

操作系统 实验2

21301114 俞贤皓

环境: Arch Linux 6.5.3-arch1-1

1. 实验步骤

1.1 编译生成内核镜像

- 编译并生成binary文件

```
root@88a1fcca9270 /m/e/os# cargo build --release
   Compiling os v0.1.0 (/mnt/expt2/os)
   Finished release [optimized] target(s) in 0.54s
root@88a1fcca9270 /m/e/os# rust-objcopy --binary-architecture=riscv64 t
arget/riscv64gc-unknown-none-elf/release/os --strip-all -O binary targe
t/riscv64gc-unknown-none-elf/release/os.bin
```

- 载入 rustsbi.bin 文件

```
root@88a1fcca9270 /m/e/os# ls
Cargo.lock  Cargo.toml  rustsbi.bin  src  target
```

- 第一次运行此文件

```
root@88a1fcca9270 /m/e/os# qemu-system-riscv64 -machine virt -nographic
-bios ../bootloader/rustsbi.bin -device loader,file=target/riscv64gc-u
nknown-none-elf/release/os.bin,addr=0x80200000
[rustsbi] RustSBI version 0.2.0-alpha.6

RUSTSBI

[rustsbi] Implementation: RustSBI-QEMU Version 0.0.2
[rustsbi-dtb] Hart count: cluster0 with 1 cores
[rustsbi] misa: RV64ACDFIMSU
[rustsbi] mideleg: ssoft, stimer, sext (0x222)
[rustsbi] medeleg: ima, ia, bkpt, la, sa, uecall, ipage, lpage, spage (
0xb1ab)
[rustsbi] pmp0: 0x10000000 ..= 0x10001fff (rwx)
[rustsbi] pmp1: 0x80000000 ..= 0xffffffff (rwx)
[rustsbi] pmp2: 0x0 ..= 0xffffffffffffffff (---)
qemu-system-riscv64: clint: invalid write: 00000004
[rustsbi] enter supervisor 0x80200000
```

- docker 容器卡死

- 执行 `sudo docker restart os` 命令

- 分析可执行程序, 发现入口确实不为 `0x80200000`

-

```
Entry: 0x114BC
```

1.2 指定内存布局

- ```
root@88a1fcca9270 /m/expt2# vim os/.cargo/config
root@88a1fcca9270 /m/expt2# vim os/src/linker.ld
```

## 1.3 配置栈空间布局

- ```
root@88a1fcca9270 /m/expt2# vim os/src/entry.asm
root@88a1fcca9270 /m/expt2# vim os/src/main.rs
```

1.4 清空bss段

- ```
12 /* stack memory settings */
13 use core::arch::global_asm;
14
15 global_asm!(include_str!("entry.asm"));
16
17 #[no_mangle]
18 pub fn rust_main() → ! {
19 loop{};
20 }
21
22 /* clear bss segment */
23 fn clear_bss() {
24 extern "C" {
25 fn sbss();
26 fn ebss();
27 }
28 (sbss as usize..ebss as usize).for_each(|a| unsafe { (a as *mut
29 u8).write_volatile(0) });
30 }
```

## 1.5 实现裸机打印输出信息

-

```

root@88a1fcca9270 /m/e/os [101]# vim src/main.rs
root@88a1fcca9270 /m/e/os# vim src/console.rs
root@88a1fcca9270 /m/e/os# vim src/main.rs
root@88a1fcca9270 /m/e/os# vim src/lang_items.rs
root@88a1fcca9270 /m/e/os# vim src/main.rs
root@88a1fcca9270 /m/e/os# vim ~/.vim/init.vim
root@88a1fcca9270 /m/e/os# vim ~/.vim/init.vim
root@88a1fcca9270 /m/e/os# vim src/main.rs
root@88a1fcca9270 /m/e/os# cargo build --release
 Compiling os v0.1.0 (/mnt/expt2/os)
error: cannot find macro `asm` in this scope
 → src/sbi.rs:17:9
 |
17 | asm!("ecall",
 | ^^^
 = note: consider importing this macro:
 core::arch::asm

error[E0425]: cannot find function `sys_exit` in this scope
 → src/main.rs:58:5
 |
58 | sys_exit(9);
 | ^^^^^^^^ not found in this scope

For more information about this error, try `rustc --explain E0425`.
error: could not compile `os` due to 2 previous errors
root@88a1fcca9270 /m/e/os [101]# vim src/sbi.rs
root@88a1fcca9270 /m/e/os# vim src/main.rs
root@88a1fcca9270 /m/e/os# cargo build --release
 Compiling os v0.1.0 (/mnt/expt2/os)
 Finished release [optimized] target(s) in 0.27s

```

## 1.6 重新编译并运行

-





```

root@88a1fcca9270 /m/e/os# make run
 Finished release [optimized] target(s) in 0.01s
[rustsbi] RustSBI version 0.2.0-alpha.6

┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐
└───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘

[rustsbi] Implementation: RustSBI-QEMU Version 0.0.2
[rustsbi-dtb] Hart count: cluster0 with 1 cores
[rustsbi] misa: RV64ACDFIMSU
[rustsbi] mideleg: ssoft, stimer, sext (0x222)
[rustsbi] medeleg: ima, ia, bkpt, la, sa, uecall, ipage, lpage, spage (
0xb1ab)
[rustsbi] pmp0: 0x100000000 ..= 0x10001fff (rwx)
[rustsbi] pmp1: 0x800000000 ..= 0x8fffffff (rwx)
[rustsbi] pmp2: 0x0 ..= 0xffffffffffffffff (---)
qemu-system-riscv64: clint: invalid write: 00000004
[rustsbi] enter supervisor 0x80200000
Hello, world!
.text [0x80200000, 0x80202000)
.rodata [0x80202000, 0x80203000)
.data [0x80203000, 0x80204000)
boot_stack [0x80204000, 0x80214000)
.bss [0x80214000, 0x80214000)
Hello, world!
Panicked at src/main.rs:49 Shutdown machine!

```

## 2. 思考问题

### 2.1

- 问题: 分析 `linker.ld` 和 `entry.asm` 所完成的功能
- 回答
  - `linker.ld`
    - 如文档所述, `linker.ld` 在这次实验中的功能是 **指定内存布局**
    - 第一行 `OUTPUT_ARCH(riscv)` 指定了程序的架构
    - 第二行 `ENTRY(_start)` 指定了程序的入口
    - 第三行指定了程序的入口地址为 `0x80200000`
    - 之后部分定义了程序各个段的布局 (按顺序, 分别定义了 `text` 代码段、`rodata` 只读数据段、`data` 数据段、`bss` 未初始化的数据段), 其中 `ALIGN(4K)` 表示新的段距离上一个段的开头偏移了4K字节
    - 通过 `rust-readobj` 分析, 可以发现程序入口确实变为了 `0x80200000`

```

Version: 1
Entry: 0x80200000
ProgramHeaderOffset: 0x40

```

- `entry.asm`
  - 如文档所述, `entry.asm` 在这次实验中的功能是 **配置栈空间布局**
  - `.section .text.entry` 定义了一个代码段, 表示程序入口
  - `.globl _start` 定义了一个全局的标记 `_start` 和 `linker.ld` 的 `ENTRY(_start)` 相对应, 表示程序入口。而 `call rust_main` 表示在 `_start` 这个函数中调用 `rust_main`, 这应该和 `main.rs` 里的 `rust_main` 相对应。
  - `.section .bss.stack` 定义了一个bss段, 表示栈空间
  - `.globl boot_stack` 和接下来的代码, 定义了一个长度为 `4096 * 16` 字节的栈空间

## 2.2

- 问题: 分析 `sbi` 模块和 `lang_items` 模块所完成的功能
- 回答
  - `sbi` 模块
    - 本模块和实验1中的 `system call` 部分代码功能类似, 都通过 **内联汇编** 语法, 与硬件进行交互, 实现了 **系统调用**。 `sbi_call` 和 `syscall` 函数, 大体类似, 只有参数和返回值的区别。
    - `console_putchar` 等3个函数, 对 `sbi_call` 函数进行封装, 实现了3种不同的系统调用, 并且以 **pub** 的方式抛出口, 供外部调用。( `sbi_call` 为私有函数, 其他模块无法调用)
    - `console` 模块对 `console_putchar` 系统调用进行封装, 实现了 rust 风格的输出函数与输出宏
  - `lang_items` 模块
    - 本模块名为 `lang_items`, 所以本模块应该实现一些有关高级语言的特性。但在本次实验中, `lang_items` 模块只实现了 **异常** 特性, 所以预测在之后的实验中, 本模块的功能会被拓展。
    - `lang_items` 实现了 **异常处理函数panic**
    - 在实验1的 `main.rs` 中也有类似的实现
    - 不同的是, 本模块的异常处理函数实现了异常信息输出, 并且新增了 `shutdown()` 系统调用。当发生异常时, 操作系统会打印出异常信息, 并且关机

## 2.3

- 问题: 可选: 如果将 `rustsbi.bin` 换成最新版本的会造成代码无法运行, 分析原因并给出解决方法。
- 操作流程
  - 新建 `os-ex` 与 `bootloader-ex` 文件夹, 并将原代码与最新版本的 `rustsbi-qemu.bin` 置于对应目录下
    - 我使用了2023-10-27最新pre-released `rustsbi-qemu`
    -

```
YXH_XianYu ~/b/O/G/expt2 (main)> tree -L 2
.
├── bootloader
│ └── rustsbi.bin
├── bootloader-ex
│ └── rustsbi-qemu.bin
├── os
│ ├── Cargo.lock
│ ├── Cargo.toml
│ ├── Makefile
│ ├── src
│ └── target
└── os-ex
 ├── Cargo.lock
 ├── Cargo.toml
 ├── Makefile
 ├── src
 └── target
```

◦ 进入 `os-ex`，将引导程序修改为 `bootloader-ex` 下的对应程序

- 修改 `Makefile:8~9` 为

```
SBI ?= rustsbi-qemu
BOOTLOADER := ../bootloader-ex/${SBI}.bin
```

◦ 测试

- `make`、`make build` 均可正常执行
- `make run` 出现了不正常的结果，在输出 `Hello, world!` 和 `Panicked ... Shutdown machine!` 之后，程序又输出了大量 `src/sbi.rs:40 It should shutdown!` 异常和乱码（应该是无限递归调用panic函数，导致栈空间溢出），并卡死

```
Panicked at src/sbi.rs:40 It should shutdown!
Panicked at src/sbi.rs:40 It should shutdown!
Panicked at src/sbi.rs:40 It should shutdown!
Panicked at src/sbi.rs:40 It should shutdown!
Panicked at src/sbi.rs:40 It should shutdown!
Panicked at src/sbi.rs: 8% h# 5556575863646566% X?! @! " & l %
% % #
X?!(0& `P# `%' 0& à?!
```

◦ 解决







```

// pub fn console_putchar(c: usize) {
// #[allow(deprecated)]
// sbi_rt::legacy::console_putchar(c);
// }
//
// pub fn console_getchar() -> usize {
// #[allow(deprecated)]
// sbi_rt::legacy::console_getchar()
// }
//
// pub fn shutdown() -> ! {
// use sbi_rt::{system_reset, NoReason, Shutdown};
// system_reset(Shutdown, NoReason);
// panic!("It should shutdown!");
// }

use core::arch::asm;

const SBI_SET_TIMER: usize = 0;
const SBI_CONSOLE_PUTCHAR: usize = 1;
const SBI_CONSOLE_GETCHAR: usize = 2;
const SBI_CLEAR_IPI: usize = 3;
const SBI_SEND_IPI: usize = 4;
const SBI_REMOTE_FENCE_I: usize = 5;
const SBI_REMOTE_SFENCE_VMA: usize = 6;
const SBI_REMOTE_SFENCE_VMA_ASID: usize = 7;
const SBI_SHUTDOWN: usize = 8;

// const SBI_STOP_EXTENSION: usize = 0x48534D;
// const SBI_STOP_FUNCTION: usize = 1;
const SBI_EID_SRST: usize = 0x53525354;
const SBI_SYSTEM_RESET: usize = 0;

#[inline(always)]
fn sbi_call(eid: usize, fid: usize, arg0: usize, arg1: usize, arg2: usize) -
> usize {
 let mut ret;
 unsafe {
 asm!("ecall",
 in("x10") arg0,
 in("x11") arg1,
 in("x12") arg2,
 in("x16") fid,
 in("x17") eid,
 lateout("x10") ret
);
 }
 ret
}

pub fn console_putchar(c: usize) {

```

```
sbi_call(SBI_CONSOLE_PUTCHAR, 0, c, 0, 0);
}

pub fn console_getchar() -> usize {
 sbi_call(SBI_CONSOLE_GETCHAR, 0, 0, 0, 0)
}

pub fn shutdown_deprecated() -> ! {
 sbi_call(SBI_SHUTDOWN, 0, 0, 0, 0);
 panic!("It should shutdown! (deprecated shutdown function)");
}

pub fn shutdown() -> ! {
 sbi_call(SBI_EID_SRST, SBI_SYSTEM_RESET, 0, 0, 0);
 panic!("It should shutdown!");
}
```

### 3. Git提交截图

- [仓库链接](#)
- 



### 4. 其他说明

- 实验很有意思