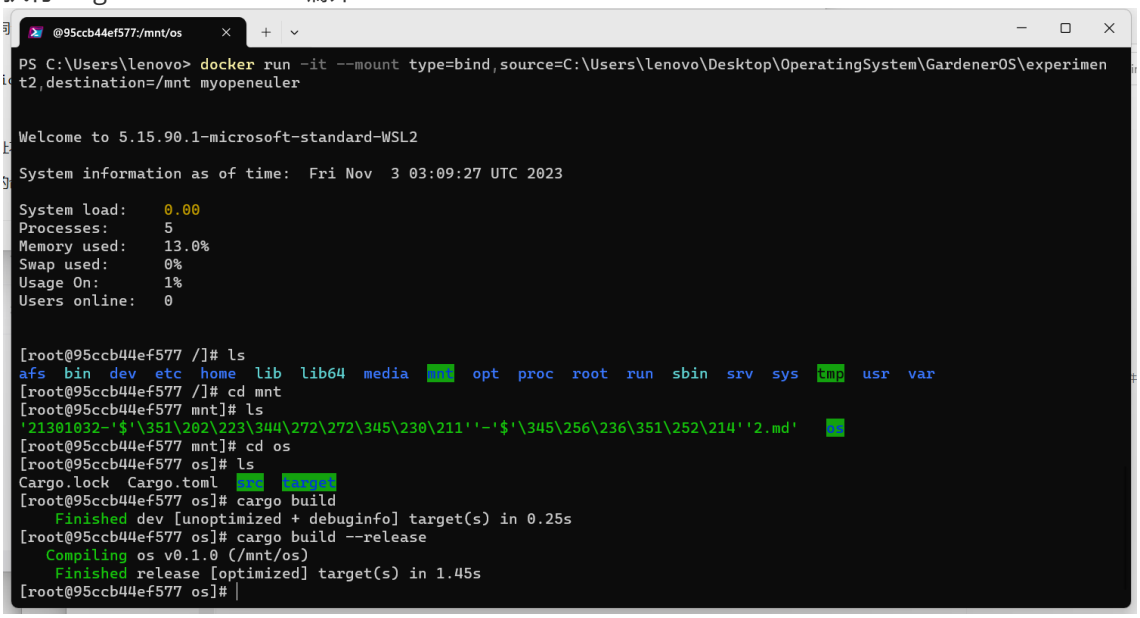
操作系统实验2

> 邓人嘉 21301032

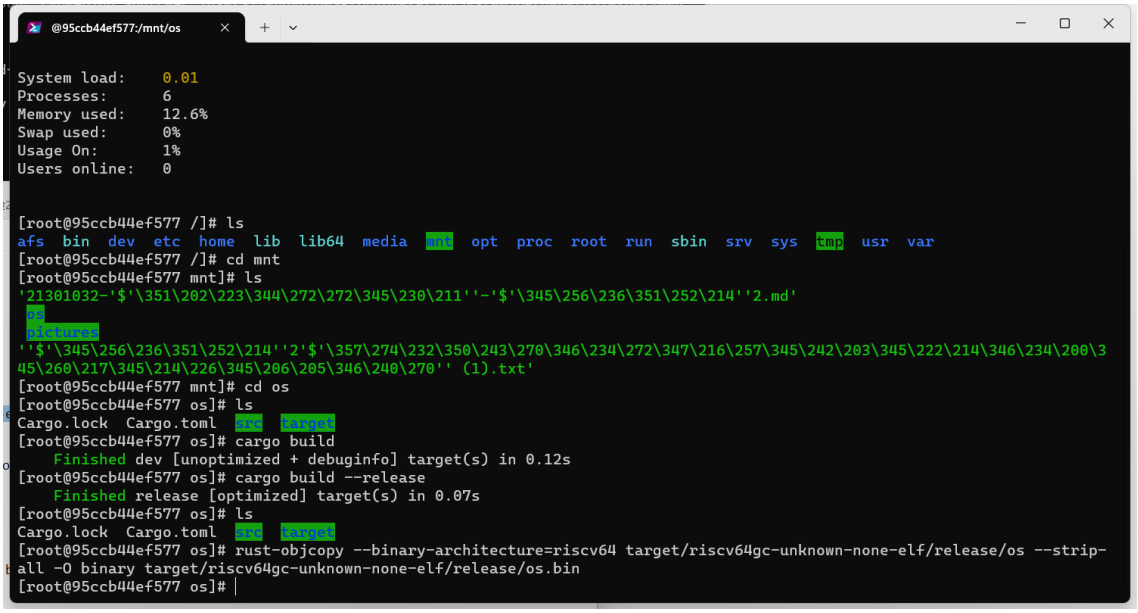
一、实验步骤

1.1 编译生成内核镜像

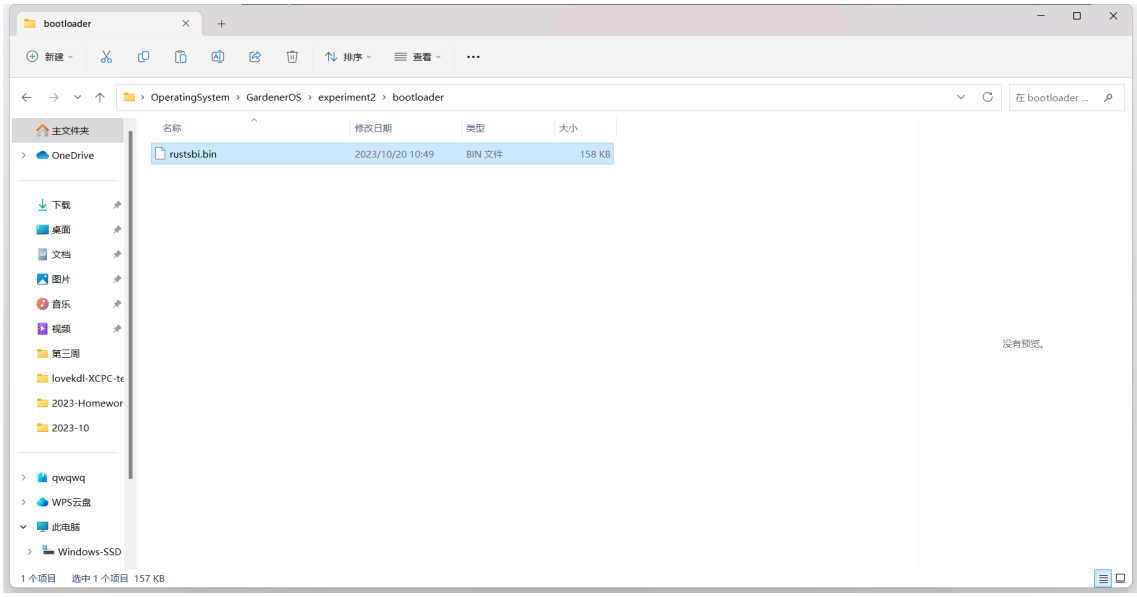
- 执行cargo build --release编译



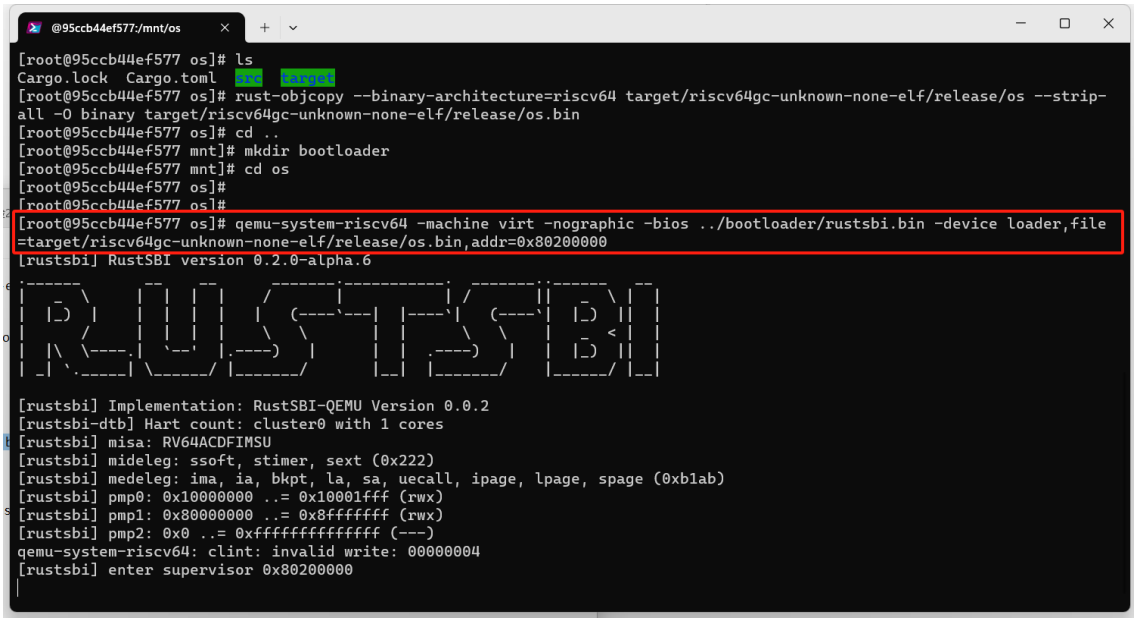
- 把编译生成的ELF执行文件转成binary文件



- 将rustsbi.bin放在bootloader目录

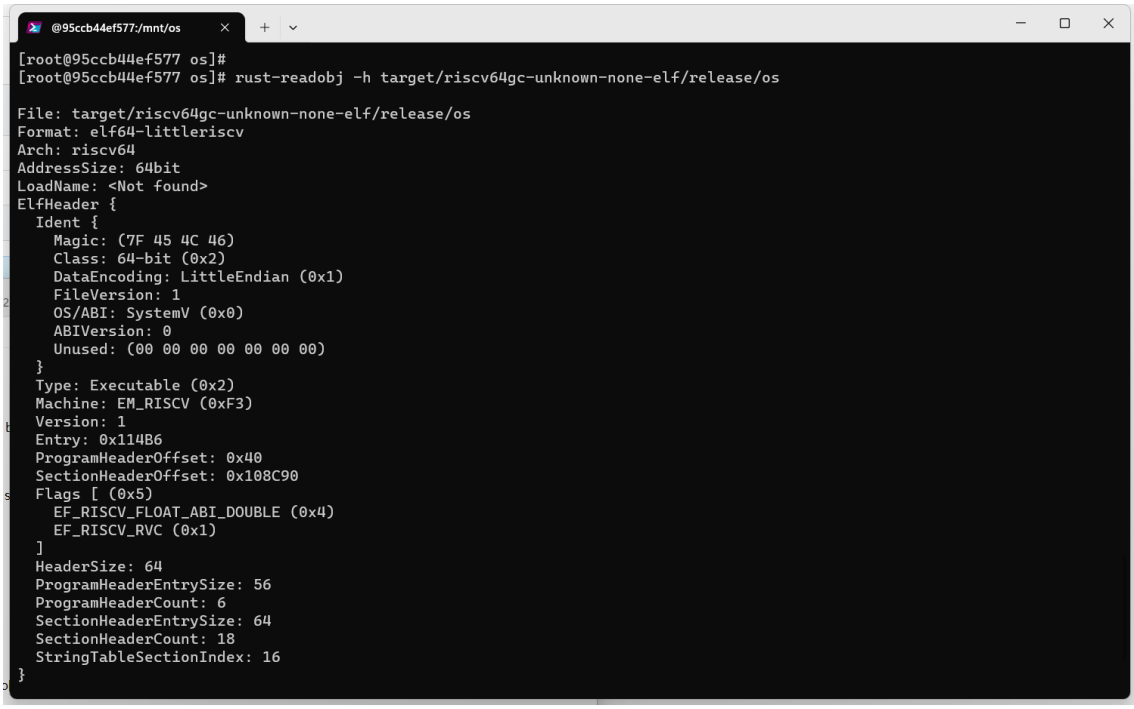


- 接着，加载运行生成的二进制文件。



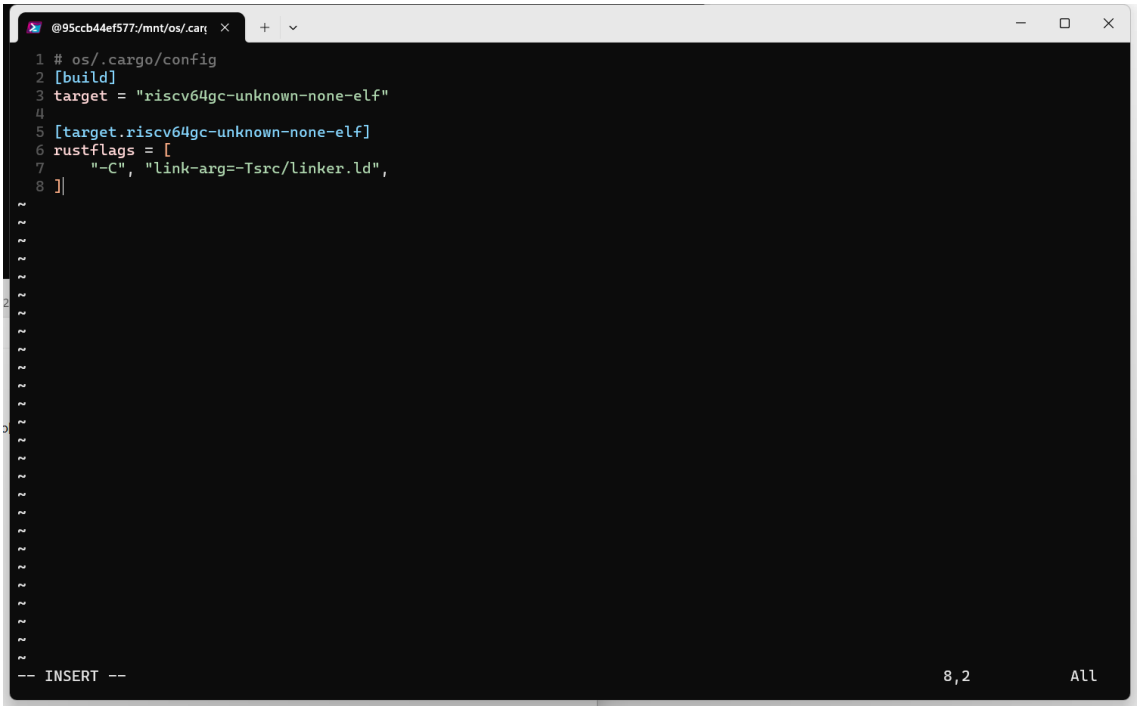
进入了死循环

- 分析，Entry不是0x80200000



1.2 指定内存布局

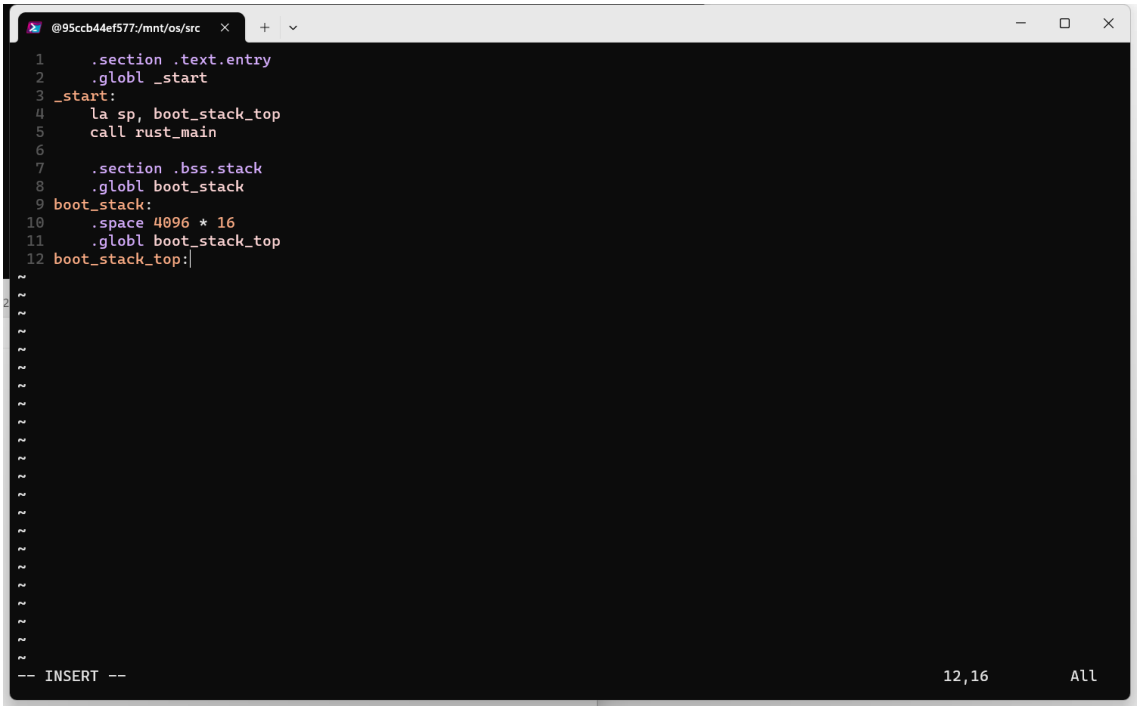
- 修改os/.cargo/config



- 创建链接文件linker.ld

1.3 配置栈空间布局

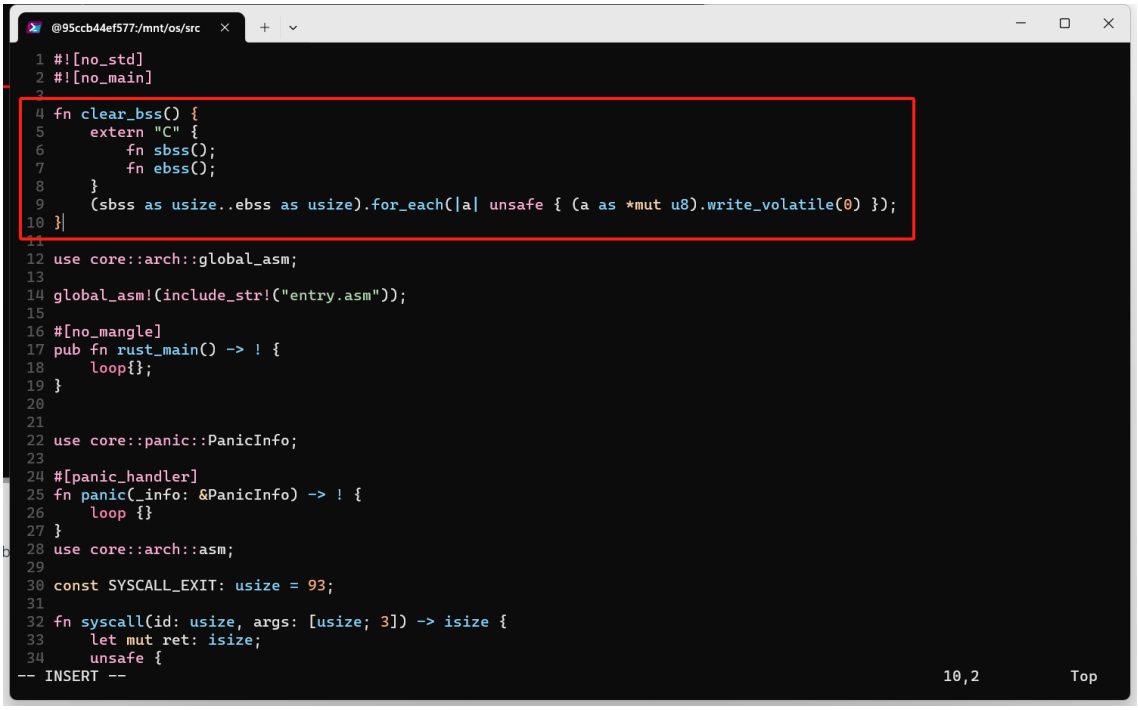
- 创建汇编文件entry.asm



- main.rs 中嵌入汇编代码并声明应用入口 rust_main

1.4 清空bss段

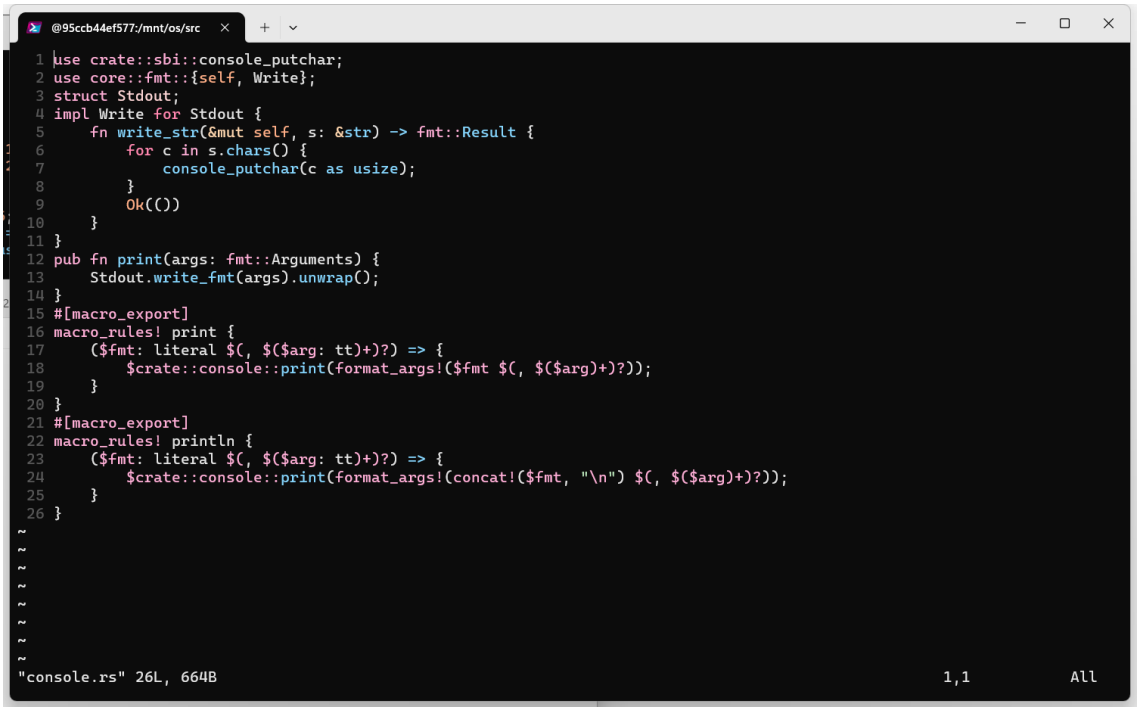
- 为了保证内存的正确性，在main.rs中撰写代码清空.bss段



1.5 实现裸机打印输出信息

- 创建文件sbi.rs

- 在os/src/console.rs中实现裸机上的print函数



```
use crate::sbi::console_putchar;
use core::fmt::{self, Write};
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)+)?));
    }
}
~
~
~
~
~
~
~
~
~
"console.rs" 26L, 664B                                                1,1           All
```

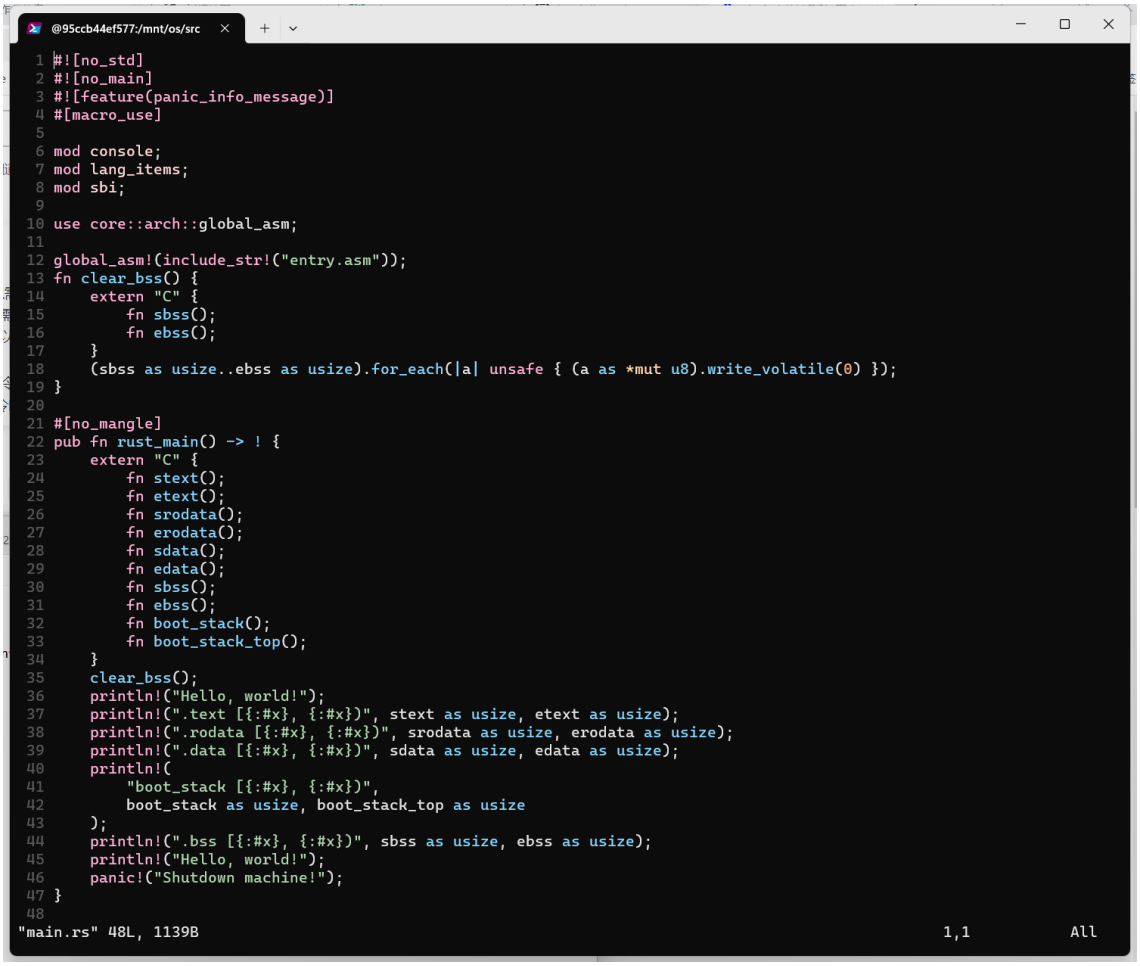
1.6 给异常处理增加输出信息

- 实现os/src/lang_items.rs

```
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                                             1,1           All
```

1.7 修改main.rs输出测试信息

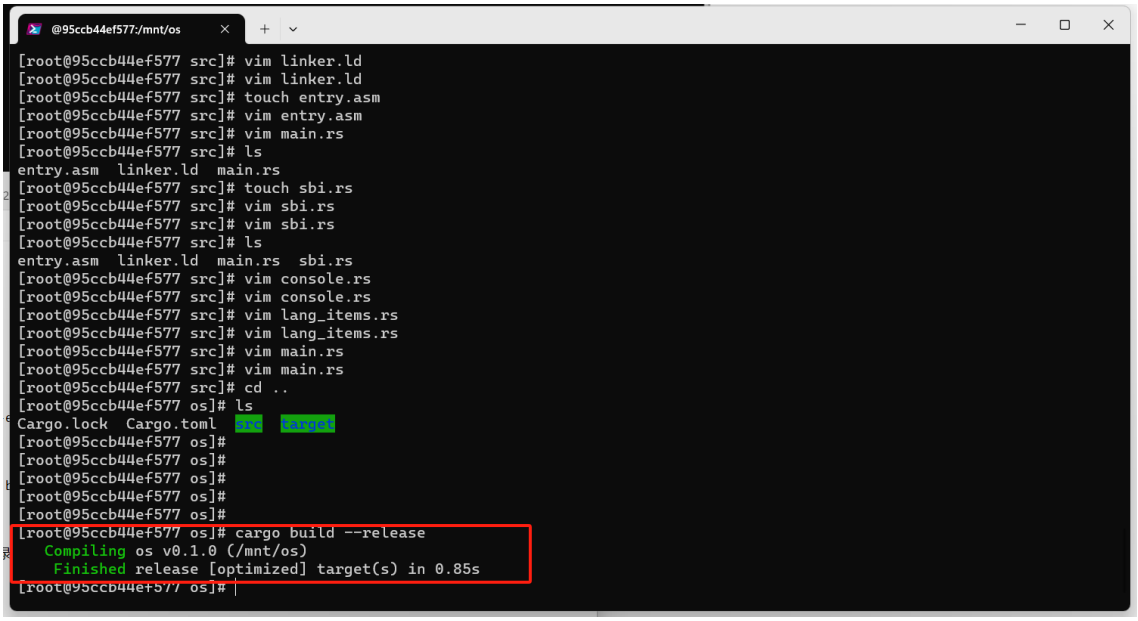
- 修改main.rs的内容



```
#![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();
        fn sbss();
        fn ebss();
        fn boot_stack();
        fn boot_stack_top();
    }
    clear_bss();
    println!("Hello, world!");
    println!(".text [{:#x}, {:#x})", stext as usize, etext as usize);
    println!(".rodata [{:#x}, {:#x})", srodata as usize, erodata as usize);
    println!(".data [{:#x}, {:#x})", sdata as usize, edata as usize);
    println!(
        "boot_stack [{:#x}, {:#x})",
        boot_stack as usize, boot_stack_top as usize
    );
    println!(".bss [{:#x}, {:#x})", sbss as usize, ebss as usize);
    println!("Hello, world!");
    panic!("Shutdown machine!");
}

"main.rs" 48L, 1139B                                                    1,1           All
```

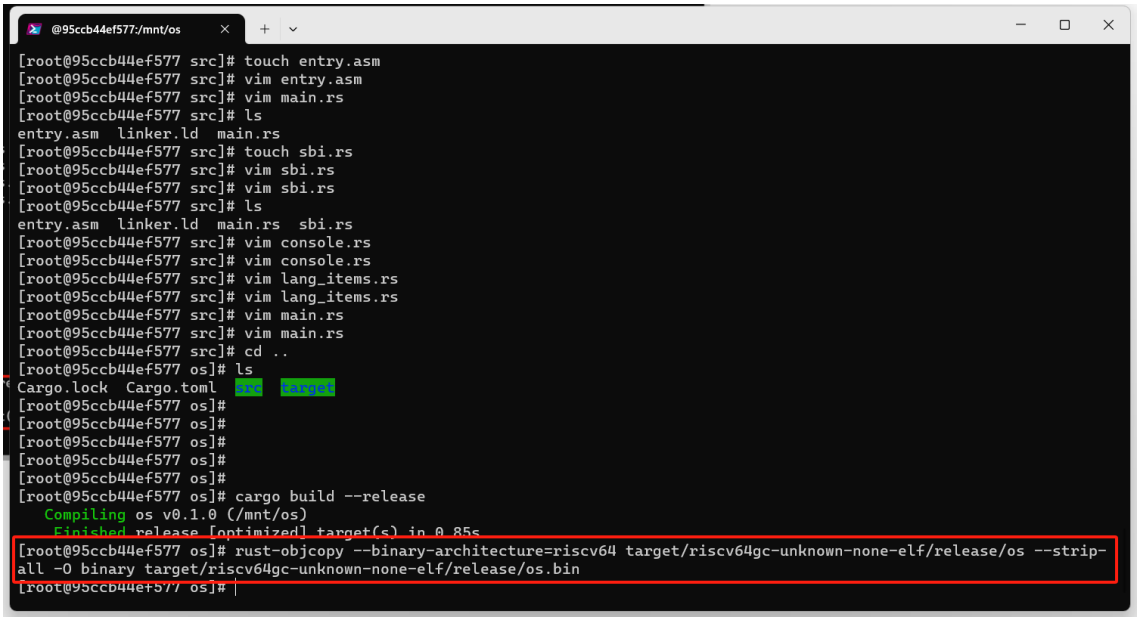
1.8 重新编译以及生成二进制文件

- 编译

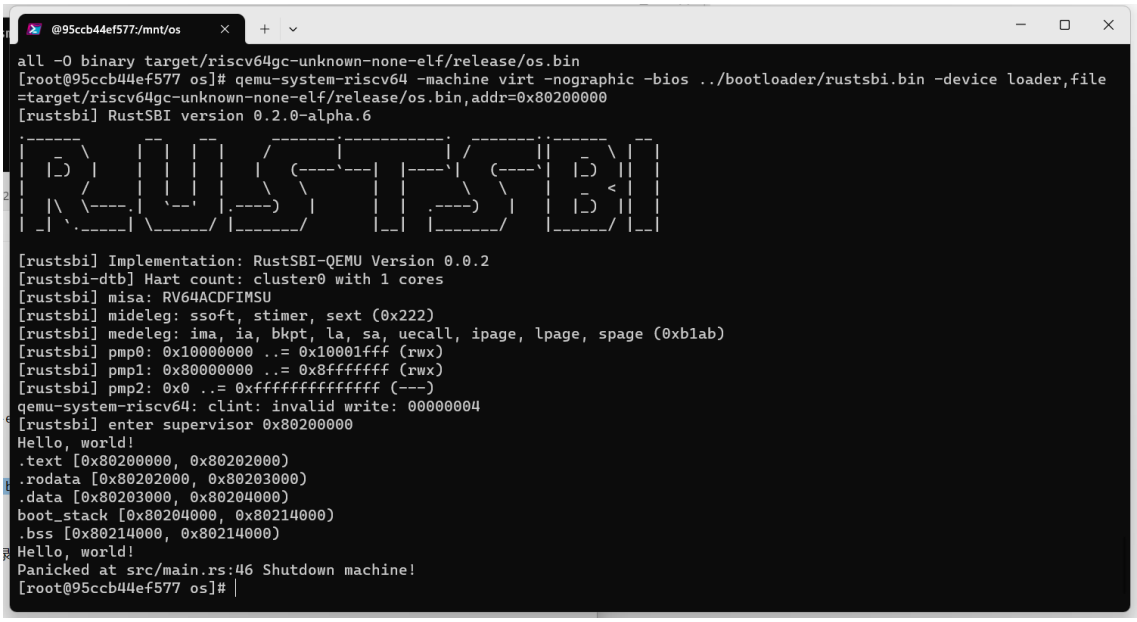


```
[root@95ccb44ef577 src]# vim linker.ld
[root@95ccb44ef577 src]# vim linker.ld
[root@95ccb44ef577 src]# touch entry.asm
[root@95ccb44ef577 src]# vim entry.asm
[root@95ccb44ef577 src]# vim main.rs
[root@95ccb44ef577 src]# ls
entry.asm  linker.ld  main.rs
[root@95ccb44ef577 src]# touch sbi.rs
[root@95ccb44ef577 src]# vim sbi.rs
[root@95ccb44ef577 src]# vim sbi.rs
[root@95ccb44ef577 src]# ls
entry.asm  linker.ld  main.rs  sbi.rs
[root@95ccb44ef577 src]# vim console.rs
[root@95ccb44ef577 src]# vim console.rs
[root@95ccb44ef577 src]# vim lang_items.rs
[root@95ccb44ef577 src]# vim lang_items.rs
[root@95ccb44ef577 src]# vim main.rs
[root@95ccb44ef577 src]# vim main.rs
[root@95ccb44ef577 src]# cd ..
[root@95ccb44ef577 os]# ls
Cargo.lock  Cargo.toml  src  target
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]# cargo build --release
   Compiling os v0.1.0 (/mnt/os)
    Finished release [optimized] target(s) in 0.85s
[root@95ccb44ef577 os]#
```

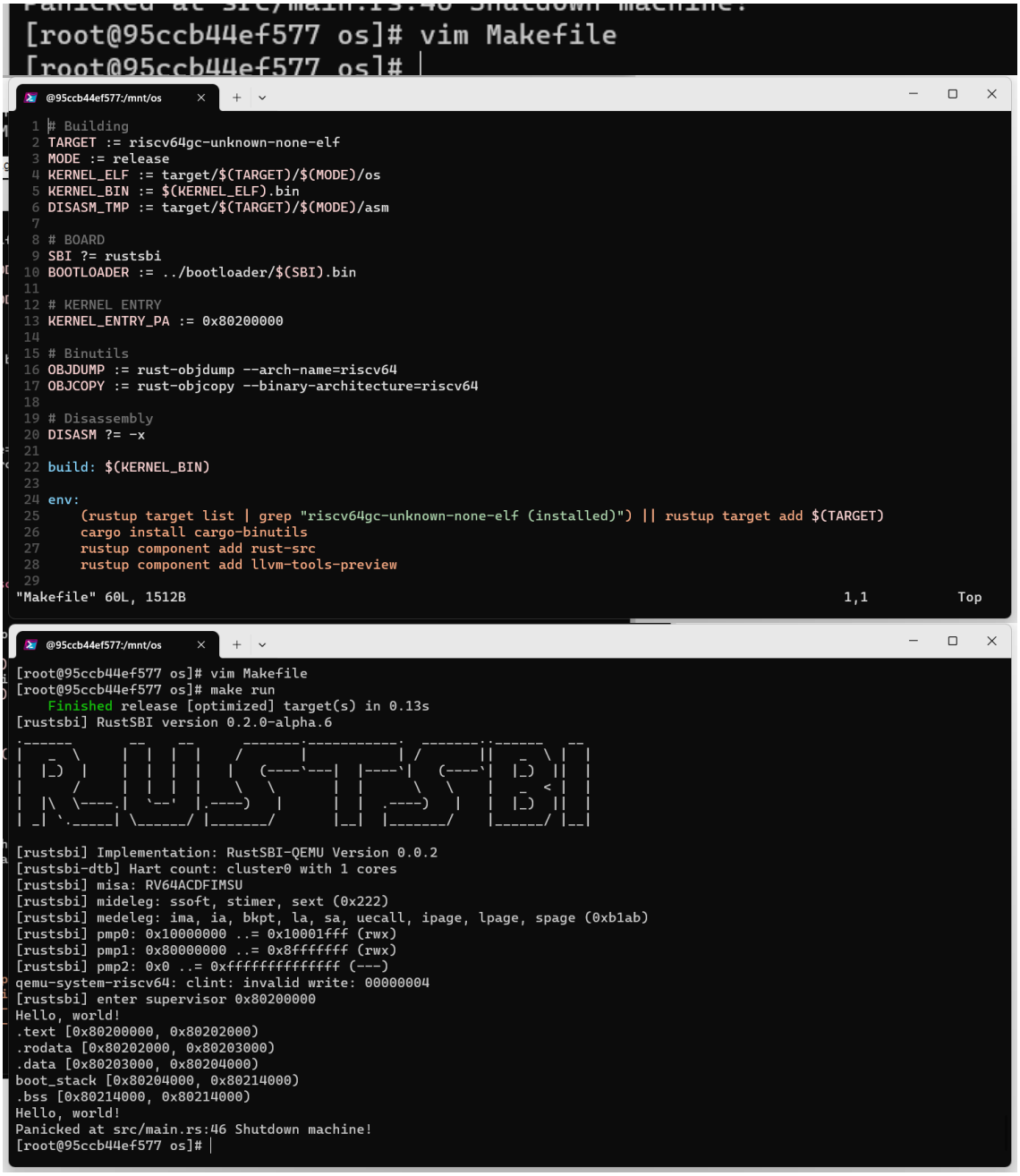
- 生成二进制文件


@95ccb44ef577:/mnt/os
[root@95ccb44ef577 src]# touch entry.asm
[root@95ccb44ef577 src]# vim entry.asm
[root@95ccb44ef577 src]# vim main.rs
[root@95ccb44ef577 src]# ls
entry.asm linker.ld main.rs
[root@95ccb44ef577 src]# touch sbi.rs
[root@95ccb44ef577 src]# vim sbi.rs
[root@95ccb44ef577 src]# vim sbi.rs
[root@95ccb44ef577 src]# ls
entry.asm linker.ld main.rs sbi.rs
[root@95ccb44ef577 src]# vim console.rs
[root@95ccb44ef577 src]# vim console.rs
[root@95ccb44ef577 src]# vim lang_items.rs
[root@95ccb44ef577 src]# vim lang_items.rs
[root@95ccb44ef577 src]# vim main.rs
[root@95ccb44ef577 src]# vim main.rs
[root@95ccb44ef577 src]# cd ..
[root@95ccb44ef577 os]# ls
Cargo.lock Cargo.toml src target
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]#
[root@95ccb44ef577 os]# cargo build --release
Compiling os v0.1.0 (/mnt/os)
Finished release [optimized] target(s) in 0.85s
[root@95ccb44ef577 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@95ccb44ef577 os]#

- 运行


@95ccb44ef577:/mnt/os
all -O binary target/riscv64gc-unknown-none-elf/release/os.bin
[root@95ccb44ef577 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@95ccb44ef577 os]#

- 创建makefile文件，方便编译



二、思考问题

2.1 分析linker.ld和entry.asm所完成的功能

- linker.ld是链接器脚本。**OUTPUT_ARCH(riscv)**指定了生成的可执行文件的体系结构为 **RISC-V**，然后指定程序入口点为**_start**，并设置**BASE_ADDRESS = 0x80200000**，因为程序入口地址约定为**0x80200000**，所以指定链接器的起始地址也设置为**0x80200000**。
  在Section部分包含了包含了代码段(**.text**)、数据段(**.data**和**.rodata**)、未初始化的数据段(**.bss**)，每一段都定义了数据在内存中的对齐格式为4KB，定义了这些段在内存中的布局。最后在**/DISCARD/**部分中舍弃了**.eh_frame**节。

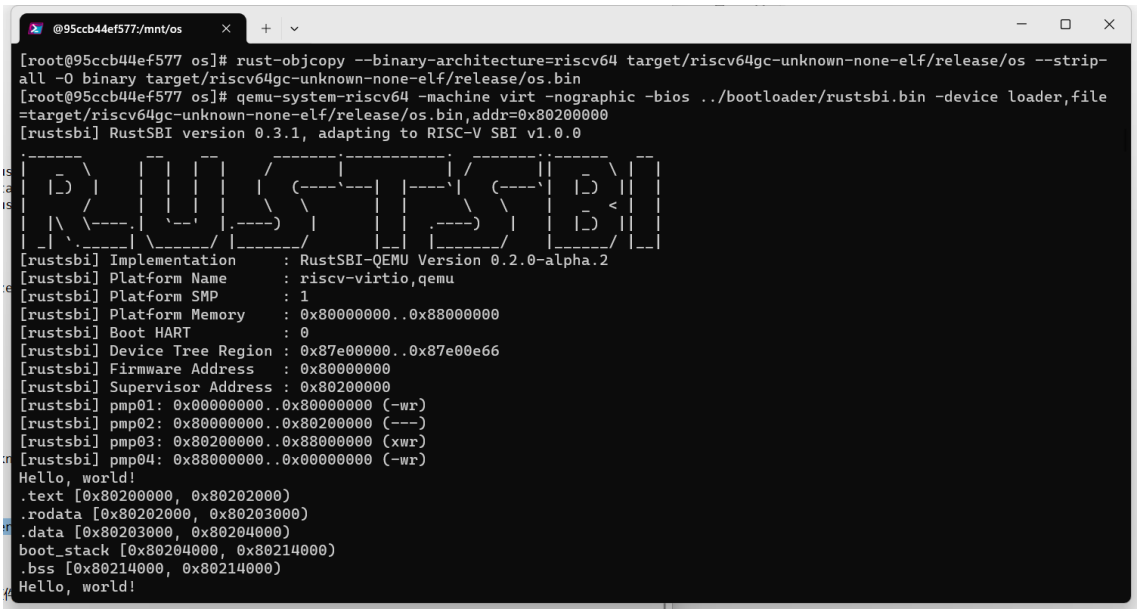
- entry.asm
  .text.entry表示代码段入口，定义了全局符号**_start**作为程序的入口点，与链接器文件中指定的程序入口点相对应，链接器会找到此文件的**_start**入口。**_start**部分里面设置栈指针(sp)地址为启动栈栈顶地址，然后调用**rust_main**函数，转到rust代码。
  .bss.stack段为**boot_stack**分配了**4096*16字节**的栈空间，声明全局符号**boot_stack_top**用于获取栈顶地址。

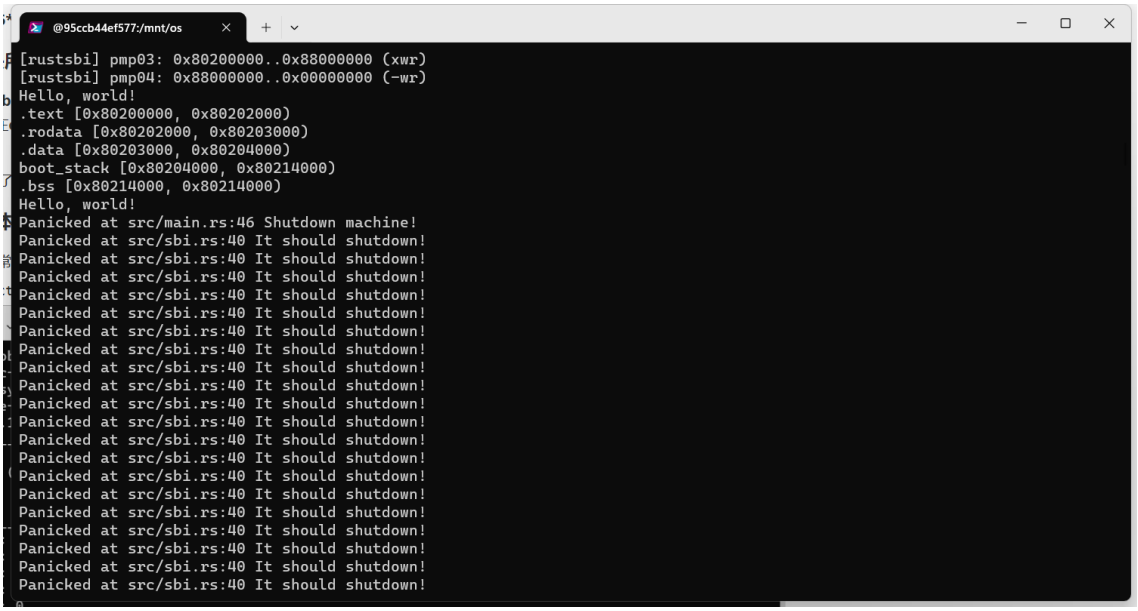
2.2 分析sbi模块和lang_items模块所完成的功能

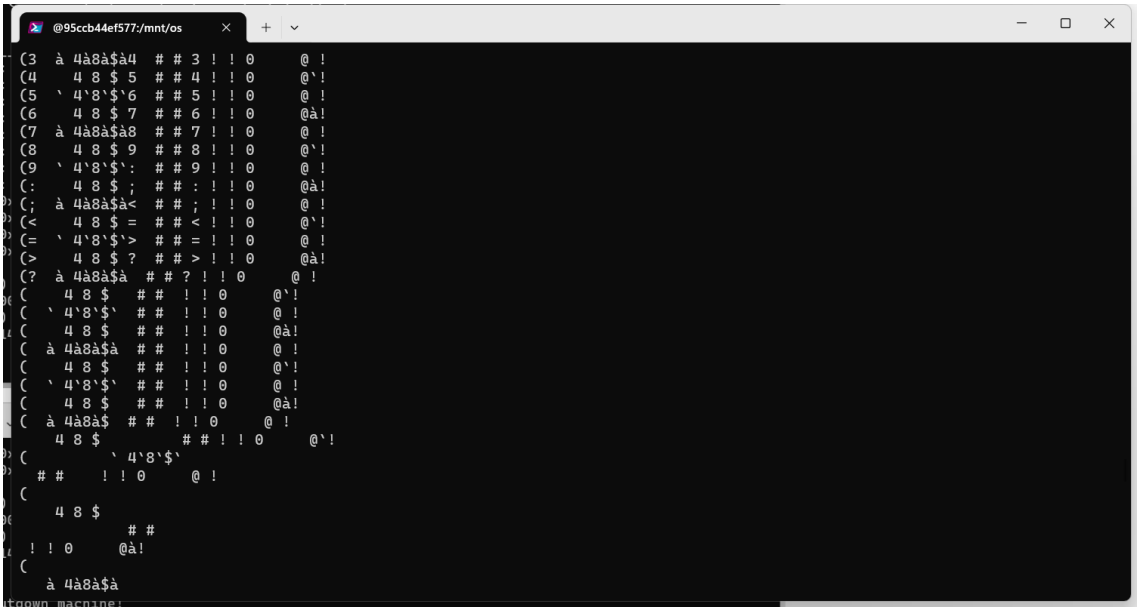
- **sbi**模块可以更改系统调用为sbi调用。**sbi.rs**文件实现了**console_puchar**、**console_getchar**、**shutdown**三个函数，作用分别是输出一个字符、输入一个字符、关闭系统。**console_puchar**接口在**console.rs**文件中被调用，封装为输出一个字符串。**shutdown**接口在**lang_items.rs**文件中的**panic**函数被调用，用于关闭系统。
- **lang_items**实现了**panic**处理器。调用了**sbi**模块的**shutdown**接口，输出**panic**发生的错误位置以及错误信息。

2.3 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。

- 使用最新版的rustsbi.bin，先输出了正常信息，然后进入了死循环，并且输出乱码，如下图：







- 分析原因：根据程序输出内容，可以判断是shutdown出现了问题。
  可以发现panic是这样实现的:

```rust
#[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()
}
```

然后sbi的shutdown是这样实现的：

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

```

所以可能是因为panic调用shutdown，然后shutdown里面又调用了panic，导致一直相互调用，然后死循环一直输出"It should shutdown!"。
说明sbi_call(SBI_SHUTDOWN, 0, 0, 0)时没有成功关闭系统。

尝试调试一下，以验证，修改shutdown：

```rust
pub fn shutdown() -> ! {
    sbi_call(SBI_SHUTDOWN, 0, 0, 0);
    //panic!("It should shutdown!");
    loop{}
}
```

然后运行发现程序没有终止，进入了死循环，也没有输出，说明原因确实是：调用sbi_call(SBI_SHUTDOWN, 0, 0, 0)时没有成功关闭系统。

查看sbi_call函数：

```rust
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
}
```

考虑到新版rustsbi可能不是这样调用接口的。发现文档中说一些SBI函数被遗弃了，其中包含了EID为0x08的shutdown。

> The legacy SBI extensions is deprecated in favor of the other extensions listed below. The legacy console SBI functions (`sbi_console_getchar()` and `sbi_console_putchar()`) are expected to be deprecated; they have no replacement.

于是参照下图的replacement EID把shutdown的EID修改为0x53525354(实际上是sbi_system_reset 的EID)。但其它被遗弃的函数的EID使用原来的并没有影响，所以不用改变，只需要改变shutdown的EID为sbi_system_reset的EID 0x53525354就好。（并且需要添加FID）

5.10. Function Listing

Table 5. Legacy Function List

Function Name	SBI Version	FID	EID	Replacement EID
sbi_set_timer	0.1	0	0x00	0x54494D45
sbi_console_putchar	0.1	0	0x01	N/A
sbi_console_getchar	0.1	0	0x02	N/A
sbi_clear_ipi	0.1	0	0x03	N/A
sbi_send_ipi	0.1	0	0x04	0x735049
sbi_remote_fence_i	0.1	0	0x05	0x52464E43
sbi_remote_sfence_vma	0.1	0	0x06	0x52464E43

16

Function Name	SBI Version	FID	EID	Replacement EID
sbi_remote_sfence_vma_asid	0.1	0	0x07	0x52464E43
sbi_shutdown	0.1	0	0x08	0x53525354
RESERVED			0x09-0x0F	

修改后的sbi.rs如下：

@95ccb44ef577:/mnt/os/src

```rust
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_SYSTEM_RESET: usize = 0x53525354;

#[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("x17") eid,
             in("x16") fid,
             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() -> ! {
    sbi_call(SBI_SYSTEM_RESET, 0, 0, 0, 0);
    panic!("It should shutdown!");
    //loop{}
}
```

然后运行成功，最后系统成功关闭：

@95ccb44ef577:/mnt/os

```
   |     ^ not found in this scope

For more information about this error, try `rustc --explain E0425`.
error: could not compile `os` (bin "os") due to previous error
make: *** [Makefile:34: kernel] Error 101
[root@95ccb44ef577 os]# vim src.sbi.rs
[root@95ccb44ef577 os]# vim src/sbi.rs
[root@95ccb44ef577 os]# make run
   Compiling os v0.1.0 (/mnt/os)
    Finished release [optimized] target(s) in 0.46s
[rustsbi] RustSBI version 0.3.1, adapting to RISC-V SBI v1.0.0
.______       __    __      _______.___________.  _______..______   __
|   _  \     |  |  |  |    /       |           | /       ||   _  \ |  |
|  |_)  |    |  |  |  |   |   (----`---|  |----`|   (----`|  |_)  ||  |
|      /     |  |  |  |    \   \       |  |      \   \    |   _  < |  |
|  |\  \----.|  `--'  |.----)   |      |  |  .----)   |   |  |_)  ||  |
| _| `._____| \______/ |_______/       |__|  |_______/    |______/ |__|
[rustsbi] Implementation     : RustSBI-QEMU Version 0.2.0-alpha.2
[rustsbi] Platform Name      : riscv-virtio,qemu
[rustsbi] Platform SMP       : 1
[rustsbi] Platform Memory    : 0x80000000..0x88000000
[rustsbi] Boot HART          : 0
[rustsbi] Device Tree Region : 0x87e00000..0x87e00e66
[rustsbi] Firmware Address   : 0x80000000
[rustsbi] Supervisor Address : 0x80200000
[rustsbi] pmp01: 0x00000000..0x80000000 (-wr)
[rustsbi] pmp02: 0x80000000..0x80200000 (---)
[rustsbi] pmp03: 0x80200000..0x88000000 (xwr)
[rustsbi] pmp04: 0x88000000..0x00000000 (-wr)
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@95ccb44ef577 os]#
```

三、git截图

- git截图