|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| 北京交通大学软件学院  **《操作系统》课程**  **实验报告** | | |

|  |
| --- |
| **姓名:韩熔** |
| **学号:20301036** |

目录

[1 实验目的 3](#_Toc118133029)

[2 实验过程设计 3](#_Toc118133030)

[3 源代码及注释 3](#_Toc118133031)

[4 运行结果与分析 4](#_Toc118133032)

[5 实验总结 4](#_Toc118133033)

# 1 实验目的

（说明：写出本次实验的主要实验目的与原因。）

挖掘一个Rust 应用程序下面的多层执行环境（移除标准库依赖，移除main函数，编译器会生成一个空程序）

学习软硬件平台（包括 RISC-V 架构）的一些基础知识

了解Linux的一些基础指令

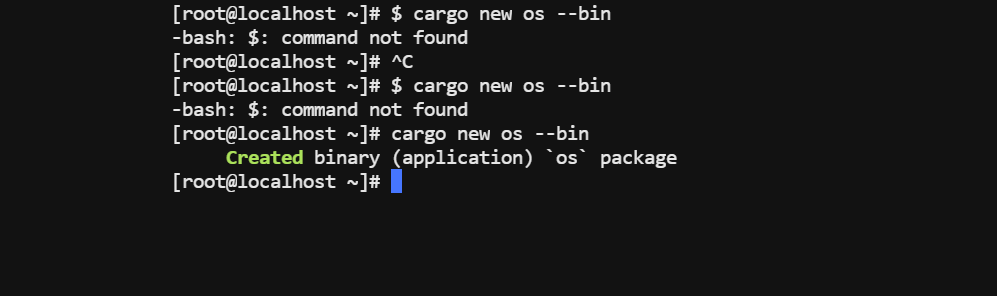
# 2 实验过程设计

（说明：写出本次实验的主要实验流程或个人实验的实施过程。）

## 执行应用程序

1. 使用 Cargo 工具来创建一个 Rust 项目，是一个可执行程序项目而不是函数库项目

cargo new os --bin

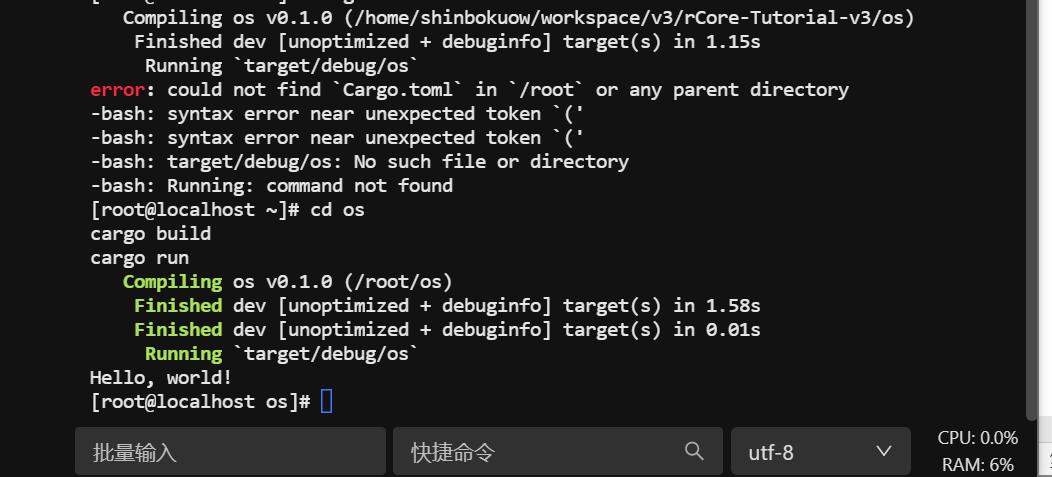


1. 进入 os 项目根目录下，利用 Cargo 工具，查看应用运行结果

cd os

cargo build

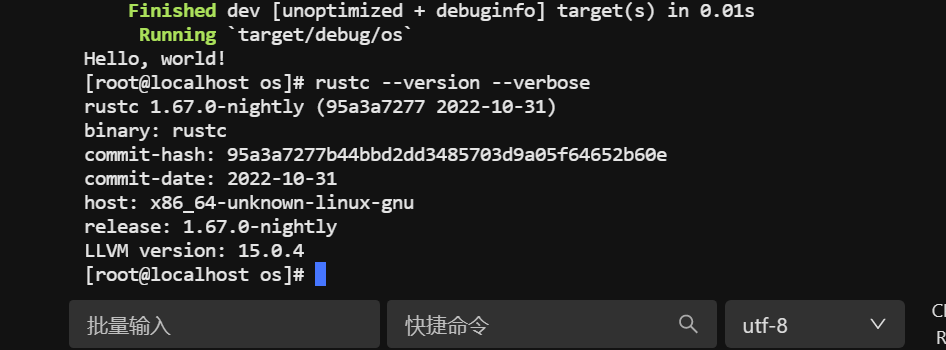
cargo run



## 平台与目标三元组

1. 查看目前 Hello, world! 程序的目标三元组是什么

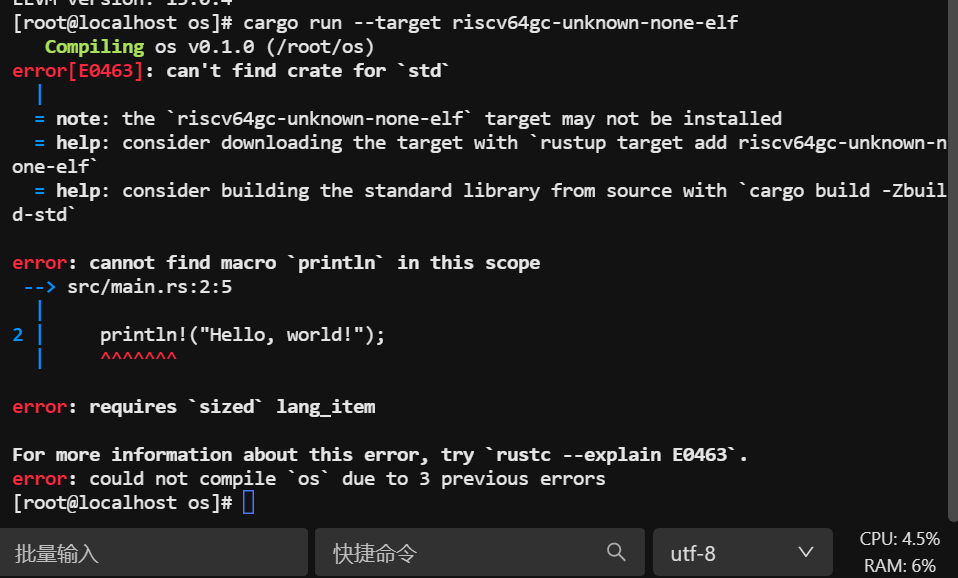
rustc --version --verbose



## 修改目标平台

1. 将程序的目标平台换成 riscv64gc-unknown-none-elf

cargo run --target riscv64gc-unknown-none-elf



报错 目标平台上没有 Rust 标准库 std

## 移除标准库依赖

1. 首先要给 rustc 添加一个target : riscv64gc-unknown-none-elf

rustup target add riscv64gc-unknown-none-elf



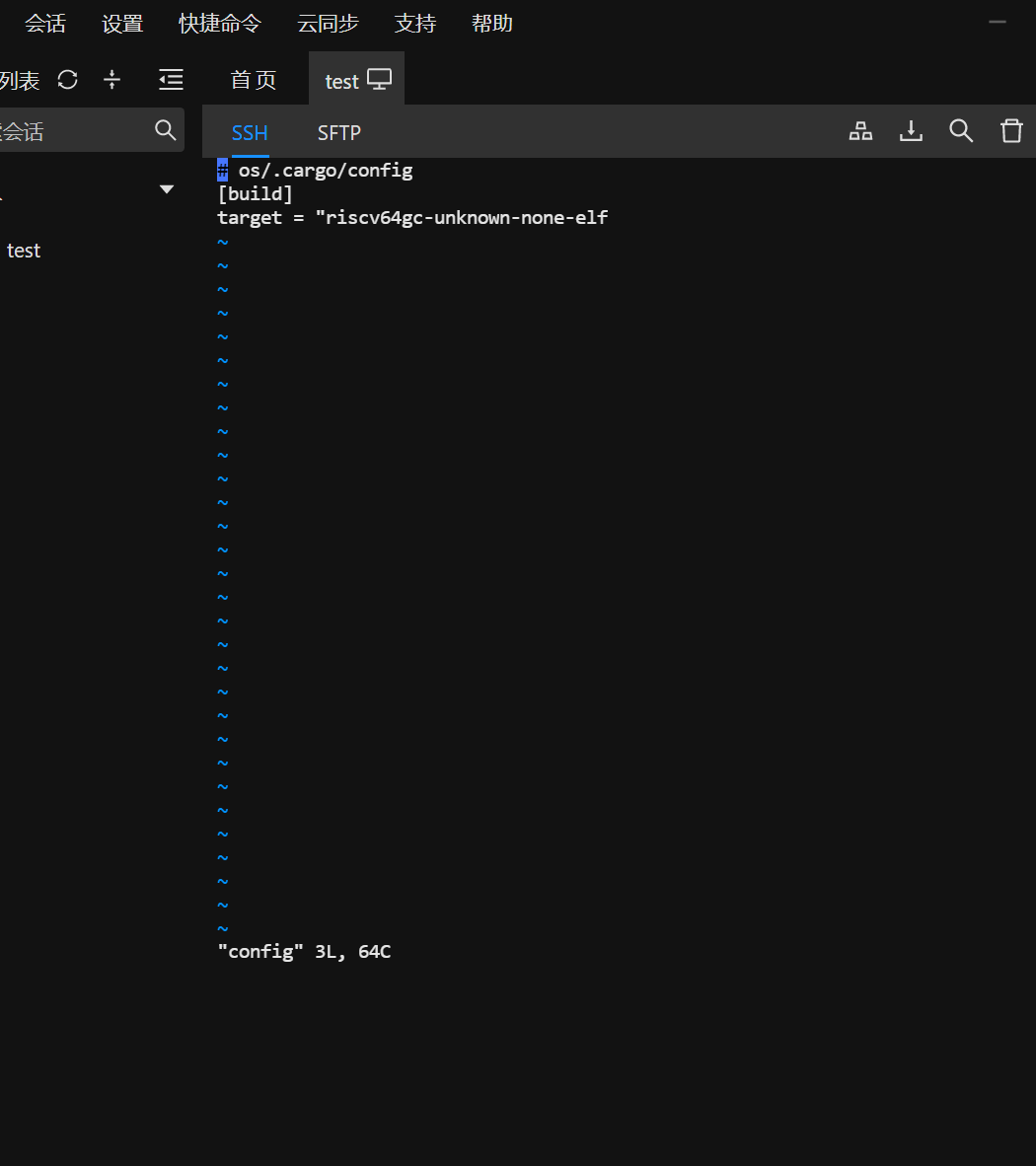
1. 在 os 目录下新建 .cargo 目录，并在这个目录下创建 config 文件

cd os

mkdir .cargo

cd .cargo/

vi config



：w 写

：q 退出

：wq 写并退出

其他指令

cd 返回根目录

ls 查看当前目录

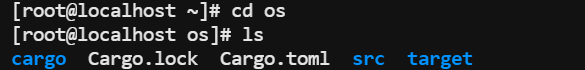
## 移除println!()宏

1. 在 main.rs 的开头加上一行 #![no\_std]

一些基本指令

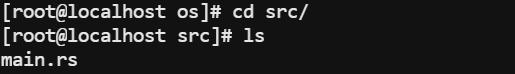
cd os

ls

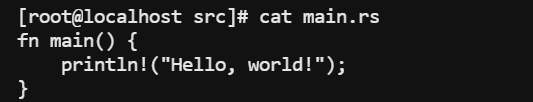


cd src/

ls

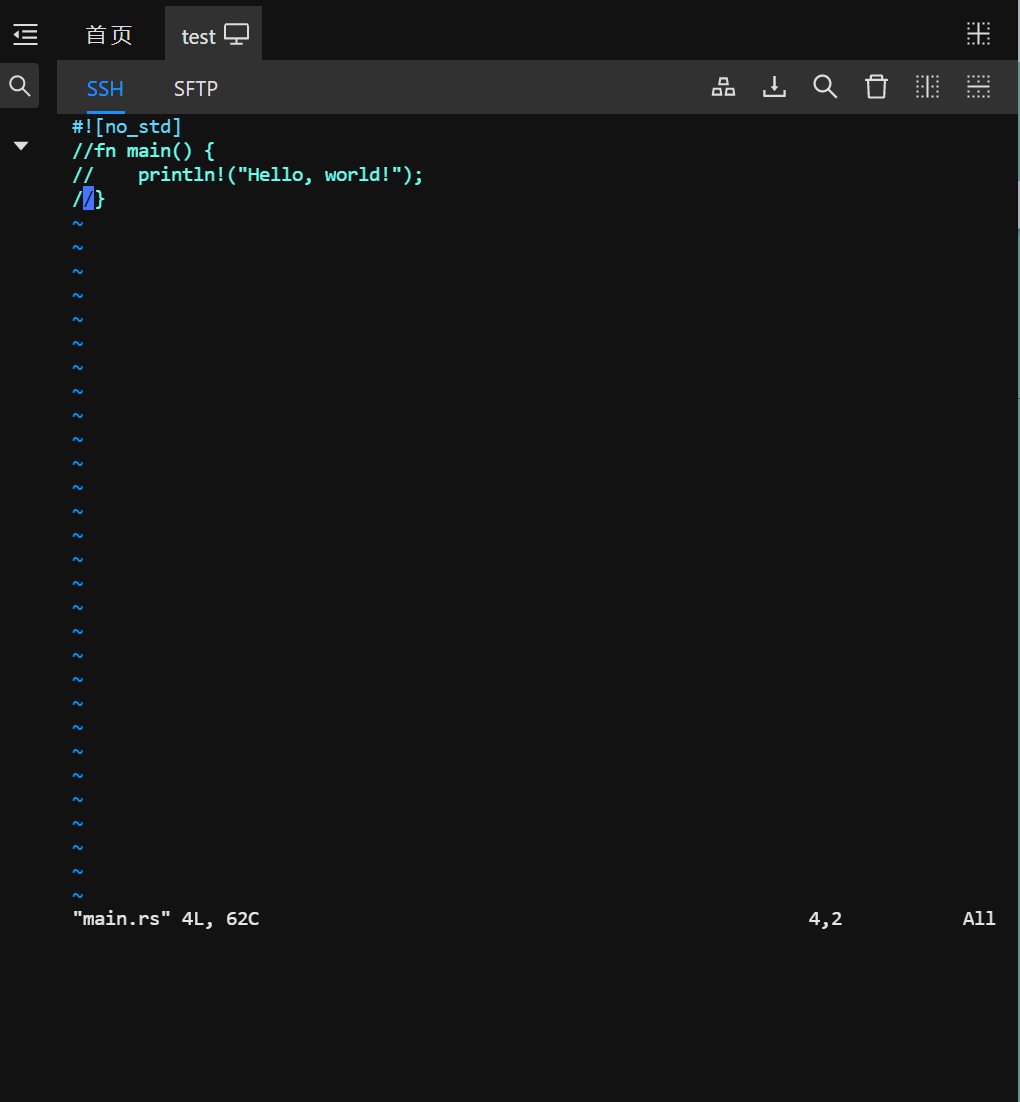


cat main.rs

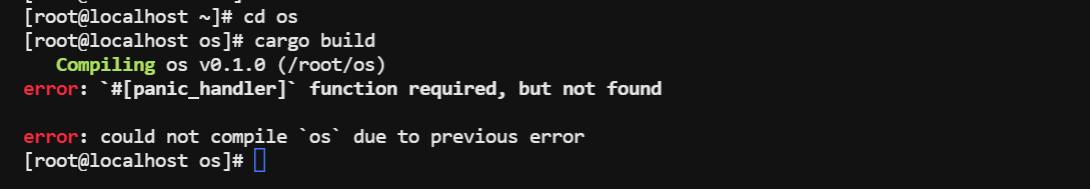


vim main.rs

注释掉之前使用的println!()宏

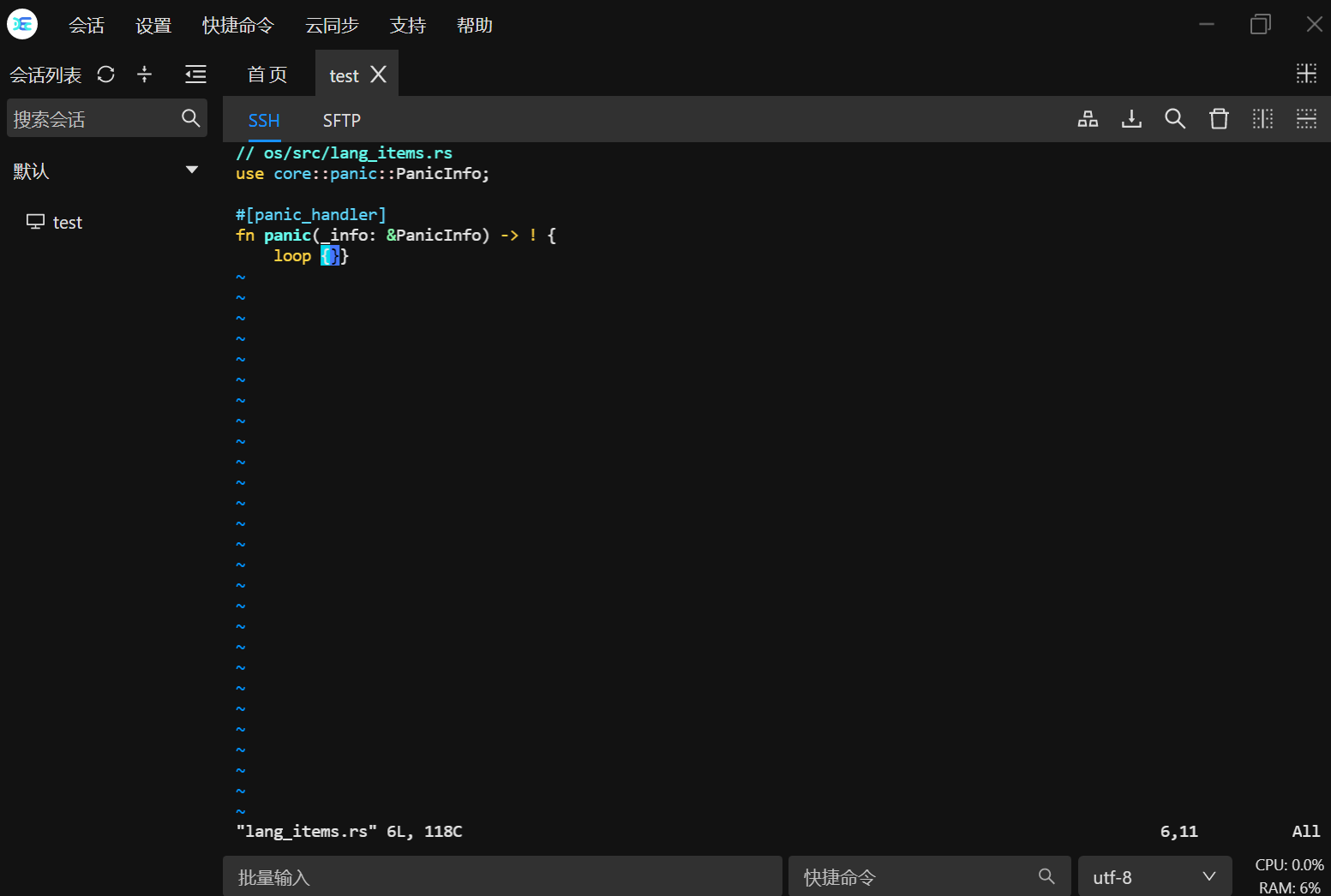


重新编译执行，得到一个报错的程序

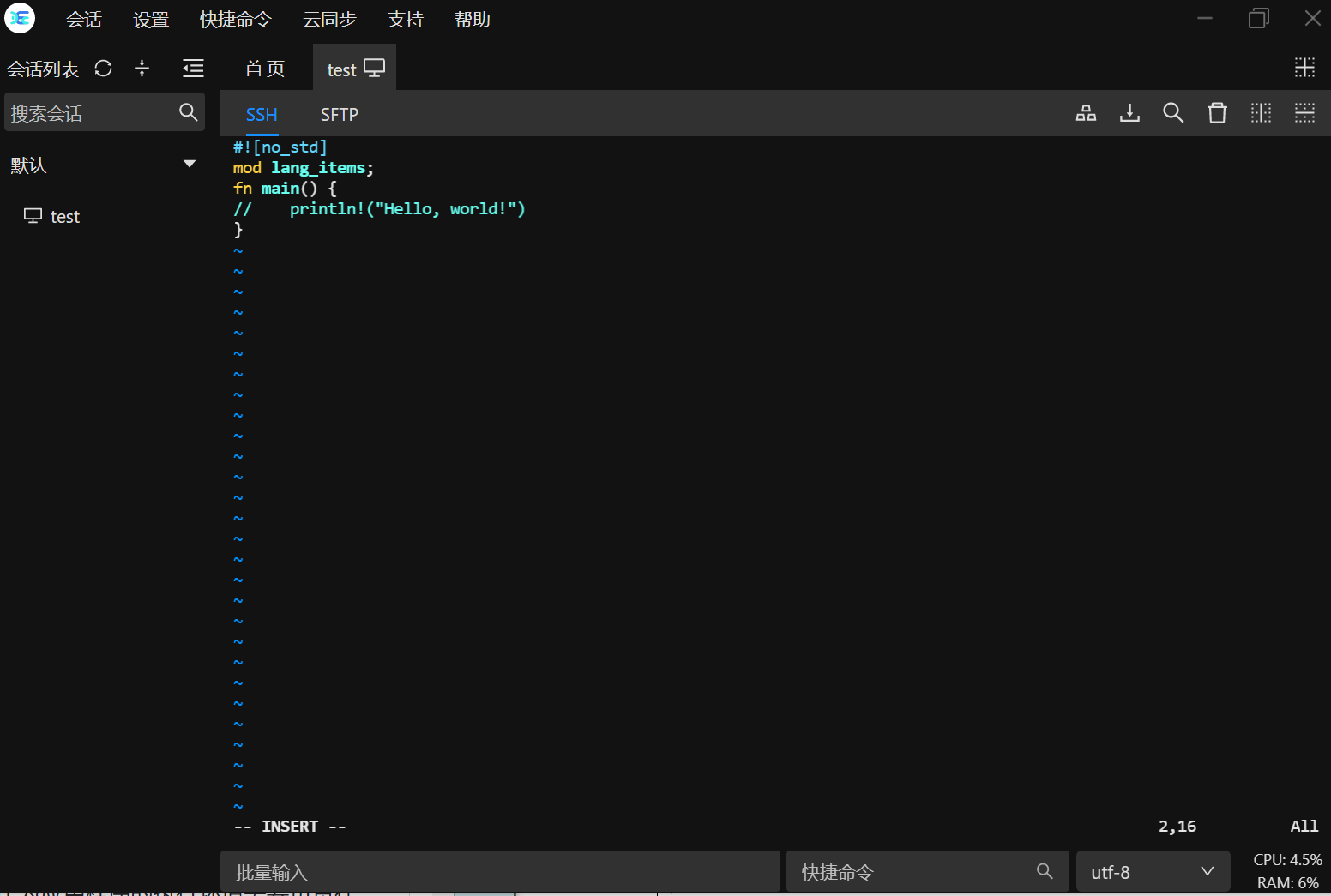


1. 创建一个新的子模块 lang\_items.rs，配置panic\_handler

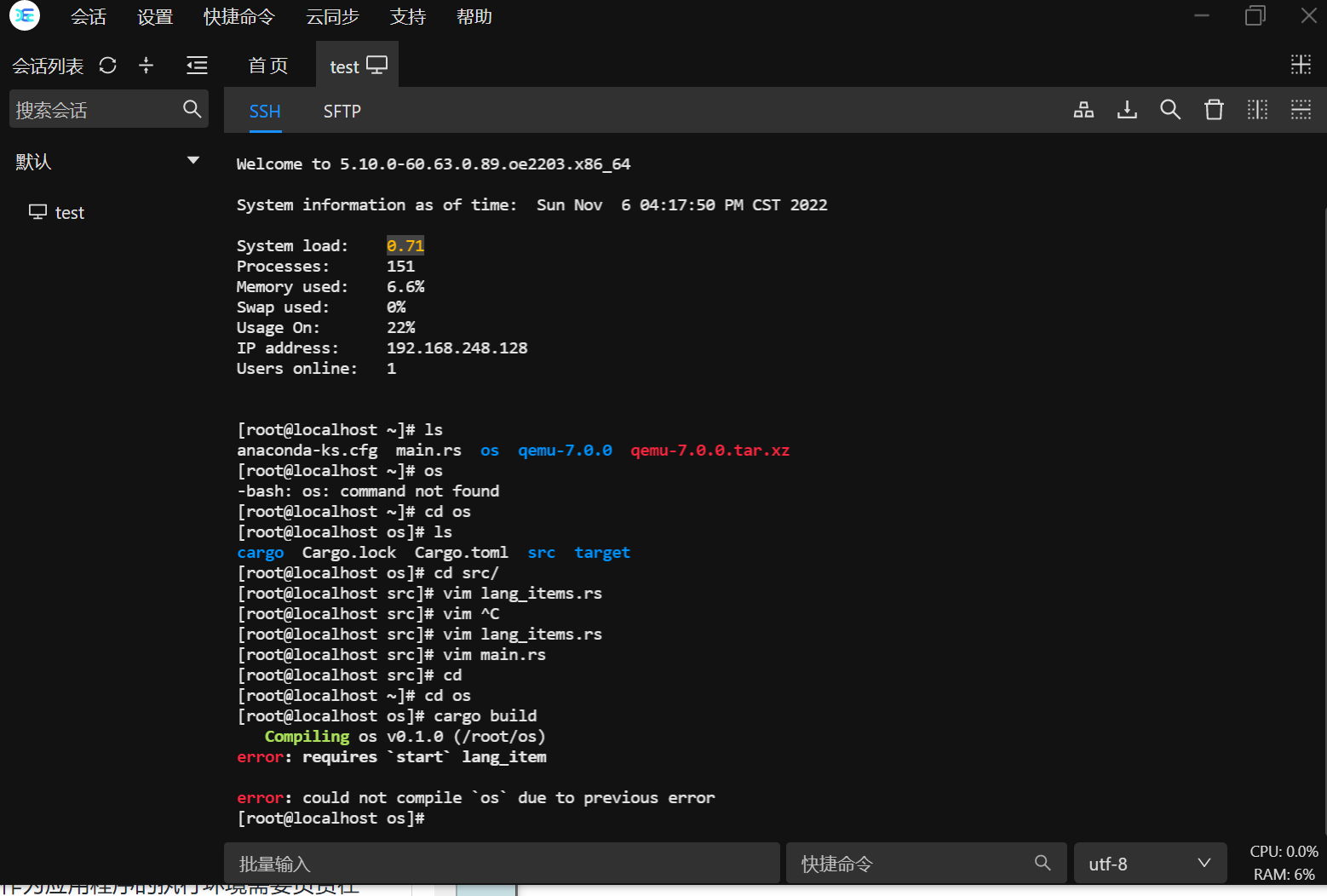
vim lang\_items.rs



在os/src/main.rs文件中添加以下内容

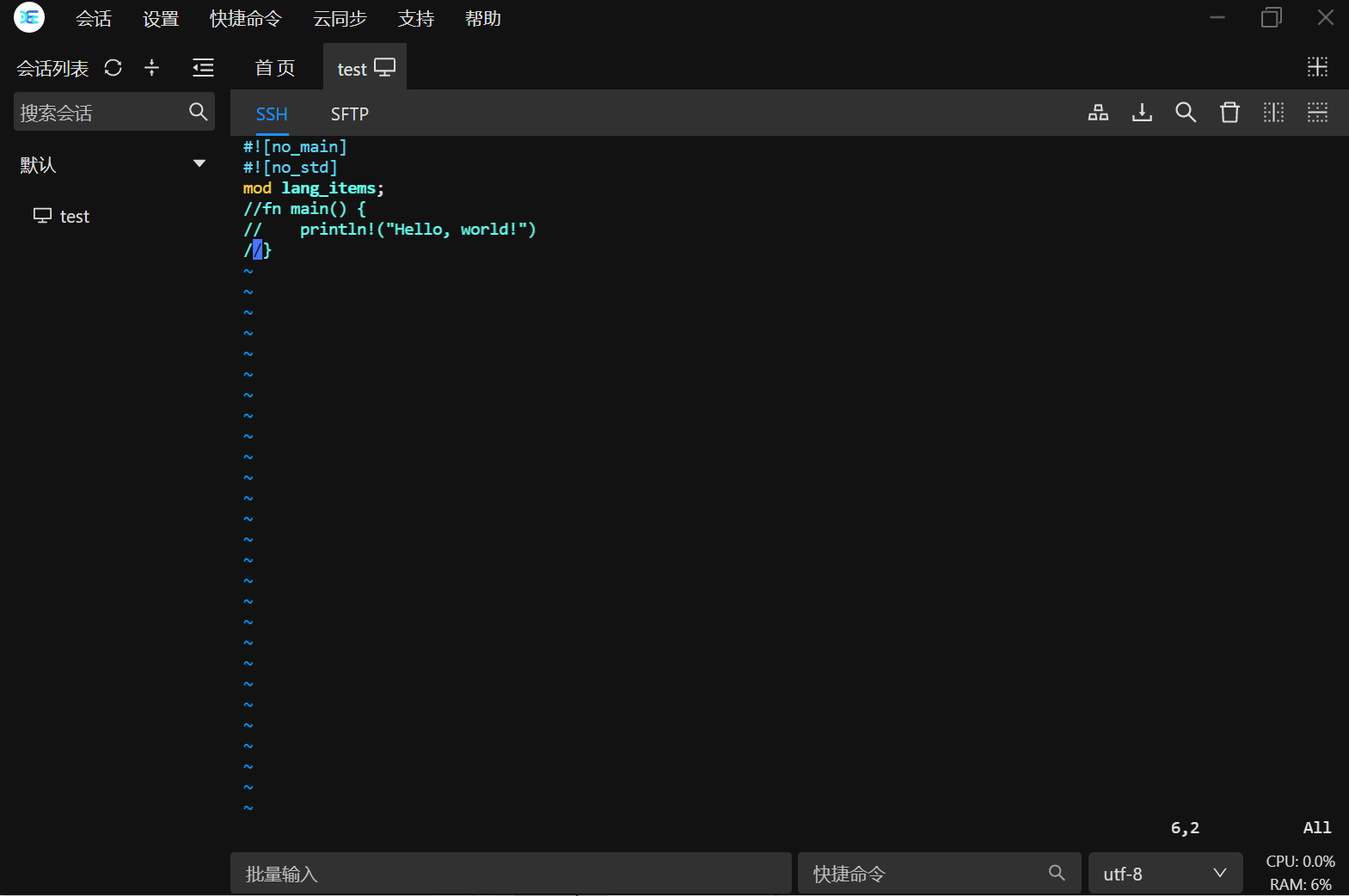


报错

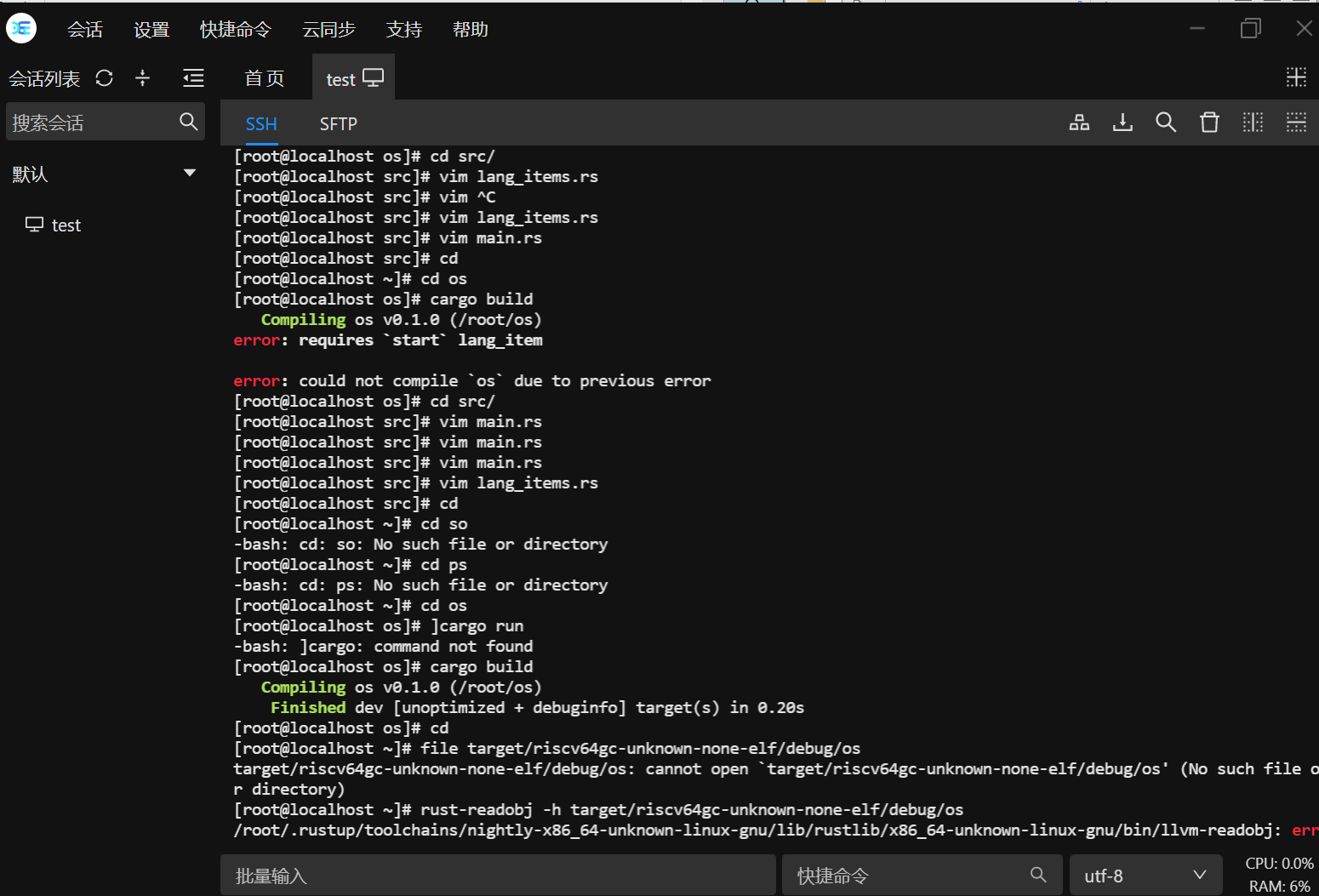


## 移除main函数

在 main.rs 的开头加入设置 #![no\_main] ，并将原来的 main 函数删除



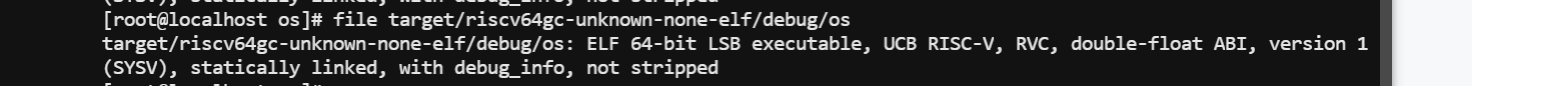
成功移除了标准库的依赖，并完成了构建裸机平台上的操作系统的第一步工作–通过编译器检查并生成执行码。



## 分析被移除标准库的程序

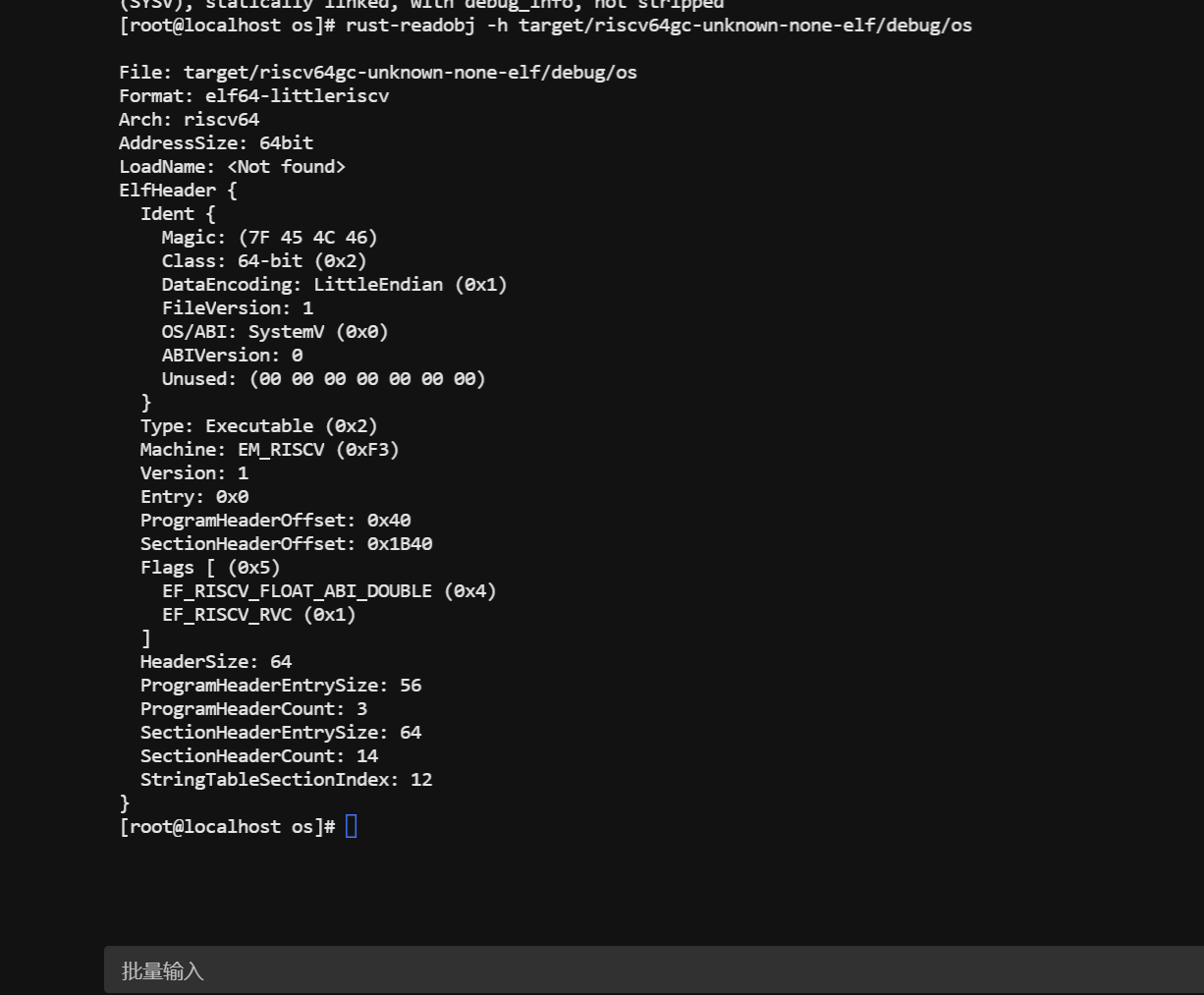
文件格式

file target/riscv64gc-unknown-none-elf/debug/os



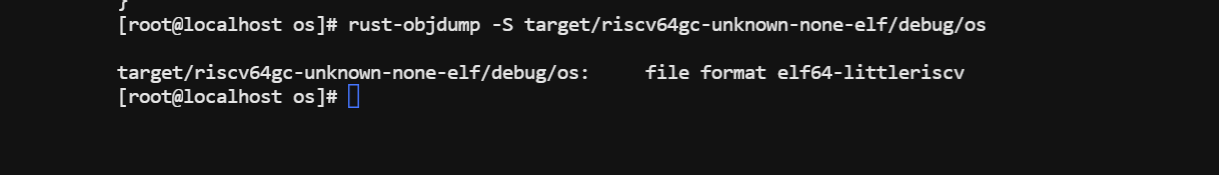
文件头信息

rust-readobj -h target/riscv64gc-unknown-none-elf/debug/os

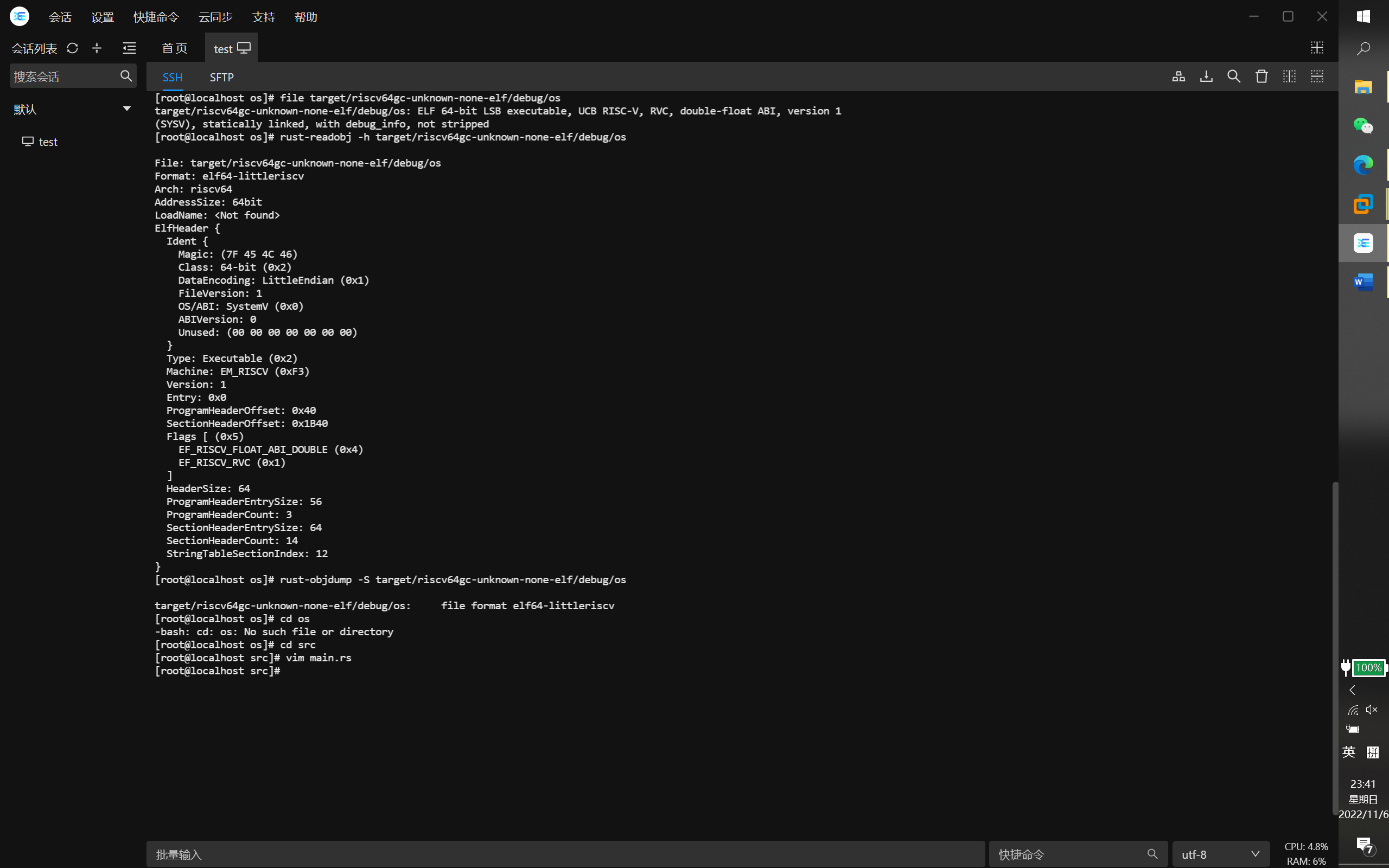


反汇编导出汇编程序

rust-objdump -S target/riscv64gc-unknown-none-elf/debug/os



（总执行过程）



# 3 源代码及注释

（说明：写出本次实验使用的命令或编写的源代码。请在下列表格中说明源代码的文件名和代码功能概述或者命令名和该命令的主要作用。要求针对核心功能代码进行注释。）

表 3-1 源代码及注释

|  |  |
| --- | --- |
| **文件名:** | **config（os/.cargo/config）** |
| **主要功能:** | **使 cargo 工具在 os 目录下默认会使用 riscv64gc-unknown-none-elf 作为目标平台** |
| **源代码:**  **[build]**  **target = "riscv64gc-unknown-none-elf"** | |

|  |  |
| --- | --- |
| **文件名:** | **main.rs（os/src/main.rs）** |
| **主要功能:** | **告诉 Rust 编译器不使用 Rust 标准库 std ，转而使用核心库 core（移除标准库依赖）** |
| **源代码:**  **#![no\_std]**  **fn main() {**  **// println!("Hello, world!");**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | **lang\_items.rs（os/src/lang\_items.rs）** |
| **主要功能:** | **实现panic函数，并通过 #[panic\_handler] 属性通知编译器用panic函数来对接 panic! 宏** |
| **源代码:**  **use core::panic::PanicInfo;**  **#[panic\_handler]**  **fn panic(\_info: &PanicInfo) -> ! {**  **loop {}**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | **main.rs（os/src/main.rs）** |
| **主要功能:** | **在main.rs中声明用到的子模块 lang\_items** |
| **源代码:**  **#![no\_std]**  **mod lang\_items;** | |

|  |  |
| --- | --- |
| **文件名:** | **main.rs（os/src/main.rs）** |
| **主要功能:** | **移除main函数** |
| **源代码:**  **#![no\_main]**  **#![no\_std]**  **mod lang\_items;**  **//fn main() {**  **// println!("Hello, world!")**  **//}** | |

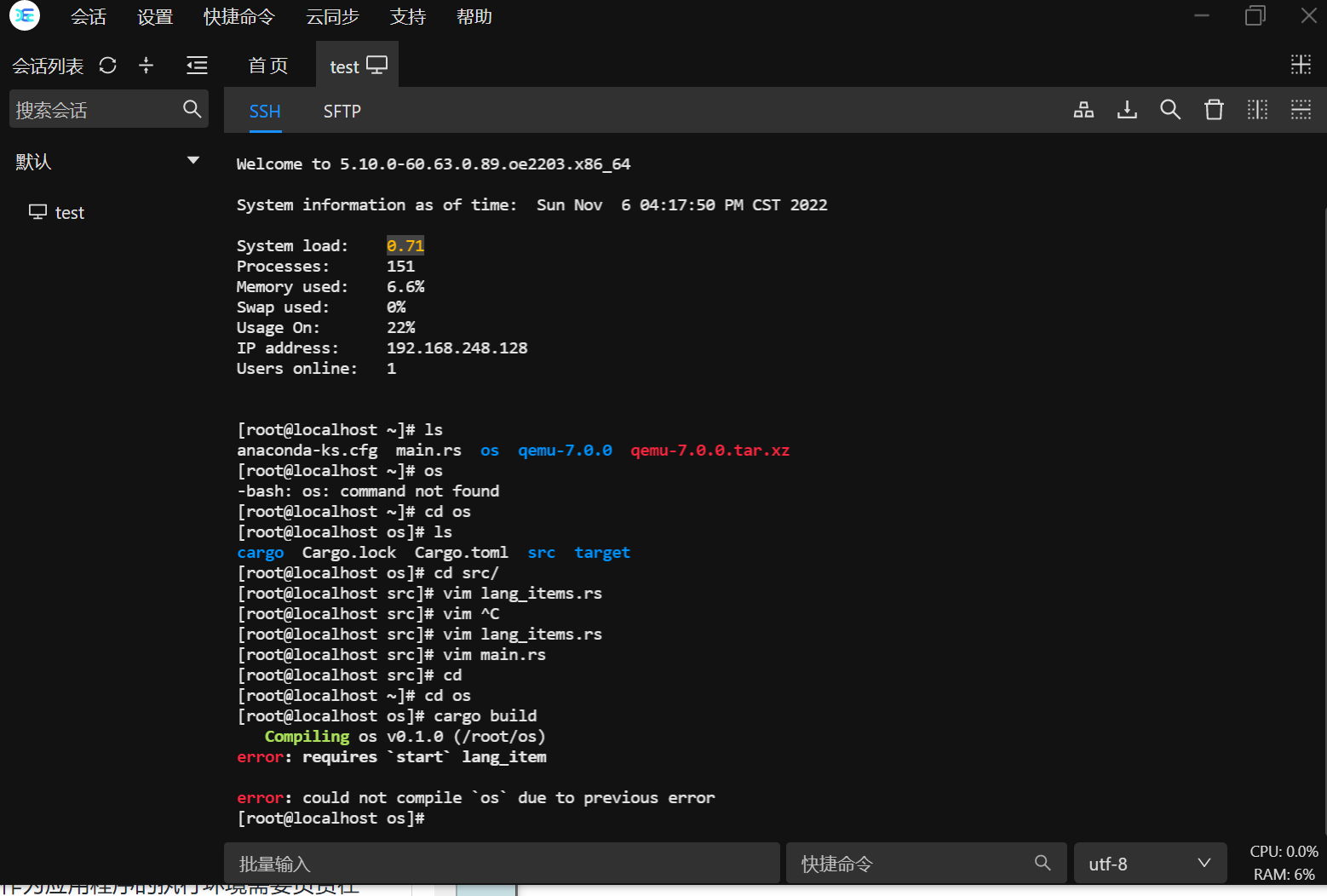
表 3-2 关键命令解释

|  |  |  |
| --- | --- | --- |
| **序号** | **命令名** | **主要作用** |
| **1** | **cargo new os --bin** | 创建一个 Rust 项目 |
| **2** | **cargo build** | 构建（build）工程 |
| **3** | **cargo run** | 运行（run）工程 |
| **4** | **rustc --version --verbose** | 查看程序的目标三元组 |
| **5** | **cargo run --target riscv64gc-unknown-none-elf** | 修改程序的目标平台，换成riscv64gc-unknown-none-elf |
| **6** | **rustup target add riscv64gc-unknown-none-elf** | 给 rustc 添加一个target : riscv64gc-unknown-none-elf |
| **7** | **file target/riscv64gc-unknown-none-elf/debug/os** | 查看文件格式 |
| **8** | **rust-readobj -h target/riscv64gc-unknown-none-elf/debug/os** | 查看文件头信息 |
| **9** | **rust-objdump -S target/riscv64gc-unknown-none-elf/debug/os** | 反汇编导出汇编程序 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

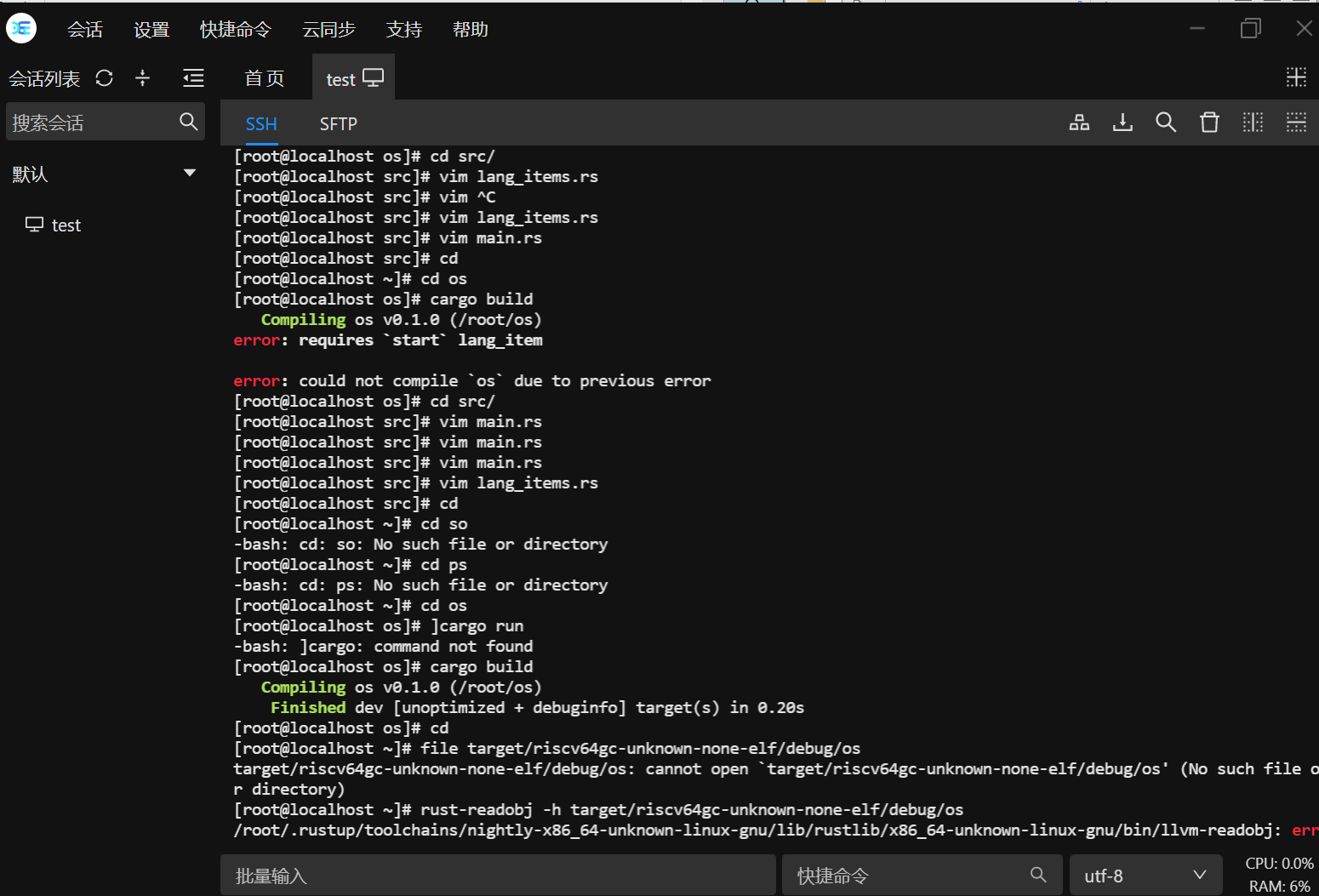
# 4 运行结果与分析

（说明：此部分请写出代码或命令运行结果及其分析。运行结果可以通过截图的方式提交,此外需要对截图进行分析与说明。）

移除标注库依赖，cargo build后报错，原因是缺少一个名为 start 的语义项



再移除main函数后，编译器就不需要完成初始化工作，编译成功，但实际上生成了一个空程序。



# 5 实验总结

（说明：此部分请写出针对本次实验的总结。也可以写出对于相同功能的不同实现方案，或是书写针对现有实验的改进方案等独创的且与本次实验有关的内容。）

1. 通过本次实验学习了一些基础的linux命令

如 cd ls vim mkdir touch等

1. **可以一次实现移除main函数和移除标准库依赖**
2. 在os/src 目录下**vim lang\_items.rs，**并修改为

use core::panic::PanicInfo;

#[panic\_handler]

fn panic(\_info: &PanicInfo) -> ! {

loop {}

}

1. 修改main.rs

#![no\_main]

#![no\_std]

mod lang\_items;

//fn main() {

// println!("Hello, world!")

//}

1. **cargo build**，编译成功