea8e5aca3c os]# cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.08s
     Running `target/debug/os`
Hello, world!
[root@dcea8e5aca3c os]#
```

1.2 移除标准库依赖

- 修改target为riscv64

```
[root@dcea8e5aca3c os]# mkdir .cargo
[root@dcea8e5aca3c os]# cd .cargo
[root@dcea8e5aca3c .cargo]# touch config
```

```
@dcea8e5aca3c:/mnt/os/.cargo
1 # os/.cargo/config
2 [build]
3 target = "riscv64gc-unknown-none-elf"
4
~
4,0-1         All
```

-
```
[root@dcea8e5aca3c os]# ls
Cargo.lock  Cargo.toml  src  target
[root@dcea8e5aca3c os]# cd src
[root@dcea8e5aca3c src]# ls
main.rs
[root@dcea8e5aca3c src]# vim main.rs
```

```
@dcea8e5aca3c/mnt/os/src
#![no_std]
#![no_main]

use core::panic::PanicInfo;

#[panic_handler]
fn panic(_info: &PanicInfo) -> ! {
        loop {}
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"main.rs" 9L, 123B                                                    1,1           All
```

- 安装相关软件包

```
[root@dcea8e5aca3c os]# rustup target add riscv64gc-unknown-none-elf
info: component 'rust-std' for target 'riscv64gc-unknown-none-elf' is up to date
[root@dcea8e5aca3c os]# cargo install cargo-binutils
    Updating `ustc` index
     Ignored package `cargo-binutils v0.3.6` is already installed, use --force to override
[root@dcea8e5aca3c os]# rustup component add llvm-tools-preview
info: component 'llvm-tools' for target 'x86_64-unknown-linux-gnu' is up to date
[root@dcea8e5aca3c os]# rustup component add rust-src
info: component 'rust-src' is up to date
[root@dcea8e5aca3c os]#
```

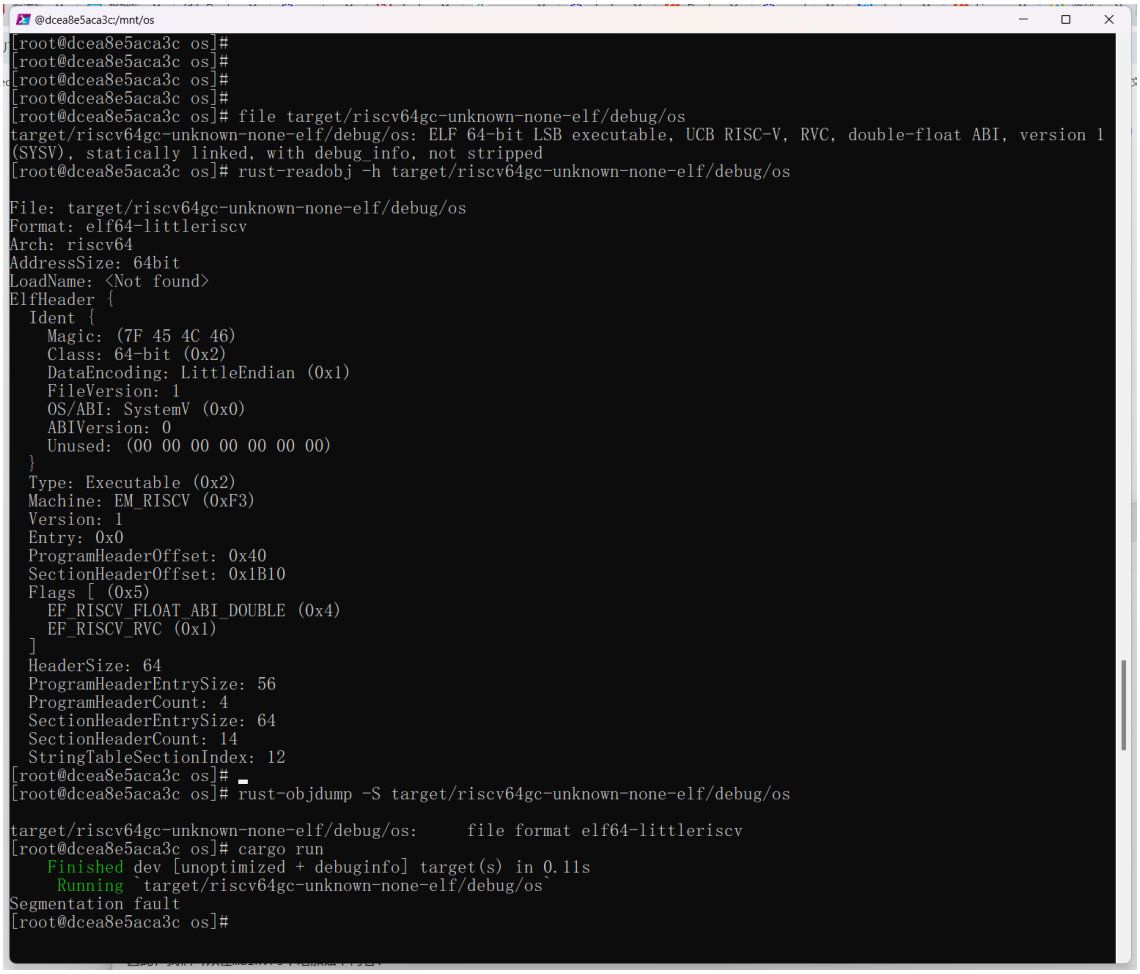
-
```
[root@dcea8e5aca3c os]# cargo build
    Finished dev [unoptimized + debuginfo] target(s) in 0.11s
[root@dcea8e5aca3c os]#
```

- 这部分的git提交

```
lenovo@LAPTOP-2O3JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git commit -m "21301032drj"
On branch main
nothing to commit, working tree clean
```

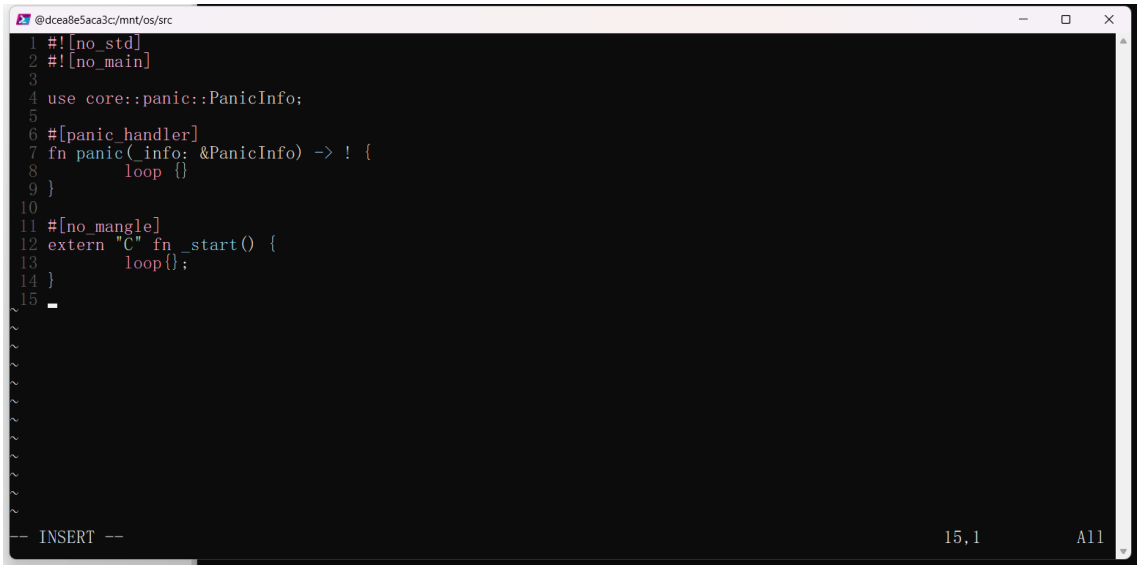
```
MINGW64:/c/Users/lenovo/Desktop/OperatingSystem/GardenerOS
lenovo@LAPTOP-2O3JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git remoet
git: 'remoet' is not a git command. See 'git --help'.

The most similar command is
        remote
lenovo@LAPTOP-2O3JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git remote
origin

lenovo@LAPTOP-2O3JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git push origin main
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 16 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (19/19), 244.74 KiB | 709.00 KiB/s, done.
Total 19 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To github.com:lovekdl/GardenerOS.git
   9455f19..4fedc75  main -> main

lenovo@LAPTOP-2O3JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$
```

- 执行命令分析移除标准库后的独立可执行程序。

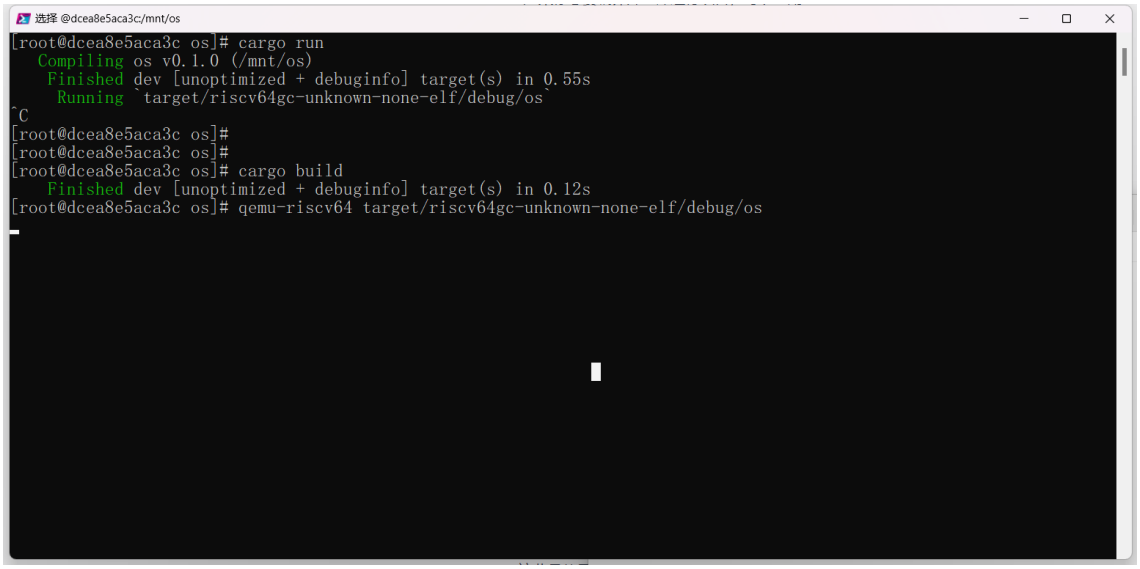


1.3 用户态可执行的环境

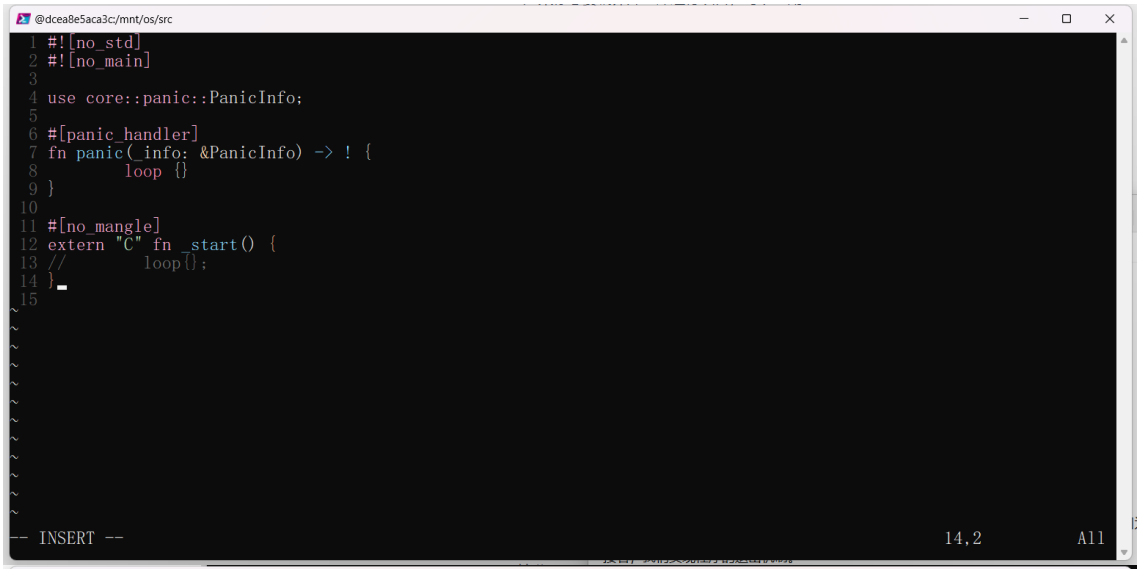
- 增加入口函数

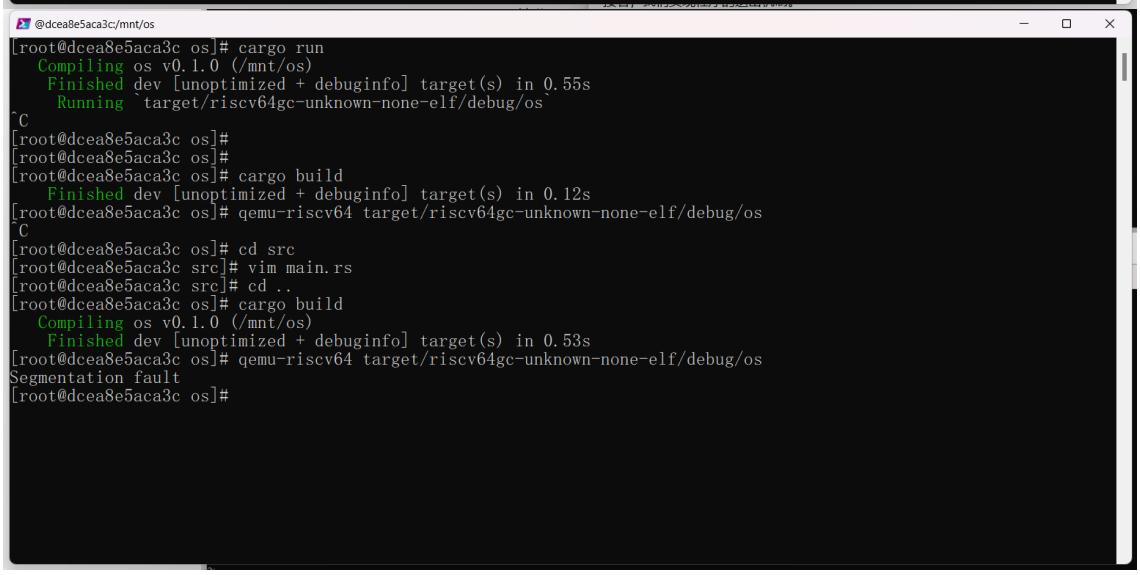


- 编译运行

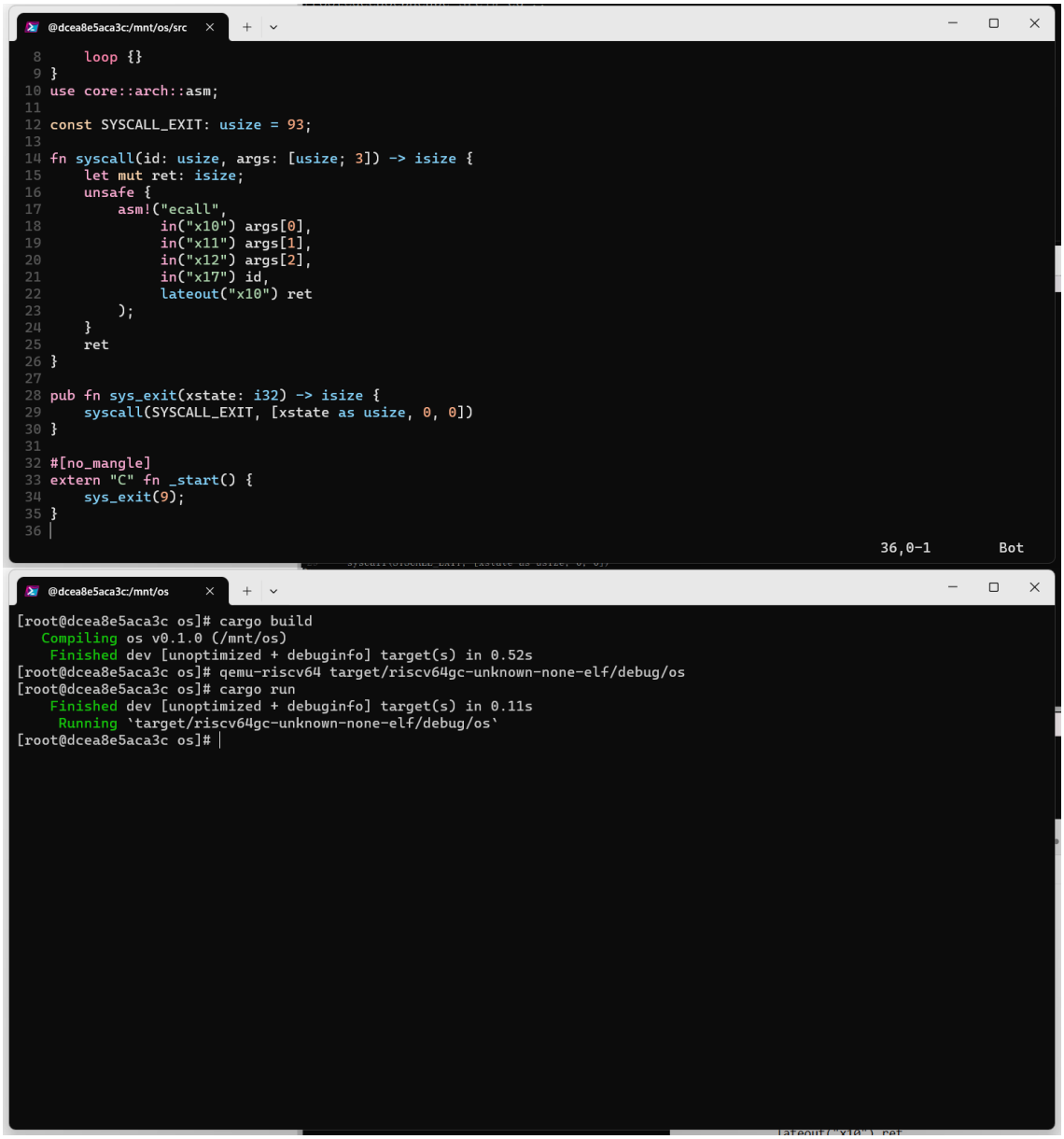


- 去掉loop尝试

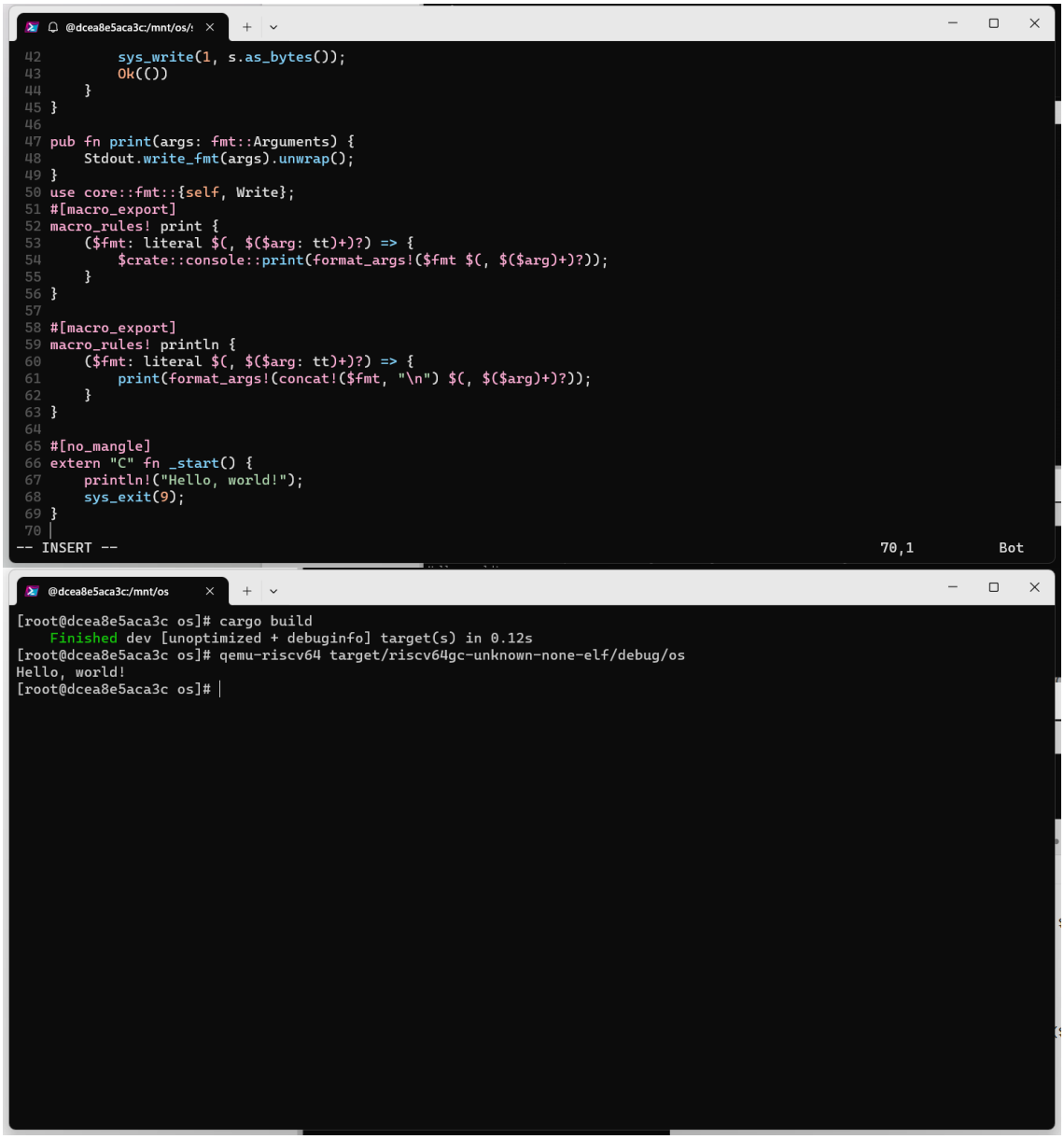




- 实现退出机制


@dcea8e5aca3c:/mnt/os/src
loop {}
}
use core::arch::asm;
const SYSCALL_EXIT: usize = 93;
fn syscall(id: usize, args: [usize; 3]) -> isize {
let mut ret: isize;
unsafe {
asm!("ecall",
in("x10") args[0],
in("x11") args[1],
in("x12") args[2],
in("x17") id,
lateout("x10") ret
);
}
ret
}
pub fn sys_exit(xstate: i32) -> isize {
syscall(SYSCALL_EXIT, [xstate as usize, 0, 0])
}
#[no_mangle]
extern "C" fn _start() {
sys_exit(9);
}
36,0-1 Bot
@dcea8e5aca3c:/mnt/os
[root@dcea8e5aca3c os]# cargo build
Compiling os v0.1.0 (/mnt/os)
Finished dev [unoptimized + debuginfo] target(s) in 0.52s
[root@dcea8e5aca3c os]# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
[root@dcea8e5aca3c os]# cargo run
Finished dev [unoptimized + debuginfo] target(s) in 0.11s
Running `target/riscv64gc-unknown-none-elf/debug/os`
[root@dcea8e5aca3c os]#

- 实现输出支持



二、思考问题

- 为什么称最后实现的程序为独立的可执行程序，它和标准的程序有什么区别？
  答：因为独立的可执行程序是源代码在经过编译处理最后形成的程序，具有独立性，不需要依赖到外部的库，通常可以直接运行，不需要额外的编译等操作。标准的程序需要依赖外部的库，文件体积相对较小，需要特定的运行环境。总的来说，标准的程序可以用于开发阶段，而独立的可执行程序移植性高，可直接运行。
- 实现和编译独立可执行程序的目的是什么？
  独立可执行程序移植性高，方便在环境不同的设备上部署，可以简化操作。独立可执行程序对外界环境依赖小，内部不易修改，所以安全性也很高。

三、git截图

- git提交(https://github.com/lovekdl/GardenerOS)