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

|  |
| --- |
| **姓名: 张鑫成** |
| **学号: 20271055** |

目录

[1 实验目的 3](#_Toc20101)

[2 实验过程设计 3](#_Toc146)

[3 源代码及注释 3](#_Toc32075)

[4 运行结果与分析 4](#_Toc10036)

[5 实验总结 7](#_Toc32554)

# 

# 1 实验目的

（说明：写出本次实验的主要实验目的与原因。）

从一个最简单的 Rust 应用程序入手，深入地挖掘它下面的多层执行环境为它的开发和运行提供了怎样的方便。

# 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。）

1. 使用cargo创建并运行程序
2. 修改程序目标平台
3. 移除标准库依赖
4. 移除println！（）宏、手动实现panic！宏
5. 移除main函数
6. 分析被移除标准库的程序

# 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 main.rs源代码及注释

|  |  |
| --- | --- |
| **文件名:** |  |
| **主要功能:** |  |
| **源代码:**   1. // os/src/main.rs 2. #![no\_main] 3. #![no\_std] 4. mod lang\_items; 5. // ... other code   **注释:**  注释1（第一行）：目标文件绝对路径  注释2（第二行）：被注释掉了、删除的main函数以及printf！宏 | |

表 3-2lang\_items.rs源代码及注释

|  |  |
| --- | --- |
| **文件名:** |  |
| **主要功能:** |  |
| **源代码:**   1. // os/src/lang\_items.rs 2. use core::panic::PanicInfo; 3. #[panic\_handler] 4. fn panic(\_info: &PanicInfo) -> ! { 5. loop {} 6. }   **注释:**  注释1（第一行）：目标文件绝对路径 | |

# 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。）

使用 Cargo 工具来创建一个 Rust 项目。

使用tree os查看Rust目录，使用vim main.rs 查看main函数内容。

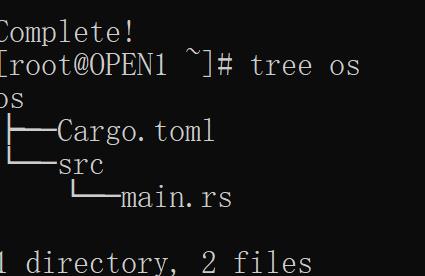


图4.1 tree os运行图

进入 os 项目根目录下，利用 Cargo run实现构建并运行项目：

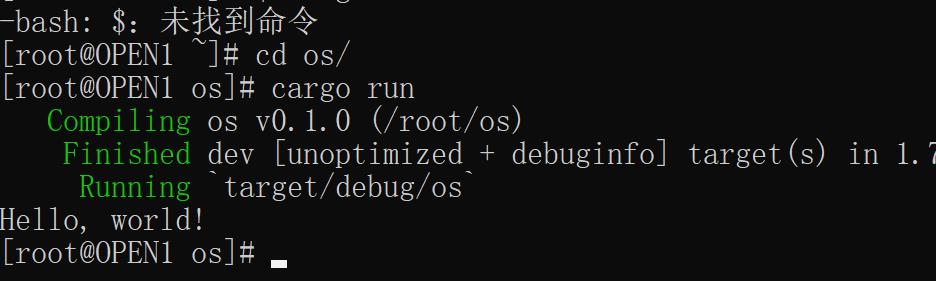


图4.2 cargo run运行图

编译器在编译、链接得到可执行文件时需要知道，程序要在哪个平台 (Platform) 上运行，目标三元组 (Target Triplet) 描述了目标平台的 CPU 指令集、操作系统类型和标准运行时库。我们使用rustc --version --verbose 查看程序的三元组：

对于本程序而言，目标三元组为：1.默认目标平台是x86\_64-unknown-linux-gnu，CPU架构x86\_64，CPU厂商unknown，操作系统是Linux，运行库是gnu libc

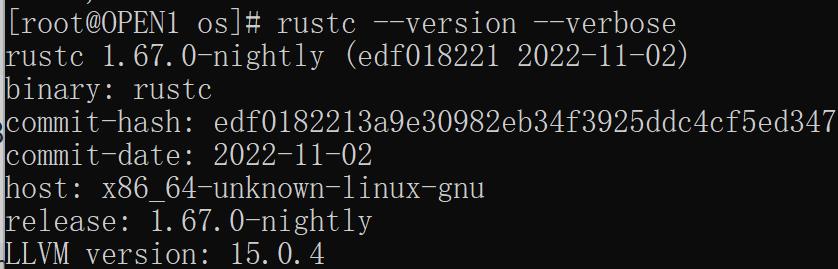


图4.3 version --verbose运行图

将程序的目标平台换成 riscv64gc-unknown-none-elf，使用cargo run --target riscv64gc-unknown-none-elf，显示报错。

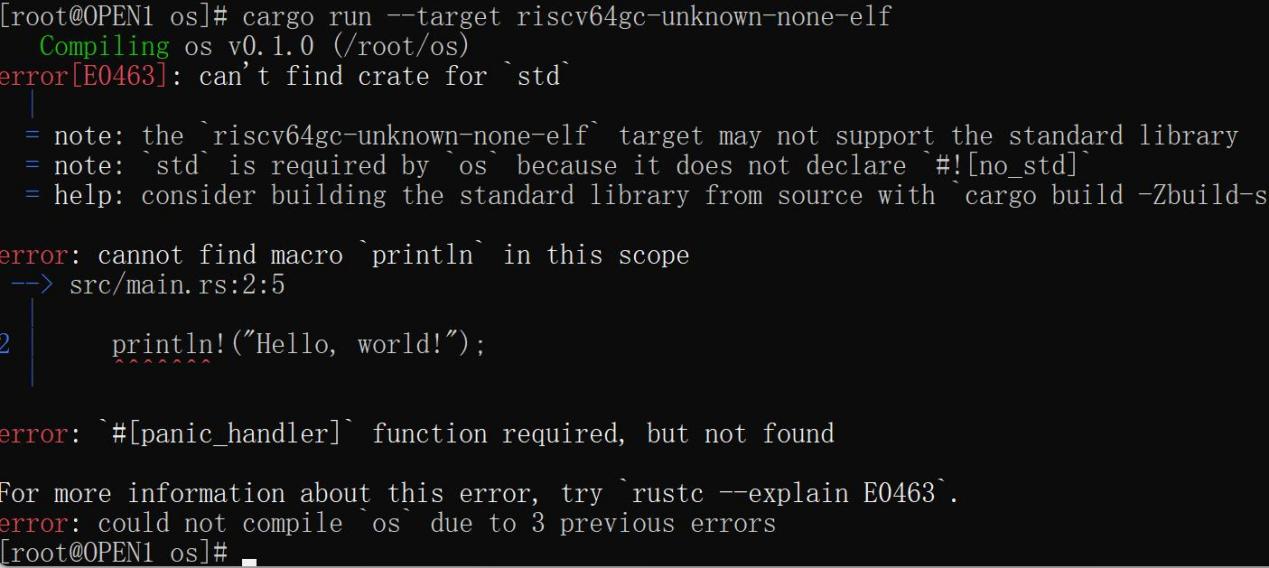


图4.4 修改目标平台运行图

报错的原因是目标平台上确实没有Rust标准库std，也不存在任何受OS支持的系统调用，这样的平台被我们称为裸机平台。幸运的是，除了 std 之外，Rust 还有一个不需要任何操作系统支持的核心库core,为了以裸机平台为目标编译程序，我们要将对标准库std的引用换成核心库core。

使用rustup target add riscv64gc-unknown-none-elf给rustc添加一个target为后续rustc编译器缺省生成RISC-V 64的目标代码服务。

然后在os目录下新建.cargo目录，并在这个目录下创建config文件，输入内容使cargo工具在os目录下默认会使用 riscv64gc-unknown-none-elf作为目标平台。

我们在main.rs的开头加上一行#![no\_std]来告诉 Rust 编译器不使用Rust 标准库std，转而使用核心库core（core库不需要操作系统的支持）。同时记得注释掉之前使用的println!()宏，重新编译执行。

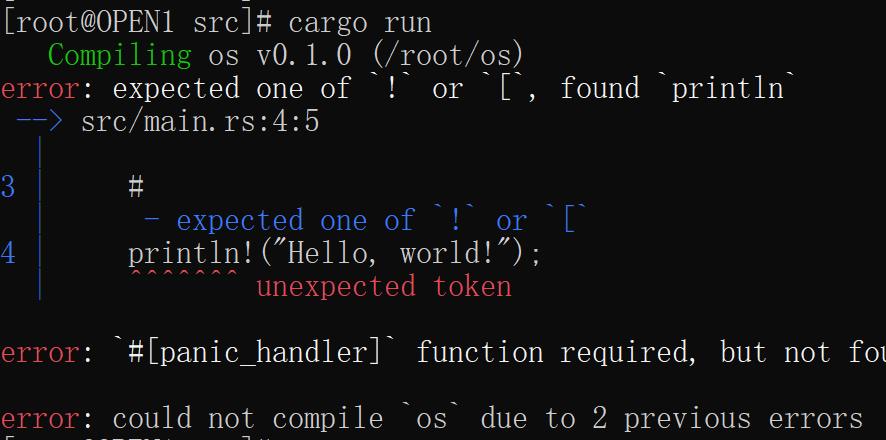


图4.5 重新运行图

Rust编译器在编译程序时，从安全性考虑，需要有panic!宏的具体实现。但是由于我们人工删掉了标准库依赖，导致这个宏失效了，因此我们需要手动实现一下这个宏。

我们创建一个新的子模块lang\_items.rs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。为了将该子模块添加到项目中，我们在main.rs的#![no\_std]的下方加上mod lang\_items;

在把panic\_handler配置在单独的文件os/src/lang\_items.rs后，需要在os/src/main.rs文件中添加#![no\_std]mod lang\_items;

此时运行报错：

**error: requires `start` lang\_item**

Rust语言标准库和三方库作为应用程序的执行环境需要负责在执行应用程序之前进行一些初始化工作，然后才跳转到应用程序的入口点。Rust语言标准库和三方库作为应用程序的执行环境需要负责在执行应用程序之前进行一些初始化工作，然后才跳转到应用程序的入口点。

我们在main.rs的开头加入设置#![no\_main]告诉编译器我们没有一般意义上的main函数，并将原来的main函数删除，压根不让编译器使用这项功能。在失去了main函数的情况下，编译器也就不需要完成所谓的初始化工作了。

运行cargo build：

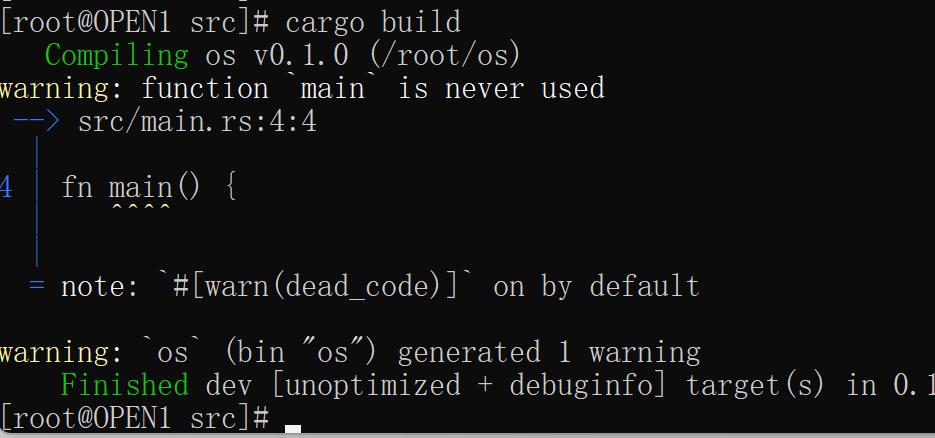


图4.6 cargo build完成运行图

通过各种工具来分析目前的程序：

分别是# 文件格式 # 文件头信息 # 反汇编导出汇编程序

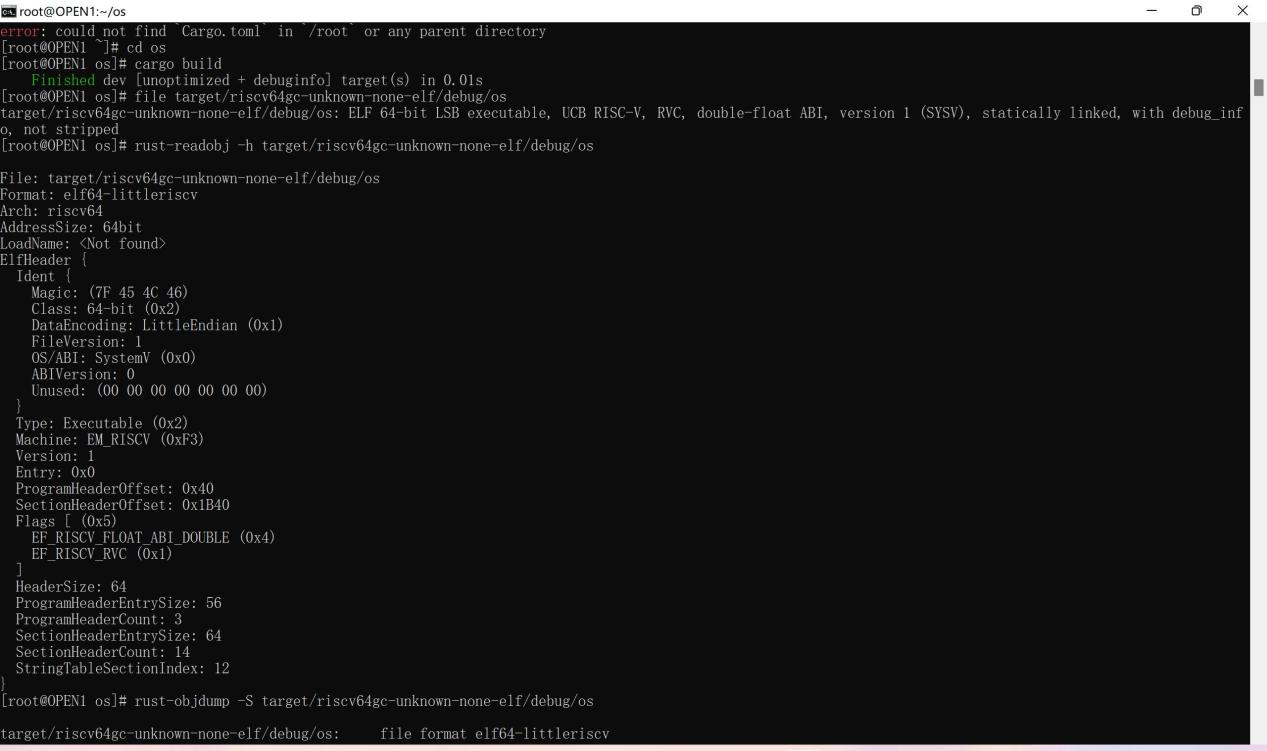


图4.7 分析被移除标准库的程序

# 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。）

在配置虚拟机、Rust之后第一次上手进行试验，学会通过命令行的方式进行文件的访问、修改以及目录的前进、后退。

通过尝试修改程序运行平台，进一步了解到：

1. 在默认平台中程序的运行使用了Rust标准库std，如果修改其他平台，我们同样可以通过调用Rust的一个不需要任何操作系统支持的核心库core来进行程序的编译。
2. 因为是在构建运行在裸机上的操作系统，所以不能再依赖标准库了。所以我们要尝试移除 println! 宏及其所在的标准库。
3. 同时得知Rust编译器在编译程序时，从安全性考虑，需要有 panic! 宏的具体实现。而Rust编译器在编译程序时，从安全性考虑，需要有 panic! 宏的具体实现。由于我们人工删掉了标准库依赖，导致这个宏失效了，因此我们需要手动实现一下这个宏。
4. 在报错缺少start的语义项之后，我们查询得知Rust语言标准库和三方库作为应用程序的执行环境需要负责在执行应用程序之前进行一些初始化工作，然后才跳转到应用程序的入口点。同时我们通过不让编译器使用这项功能的方式解决问题。

在平台修改的过程中，我们通过错误分析，进一步深入了解了操作系统于Rust配合下应用程序是如何执行的。标准库std、核心库core、std库函数的实现与调用、Rust从安全性考虑默认的panic！宏的具体出现、以及start的语义项的具体作用。进一步了解了上述几点是如何相互制约、限制、辅助的。