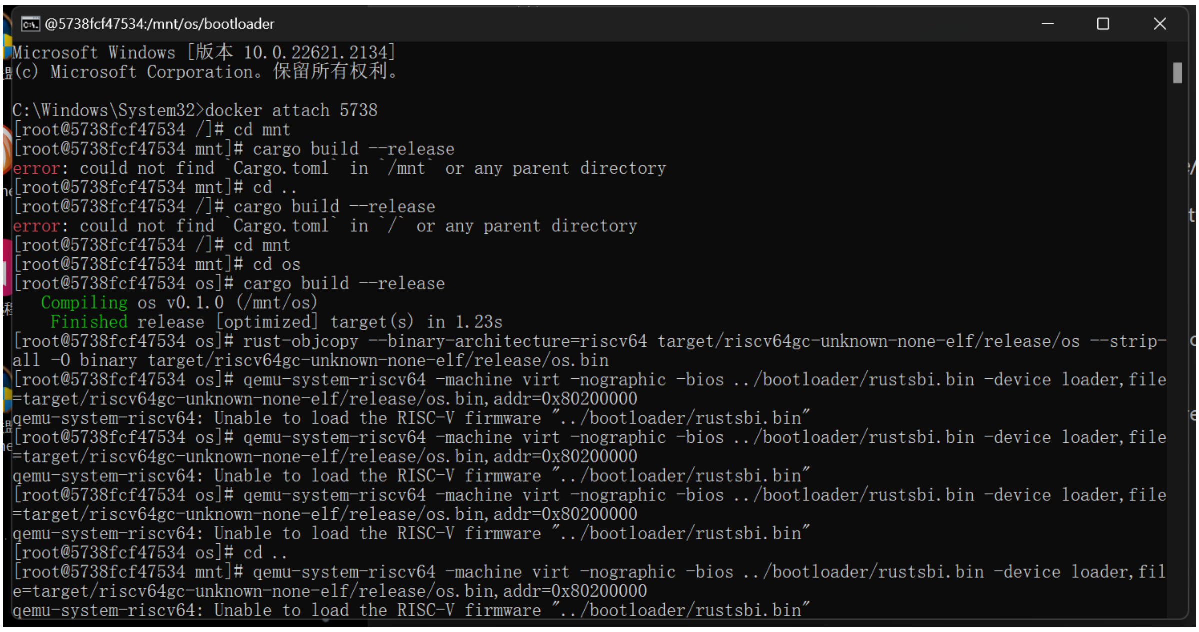
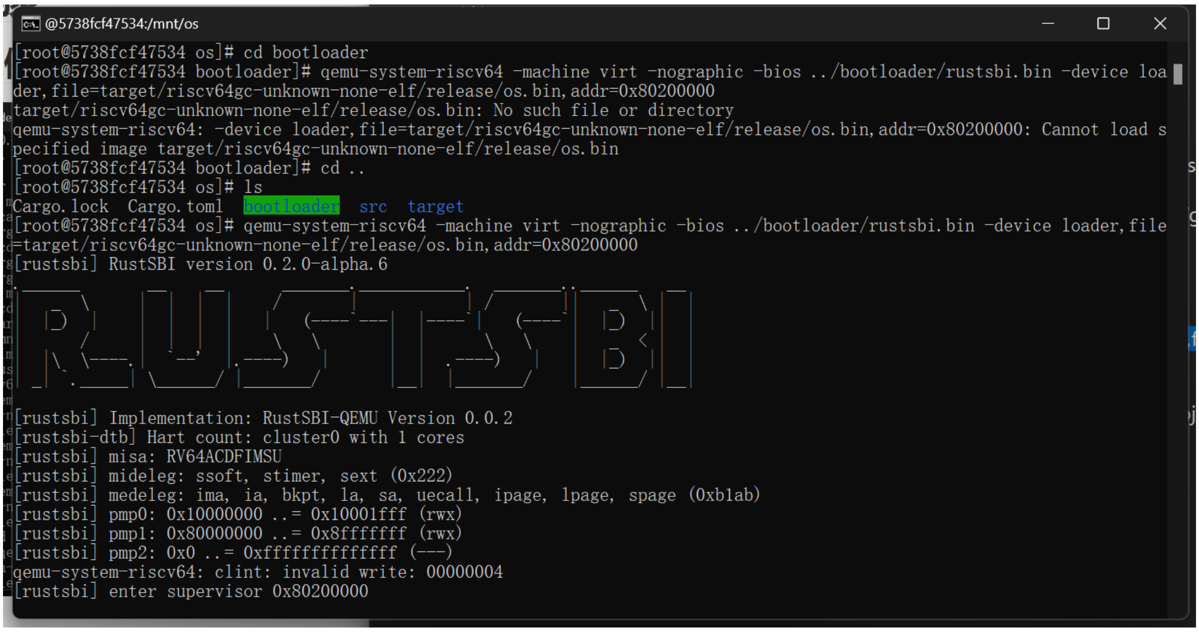
> 操作系统lab2 21301117周书扬
>
> 本实验的主要目的是实现裸机上的执行环境以及一个最小化的操作系统内核。

一. 实验步骤

1. 编译内核镜像&将编译生成的ELF执行文件转成binary文件：

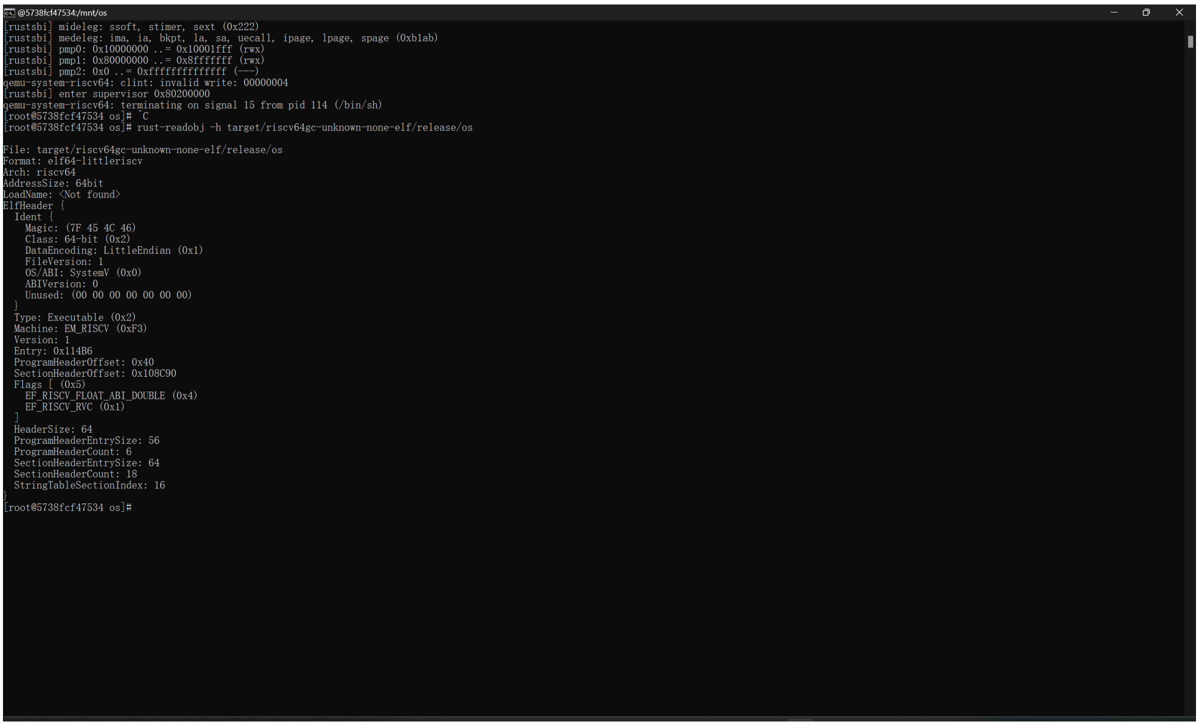


2. 加载运行生成的二进制文件，发现程序死循环



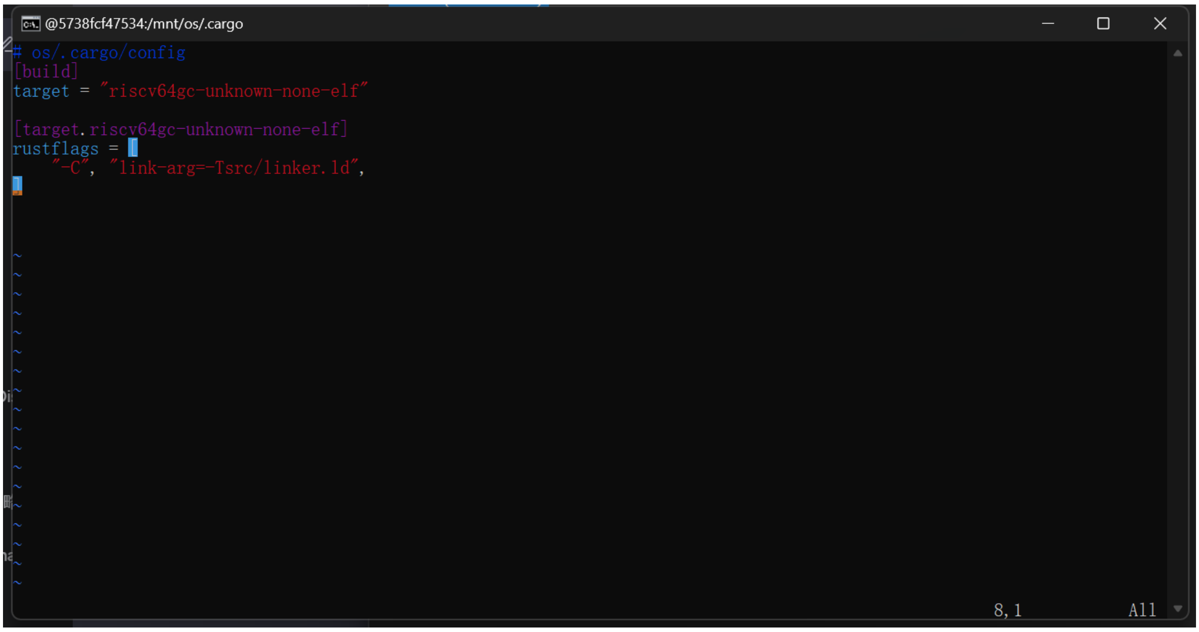
3. 使用rust-readobj分析

发现入口地址不是0x80200000而是0x114B6



4. 指定内存布局

4.1 修改cargo配置文件



4.2 增加链接脚本文件

```
OUTPUT_ARCH(riscv)
ENTRY(_start)
BASE_ADDRESS = 0x80200000;

SECTIONS
{
    . = BASE_ADDRESS;
    skernel = .;

    stext = .;
    .text : {
        *(.text.entry)
        *(.text .text.*)
    }

    . = ALIGN(4K);
    etext = .;
    srodata = .;
    .rodata : {
        *(.rodata .rodata.*)
        *(.srodata .srodata.*)
    }

    . = ALIGN(4K);
    erodata = .;
    sdata = .;
    .data : {
        *(.data .data.*)
        *(.sdata .sdata.*)
-- INSERT --                                                    1,2           Top
```

5. 配置栈空间布局

- 增加entry.asm

```
    .section .text.entry
    .globl _start
_start:
    la sp, boot_stack_top
    call rust_main

    .section .bss.stack
    .globl boot_stack
boot_stack:
    .space 4096 * 16
    .globl boot_stack_top
boot_stack_tiop:
-- INSERT --                                                    1,5           All
```

- 在main.rs中增加汇编代码，声明应用入口

```
@5738fcf47534:/mnt/os/src

#[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)+)?));
    }
}

use core::arch::global_asm;

global_asm!(include_str!("entry.asm"));

#[no_mangle]
pub fn rust_main() -> ! {
        loop{};
}

~
:wq
```

6. 清空bss段

```
@5738fcf47534:/mnt/os/src
    }
}

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

use core::arch::global_asm;

global_asm!(include_str!("entry.asm"));

#[no_mangle]
pub fn rust_main() -> ! {
        loop{};
}

fn clear_bss() {
    extern "C" {
        fn sbss();
        fn ebss();
    }
    (sbss as usize..ebss as usize).for_each(|a| unsafe { (a as *mut u8).write_volatile(0) });
}

"main.rs" 93L, 1698B                                                    85,6          Bot
```

7. 实现裸机打印输出信息

- 将系统调用改为sbi调用

```
@5738fcf47534:/mnt/os/src
const SBI_SHUTDOWN: usize = 8;

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

pub fn console_putchar(c: usize) {
    sbi_call(SBI_CONSOLE_PUTCHAR, c, 0, 0);
}

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

pub fn shutdown() -> ! {
    sbi_call(SBI_SHUTDOWN, 0, 0, 0);
    panic!("It should shutdown!");
"sbi.rs" 42L, 946B                                                    42,1          Bot
```

- 实现裸机上的`print`函数

```
@5738fcf47534:/mnt/os/src
struct Stdout;
impl Write for Stdout {
    fn write_str(&mut self, s: &str) -> fmt::Result {
        for c in s.chars() {
            console_putchar(c as usize);
        }
        Ok(())
    }
}
pub fn print(args: fmt::Arguments) {
    Stdout.write_fmt(args).unwrap();
}
#[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)+)?) => {
        $crate::console::print(format_args!(concat!($fmt, "\n") $(, $($arg)+)?));
    }
}
~
~
~
~
-- INSERT (paste) --
```

8. 给异常处理增加输出信息

- 实现`os/src/lang_items.rs`

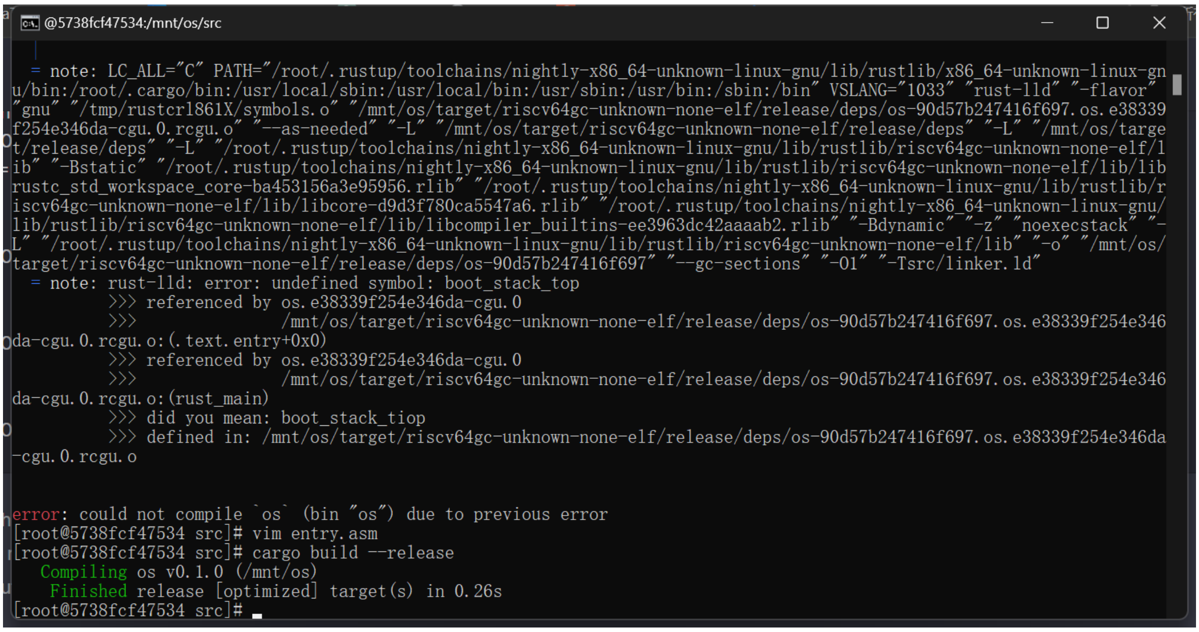
```
@5738fcf47534:/mnt/os/src
use crate::sbi::shutdown;
use core::panic::PanicInfo;

#[panic_handler]
fn panic(info: &PanicInfo) -> ! {
    if let Some(location) = info.location() {
        println!(
            "Panicked at {}:{} {}",
            location.file(),
            location.line(),
            info.message().unwrap()
        );
    } else {
        println!("Panicked: {}", info.message().unwrap());
    }
    shutdown()
}
"lang_items.rs" 17L, 406B                                1,2           All
```

9. 修改main.rs

```
@5738fcf47534:/mnt/os/src
#![no_std]
#![no_main]
#![feature(panic_info_message)]
#[macro_use]

mod console;
mod lang_items;
mod sbi;

use core::arch::global_asm;

global_asm!(include_str!("entry.asm"));
fn clear_bss() {
    extern "C" {
        fn sbss();
        fn ebss();
    }
    (sbss as usize..ebss as usize).for_each(|a| unsafe { (a as *mut u8).write_volatile(0) });
}

#[no_mangle]
pub fn rust_main() -> ! {
    extern "C" {
        fn stext();
        fn etext();
        fn srodata();
        fn erodata();
        fn sdata();
        fn edata();
"main.rs" 47L, 1138B                                     1,1           Top
```

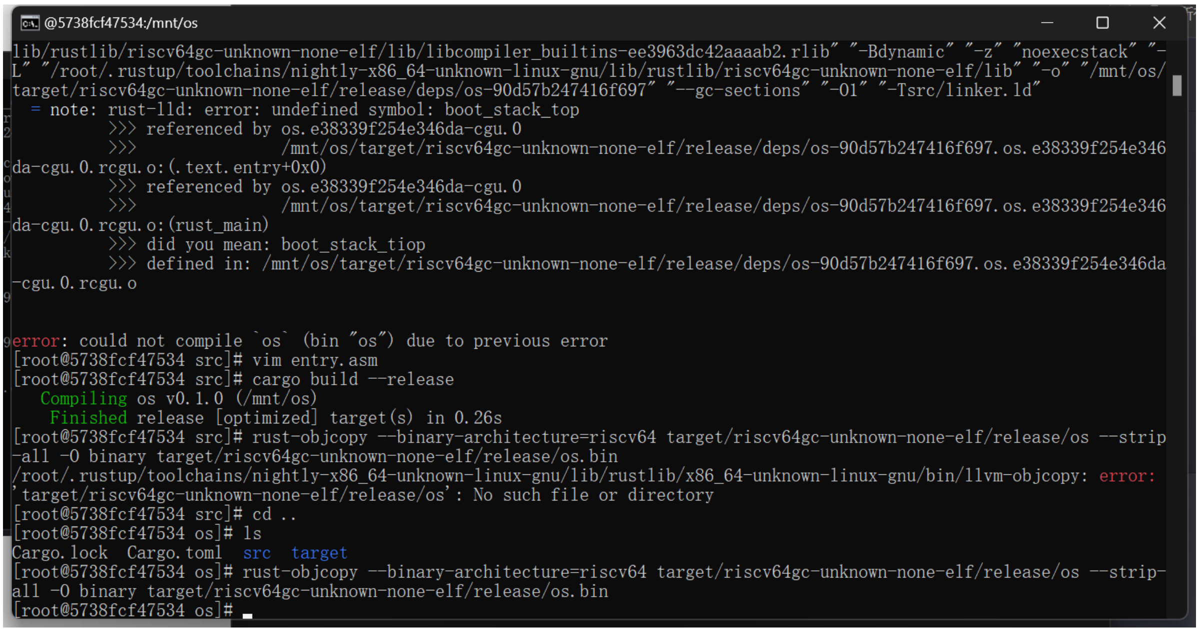
10. 重新编译以及生成二进制文件

- 编译



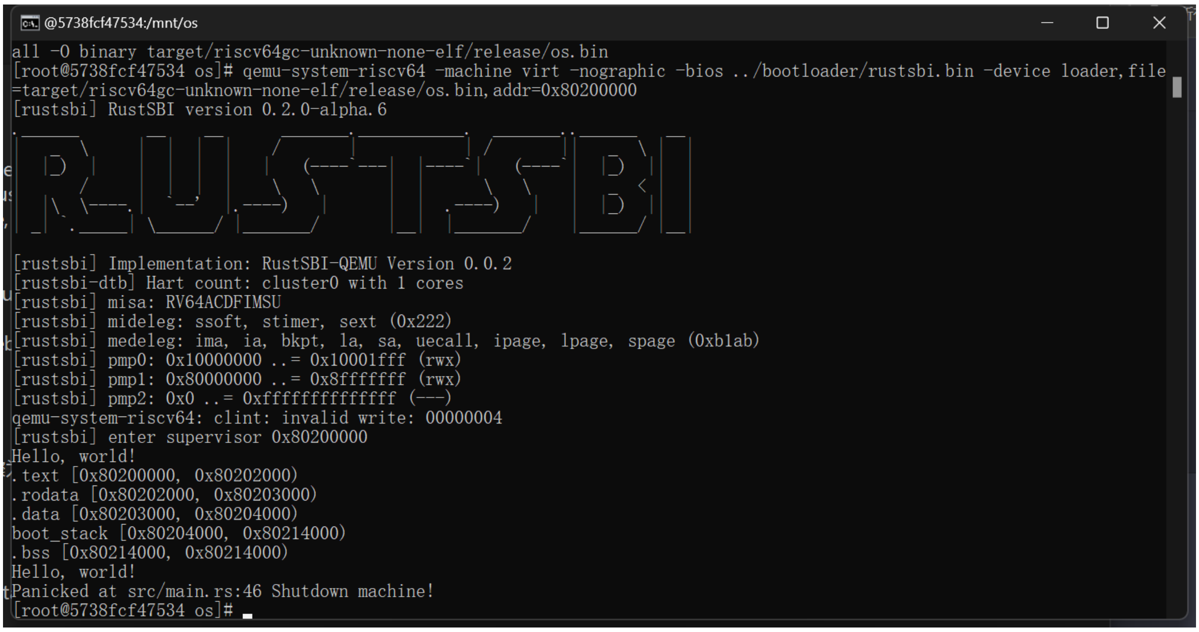
```
@5738fcf47534:/mnt/os/src
  = note: LC_ALL="C" PATH="/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/x86_64-unknown-linux-gnu/bin:/root/.cargo/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin" VSLANG="1033" "rust-lld" "-flavor" "gnu" "/tmp/rustcrl861X/symbols.o" "/mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o" "--as-needed" "-L" "/mnt/os/target/riscv64gc-unknown-none-elf/release/deps" "-L" "/mnt/os/target/release/deps" "-L" "/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/riscv64gc-unknown-none-elf/lib" "-Bstatic" "/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/riscv64gc-unknown-none-elf/lib/librustc_std_workspace_core-ba453156a3e95956.rlib" "/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/riscv64gc-unknown-none-elf/lib/libcore-d9d3f780ca5547a6.rlib" "/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/riscv64gc-unknown-none-elf/lib/libcompiler_builtins-ee3963dc42aaaab2.rlib" "-Bdynamic" "-z" "noexecstack" "-L" "/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/riscv64gc-unknown-none-elf/lib" "-o" "/mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697" "--gc-sections" "-O1" "-Tsrc/linker.ld"
  = note: rust-lld: error: undefined symbol: boot_stack_top
          >>> referenced by os.e38339f254e346da-cgu.0
          >>>               /mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o:(.text.entry+0x0)
          >>> referenced by os.e38339f254e346da-cgu.0
          >>>               /mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o:(rust_main)
          >>> did you mean: boot_stack_tiop
          >>> defined in: /mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o


error: could not compile `os` (bin "os") due to previous error
[root@5738fcf47534 src]# vim entry.asm
[root@5738fcf47534 src]# cargo build --release
   Compiling os v0.1.0 (/mnt/os)
    Finished release [optimized] target(s) in 0.26s
[root@5738fcf47534 src]#
```

- 生成二进制文件



```
@5738fcf47534:/mnt/os
lib/rustlib/riscv64gc-unknown-none-elf/lib/libcompiler_builtins-ee3963dc42aaaab2.rlib" "-Bdynamic" "-z" "noexecstack" "-L" "/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/riscv64gc-unknown-none-elf/lib" "-o" "/mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697" "--gc-sections" "-O1" "-Tsrc/linker.ld"
  = note: rust-lld: error: undefined symbol: boot_stack_top
          >>> referenced by os.e38339f254e346da-cgu.0
          >>>               /mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o:(.text.entry+0x0)
          >>> referenced by os.e38339f254e346da-cgu.0
          >>>               /mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o:(rust_main)
          >>> did you mean: boot_stack_tiop
          >>> defined in: /mnt/os/target/riscv64gc-unknown-none-elf/release/deps/os-90d57b247416f697.os.e38339f254e346da-cgu.0.rcgu.o

error: could not compile `os` (bin "os") due to previous error
[root@5738fcf47534 src]# vim entry.asm
[root@5738fcf47534 src]# cargo build --release
   Compiling os v0.1.0 (/mnt/os)
    Finished release [optimized] target(s) in 0.26s
[root@5738fcf47534 src]# rust-objcopy --binary-architecture=riscv64 target/riscv64gc-unknown-none-elf/release/os --strip-all -O binary target/riscv64gc-unknown-none-elf/release/os.bin
/root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/x86_64-unknown-linux-gnu/bin/llvm-objcopy: error: 'target/riscv64gc-unknown-none-elf/release/os': No such file or directory
[root@5738fcf47534 src]# cd ..
[root@5738fcf47534 os]# ls
Cargo.lock  Cargo.toml  src  target
[root@5738fcf47534 os]# rust-objcopy --binary-architecture=riscv64 target/riscv64gc-unknown-none-elf/release/os --strip-all -O binary target/riscv64gc-unknown-none-elf/release/os.bin
[root@5738fcf47534 os]#
```

- 运行



```
@5738fcf47534:/mnt/os
all -O binary target/riscv64gc-unknown-none-elf/release/os.bin
[root@5738fcf47534 os]# qemu-system-riscv64 -machine virt -nographic -bios ../bootloader/rustsbi.bin -device loader,file=target/riscv64gc-unknown-none-elf/release/os.bin,addr=0x80200000
[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: 0x10000000 ..= 0x10001fff (rwx)
[rustsbi] pmp1: 0x80000000 ..= 0x8fffffff (rwx)
[rustsbi] pmp2: 0x0 ..= 0xffffffffffffff (---)
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:46 Shutdown machine!
[root@5738fcf47534 os]#
```

增加Makefile文件

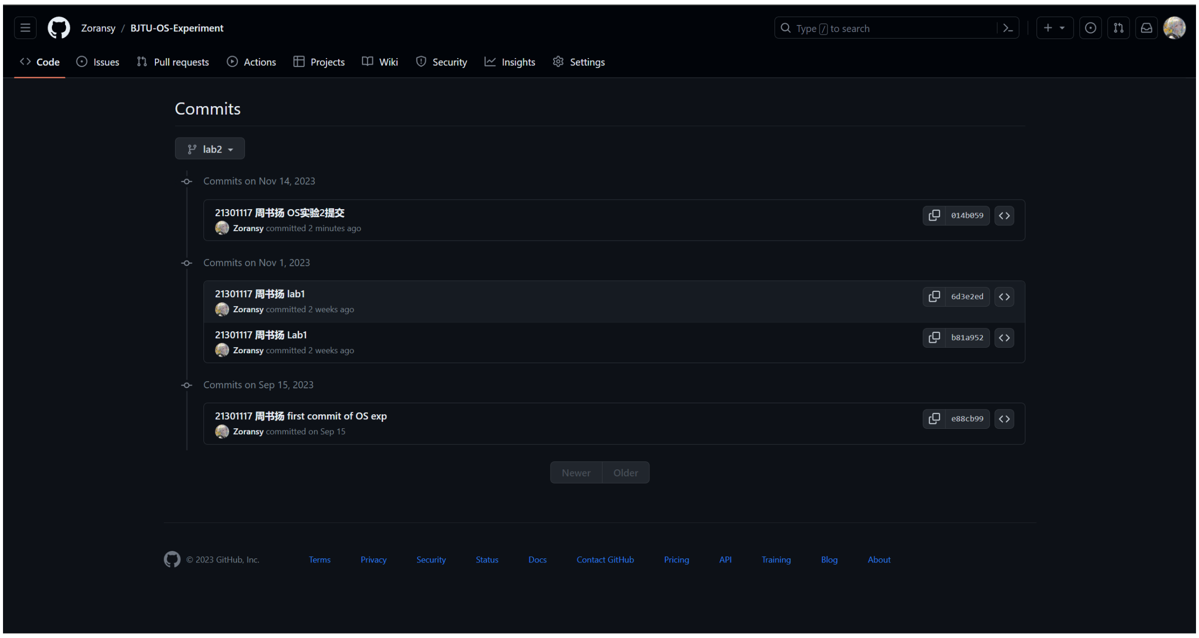
- Makefile文件的缩进需要自己手动规范，使用tab而不是空格

```
@5738fcf47534:/mnt/os

clean:
	@cargo clean

disasm: kernel
	@$(OBJDUMP) $(DISASM) $(KERNEL_ELF) | less

disasm-vim: kernel
	@$(OBJDUMP) $(DISASM) $(KERNEL_ELF) > $(DISASM_TMP)
	@vim $(DISASM_TMP)
	@rm $(DISASM_TMP)

run: build
	@qemu-system-riscv64 \
		-machine virt \
		-nographic \
		-bios $(BOOTLOADER) \
		-device loader,file=$(KERNEL_BIN),addr=$(KERNEL_ENTRY_PA)

debug: build
	@tmux new-session -d \
		"qemu-system-riscv64 -machine virt -nographic -bios $(BOOTLOADER) -device loader,file=$(KERNEL_BIN),addr=$(KERNEL_ENTRY_PA) -s -S" && \
		tmux split-window -h "riscv64-unknown-elf-gdb -ex 'file $(KERNEL_ELF)' -ex 'set arch riscv:rv64' -ex 'target remote localhost:1234'" && \
		tmux -2 attach-session -d

.PHONY: build env kernel clean disasm disasm-vim run debug

"Makefile" 61L, 1519B                                      61,0-1        Bot
```

- `make run` 执行后成功运行

```
@5738fcf47534:/mnt/os
[root@5738fcf47534 os]# vim Makefile
[root@5738fcf47534 os]# make run
    Finished release [optimized] target(s) in 0.05s
[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: 0x10000000 ..= 0x10001fff (rwx)
[rustsbi] pmp1: 0x80000000 ..= 0x8fffffff (rwx)
[rustsbi] pmp2: 0x0 ..= 0xffffffffffffff (---)
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:46 Shutdown machine!
[root@5738fcf47534 os]#
```

二. git提交信息

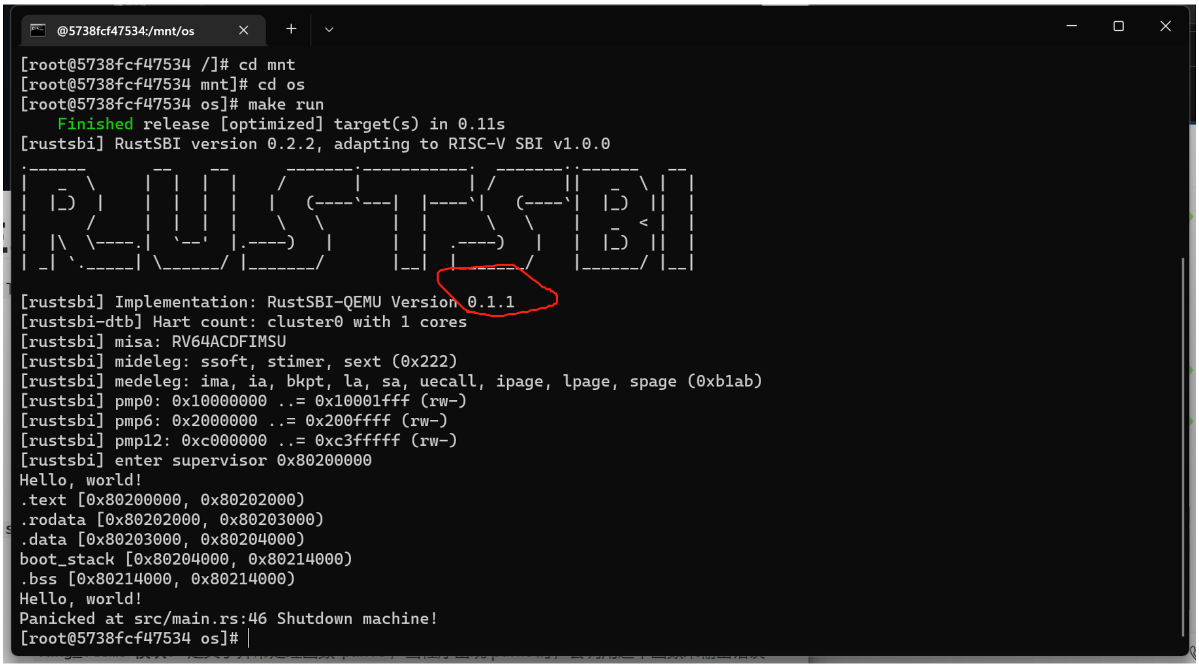


三. 思考题

1. **`linker.ld` 和 `entry.asm` 功能分析:**
   - **`linker.ld`:** 这个文件是链接器脚本，定义了可执行文件的内存布局，包括代码段、只读数据段、可读写数据段、BSS 段等。在这里，它指定了操作系统的入口地址，即 `BASE_ADDRESS` 设置为 `0x80200000`。通过 `SECTIONS` 部分，定义了各个段的起始和结束地址，以及对齐方式。这有助于确保生成的可执行文件在指定内存范围内正确加载和运行。
   - **`entry.asm`:** 这是一个汇编文件，设置了程序的入口点 `_start`，在这里通过 `la sp, boot_stack_top` 设置了栈指针 `sp` 的初始值，然后调用 `rust_main` 函数。同时，在 `.bss.stack` 段定义了一个大小为 4KB * 16 的栈空间 `boot_stack`。
2. **`sbi` 模块和 `lang_items` 模块功能分析:**
   - **`sbi` 模块:** 定义了一系列 SBI (Supervisor Binary Interface) 调用的常量和函数。SBI 是 RISC-V 平台上用于与监管模式（Supervisor Mode）交互的标准接口。通过这个模块，操作系统可以调用 SBI 提供的功能，如控制台输出、关机等。
   - **`lang_items` 模块:** 定义了异常处理函数 `panic`，当程序出现 panic 时，会调用这个函数来输出错误信息，并最终调用 `shutdown` 函数关机。这是一个用于处理异常情况的通用模块。
3. **关于 `rustsbi` 版本不同导致无法运行的问题:**

   这不是可以运行吗? ? ? ? ? ? ? ? ?

   > 截图中`rustsbi`的版本为0.1.1，之前使用的老师给的版本可以看step10中的运行截图，为0.0.2



- 使用`qemu-system-riscv64 -machine virt -nographic -bios ../bootloader/rustsbi.bin -device loader,file=target/riscv64gc-unknown-none-elf/release/os.bin,addr=0x80200000`依然正常运行