19271169-张东植-实验报告1

1. 创建Rust项目

1. New a Rust project with the following commands.

```
root@iZuf6hkk04t6lezxufufk6Z:~# cd os
```

2. Build the created Rust project.

```
root@iZuf6hkk04t6lezxufufk6Z:~# cargo build
```

3. Run the Rust project.

```
root@iZuf6hkk04t6lezxufufk6Z:~# cargo run
```

```
root@iZuf6hkk04t6lezxufufk6Z:~# cargo new os --bin
     Created binary (application) `os` package
root@iZuf6hkk04t6lezxufufk6Z:~# ls
os
root@iZuf6hkk04t6lezxufufk6Z:~# cd os
root@iZuf6hkk04t6lezxufufk6Z:~/os# cargo build
   Compiling os v0.1.0 (/root/os)
    Finished dev [unoptimized + debuginfo] target(s) in 1.90s
root@iZuf6hkk04t6lezxufufk6Z:~/os#
root@iZuf6hkk04t6lezxufufk6Z:~/os# cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/os`
Hello, world!
```

2. 移除标准库依赖

remove the dependencies of standard libs.

2.1 修改target

Modify target to riscv64. Add build config to the configuration file.

```
>_ 2. root@iZuf6hkk04t6lezxufufk6Z: ~/os/.cargo
[build]
target = "riscv64gc-unknown-none-elf"
```

2.2 修改main.rs

Modify file "main.rs". Delete main function and add some codes.

```
// main.rs
use core::panic::PanicInfo;

#[panic_handler]
fn panic(_info: &PanicInfo) -> ! {
    loop {}
}
```

```
>_ 2. root@iZuf6hkk04t6lezxufufk6Z: ~/os/src
#![no_std]
#![no_main]

use core::panic::PanicInfo;

#[panic_handler]
fn panic(_info: &PanicInfo) -> ! {
    loop {}
}
```

2.3 分析独立可执行程序

In the end, analyze the independent executable program.

We can find out that the binary generated by the compilation was found to be an empty program because the compiler could not find the entry function, so no subsequent code was generated.

```
root@iZuf6hkk04t6lezxufufk6Z:~# file target/riscv64gc-unknown-none-elf/debug/os
root@iZuf6hkk04t6lezxufufk6Z:~# rust-readobj -h target/riscv64gc-unknown-none-elf/debug/os
root@iZuf6hkk04t6lezxufufk6Z:~# rust-objdump -S target/riscv64gc-unknown-none-elf/debug/os
```

```
root@iZuf6hkk04t6lezxufufk6Z:~/os# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
Segmentation fault
root@iZuf6hkk04t6lezxufufk6Z:~/os# file target/riscv64gc-unknown-none-elf/debug/os
target/riscv64gc-unknown-none-elf/debug/os: ELF 64-bit LSB executable, UCB RISC-V, version 1 (SYSV), statically linked, with debug_in
fo, not stripped
root@iZuf6hkk04t6lezxufufk6Z:~/os# rust-readobj -h target/riscv64gc-unknown-none-elf/debug/os

File: target/riscv64gc-unknown-none-elf/debug/os
Format: elf64-littleriscv
Arch: riscv64
AddressSize: 64bit
LoadName: <Not found>
ElfHeader {
  Ident {
    Magic: (7F 45 4C 46)
    Class: 64-bit (0x2)
    DataEncoding: LittleEndian (0x1)
    FileVersion: 1
    OS/ABI: SystemV (0x0)
    ABIVersion: 0
    Unused: (00 00 00 00 00 00 00)
  }
  Type: Executable (0x2)
  Machine: EM_RISCV (0xF3)
  Version: 1
  Entry: 0x0
  ProgramHeaderOffset: 0x40
  SectionHeaderOffset: 0x10B0
  Flags [ (0x5)
    EF_RISCV_FLOAT_ABI_DOUBLE (0x4)
```

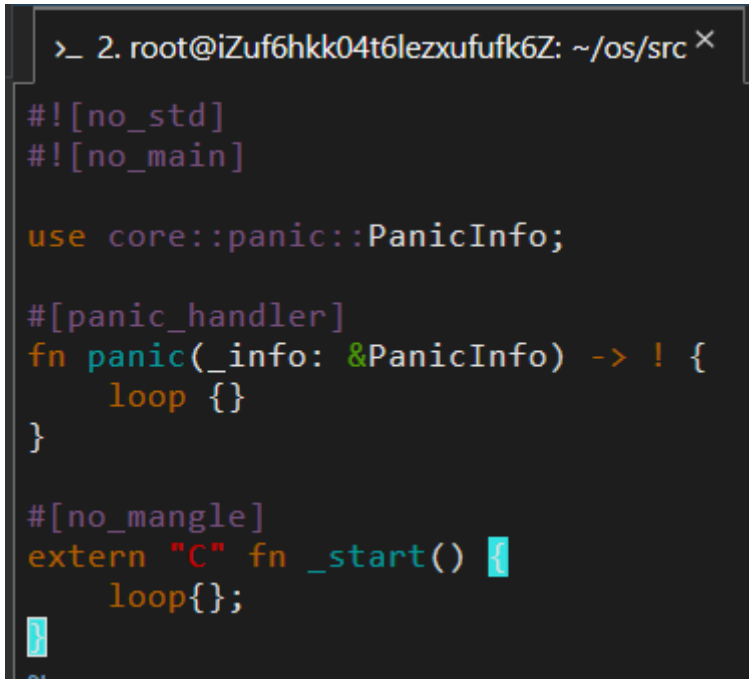
3. 用户态可执行的环境

3.1 增加入口函数

Add the following codes to "main.rs". Then run the program.

A Segmentation fault occurs. This is because the program currently lacks a proper exit mechanism.

```
#[no_mangle]
extern "C" fn _start() {
    loop{};
}
```




```
root@iZuf6hkk04t6lezxufufk6Z:~/os# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
Segmentation fault
```

3.2 实现退出机制

To implement the exit mechanism, we need to add the following codes to "main.rs".

```rust
#![feature(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);
}
```

```
root@iZuf6hkk04t6lezxufufk6Z:~/os# cargo build
   Compiling os v0.1.0 (/root/os)
    Finished dev [unoptimized + debuginfo] target(s) in 0.13s
root@iZuf6hkk04t6lezxufufk6Z:~/os# cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/riscv64gc-unknown-none-elf/debug/os`
target/riscv64gc-unknown-none-elf/debug/os: 1: Syntax error: word unexpected (expecting ")")
```

3.3 实现输出支持

1. First, encapsulate the SYSCALL_WRITE system call. This is a system call provided by the Linux operating system kernel with the ID SYSCALL_WRITE.

   ```rust
   const SYSCALL_WRITE: usize = 64;

   pub fn sys_write(fd: usize, buffer: &[u8]) -> isize {
     syscall(SYSCALL_WRITE, [fd, buffer.as_ptr() as usize, buffer.len()])
   }
   ```

2. Then, implement the data structure based on Write Trait, complete the write_str function required for Write Trait, and wrap it with the print function.

```
struct Stdout;

impl Write for Stdout {
    fn write_str(&mut self, s: &str) -> fmt::Result {
        sys_write(1, s.as_bytes());
        Ok(())
    }
}

pub fn print(args: fmt::Arguments) {
    Stdout.write_fmt(args).unwrap();
}
```

3. Finally, implement Rust language formatting macros based on print function.

```
use core::fmt::{self, Write};

#[macro_export]
macro_rules! print {
    ($fmt: literal $(, $($arg: tt)+)?) => {
        $crate::console::print(format_args!($fmt $(, $($arg)+)?));
    }
}

#[macro_export]
macro_rules! println {
    ($fmt: literal $(, $($arg: tt)+)?) => {
        print(format_args!(concat!($fmt, "\n") $(, $($arg)+)?));
    }
}
```

>_ 2. root@iZuf6hkk04t6lezxufufk6Z: ~/os/src

```
#![no_std]
#![no_main]
#![feature(asm)]

use core::panic::PanicInfo;
use core::fmt::{self, Write};

#[macro_export]
macro_rules! print {
    ($fmt: literal $(, $($arg: tt)+)?) => {
        $crate::console::print(format_args!($fmt $(, $($arg)+)?));
    }
}

#[macro_export]
macro_rules! println {
    ($fmt: literal $(, $($arg: tt)+)?) => {
        print(format_args!(concat!($fmt, "\n") $(, $($arg)+)?));
    }
}

#[panic_handler]
fn panic(_info: &PanicInfo) -> ! {
    loop {}
}

const SYSCALL_WRITE: usize = 64;
const SYSCALL_EXIT: usize = 93;

struct Stdout;
```

```rust
impl Write for Stdout {
    fn write_str(&mut self, s: &str) -> fmt::Result {
        sys_write(1, s.as_bytes());
        Ok(())
    }
}

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_write(fd: usize, buffer: &[u8]) -> isize {
  syscall(SYSCALL_WRITE, [fd, buffer.as_ptr() as usize, buffer.len()])
}

pub fn print(args: fmt::Arguments) {
    Stdout.write_fmt(args).unwrap();
}

pub fn sys_exit(xstate: i32) -> isize {
```

```
root@iZuf6hkk04t6lezxufufk6Z:~/os# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
Hello, world!
```

```
root@iZuf685rpkoqg9okbzi14iZ:~/os# cargo build
   Compiling os v0.1.0 (/root/os)
    Finished dev [unoptimized + debuginfo] target(s) in 0.30s
root@iZuf685rpkoqg9okbzi14iZ:~/os# cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/riscv64gc-unknown-none-elf/debug/os`
target/riscv64gc-unknown-none-elf/debug/os: 1: ELF[illegible]
                                                @8@@@@[illegible]88Q[illegible]td/rustc/
/src/char/methods.rs: not found
target/riscv64gc-unknown-none-elf/debug/os: 2: Syntax error: "(" unexpected
```