

19271169-张东植-实验报告2

19271169-张东植-实验报告2

1. 生成内核镜像
2. 指定内存布局
 - 2.1 config
 - 2.2 linker.ld
3. 栈空间布局配置
 - 3.1 assemble
 - 3.2 embed
4. 清空bss段
5. 实现裸机输出
 - 5.1 sbi.rs
 - 5.2 console.rs
6. 异常处理输出
7. 测试信息输出

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。

1. 生成内核镜像

1. Compile using the following commands. Convert the ELF execution files generated by the compilation into binary files.

```
$ cargo build --release

$ rust-objcopy --binary-architecture=riscv64 target/riscv64gc-unknown-none-elf/release/os --strip-all -O binary target/riscv64gc-unknown-none-elf/release/os.bin
```

2. Load the binaries generated by the run. Before this, add bootloader rustsbi.bin to the same directory as the OS directory. Rustsbi. Bin can be downloaded at <https://github.com/rustsbi/rustsbi>.

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

$ rust-readobj -h target/riscv64gc-unknown-none-elf/release/os
```

```
Dal-Z41@ZAYY MINGW64 /d/Develop/temp/os/lab2/gardeneros
$ ls
bootloader/  os/  rust-toolchain

Dal-Z41@ZAYY MINGW64 /d/Develop/temp/os/lab2/gardeneros
$ ls bootloader/
opensbi.bin  rustsbi.bin
```

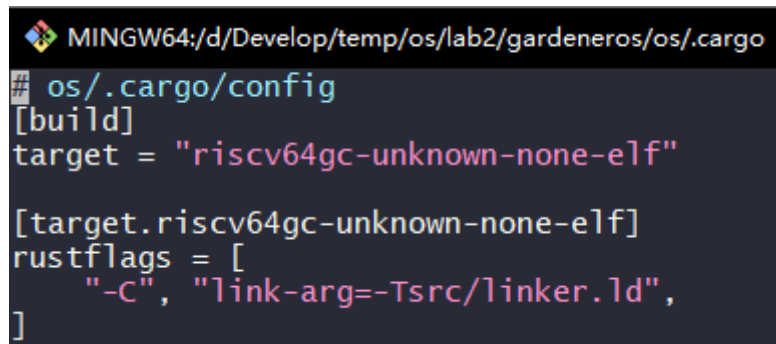
2. 指定内存布局

Memory layouts that specify executable files can be implemented by linking the file "linker.ld". Also, we need to modify Cargo's configuration file to use our linking script instead of the default memory layout.

2.1 config

Modify file "os/.cargo/config".

```
[target.riscv64gc-unknown-none-elf]
rustflags = [
    "-C", "link-arg=-Tsrc/linker.ld",
]
```



```
MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/.cargo
# os/.cargo/config
[build]
target = "riscv64gc-unknown-none-elf"

[target.riscv64gc-unknown-none-elf]
rustflags = [
    "-C", "link-arg=-Tsrc/linker.ld",
]
```

2.2 linker.ld

Modify the file "os/src/linker.ld".

```
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.*)  
}  
  
. = ALIGN(4K);  
edata = .;  
.bss : {  
    *(.bss.stack)  
    sbss = .;  
    *(.bss .bss.*)  
    *(.sbss .sbss.*)  
}  
  
. = ALIGN(4K);  
ebss = .;  
ekernel = .;  
  
/DISCARD/ : {  
    *(.eh_frame)  
}  
}
```

```

MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/src
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.*)
    }

    . = ALIGN(4K);
    edata = .;
    .bss : {
        *(.bss.stack)
        sbss = .;
        *(.bss .bss.*)
        *(.sbss .sbss.*)
    }
}

```

3. 栈空间布局配置

In order for the program to execute correctly, we also need to set up the correct stack space.

3.1 assemble

First, since the stack space is created by assembling "**entry.asm**", modify it.

```

os/src/entry.asm

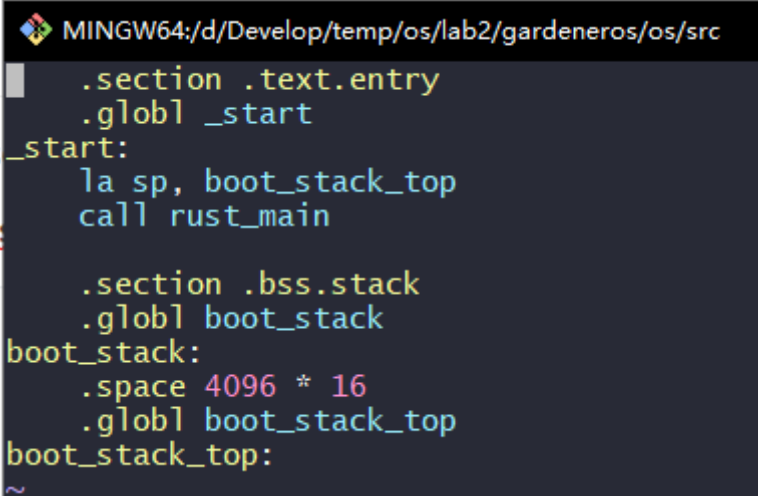
.section .text.entry
.globl _start
_start:
    la sp, boot_stack_top
    call rust_main

```

```

.section .bss.stack
.globl boot_stack
boot_stack:
.space 4096 * 16
.globl boot_stack_top
boot_stack_top:

```



```

MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/src
.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_top:

```

3.2 embed

Embed the assembly code in "**main.rs**" and declare the application entry "**rust_main**".

```

#![feature(global_asm)]

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

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

```



```

MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/src
#![no_std]
#![no_main]
#![feature(asm)]
#![feature(global_asm)]
#![feature(panic_info_message)]
#[macro_use]
mod console;
mod lang_items;
mod sbi;
global_asm!(include_str!("entry.asm"));

```

4. 清空bss段

To ensure correct memory, we also need to write code to clean the .bSS section.

Hence, we need to add some codes to the file "main.rs".

```
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) });
}
```

```
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) });
}
```

5. 实现裸机输出

In order to print information on the bare machine, we need to change the previous system call to SBI call. At the same time, we can also call the interface provided by SBI to realize the shutdown function.

5.1 sbi.rs

First, we need to modify the file "**sbi.rs**" as below.

```
#![allow(unused)]

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;

#[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) {
```

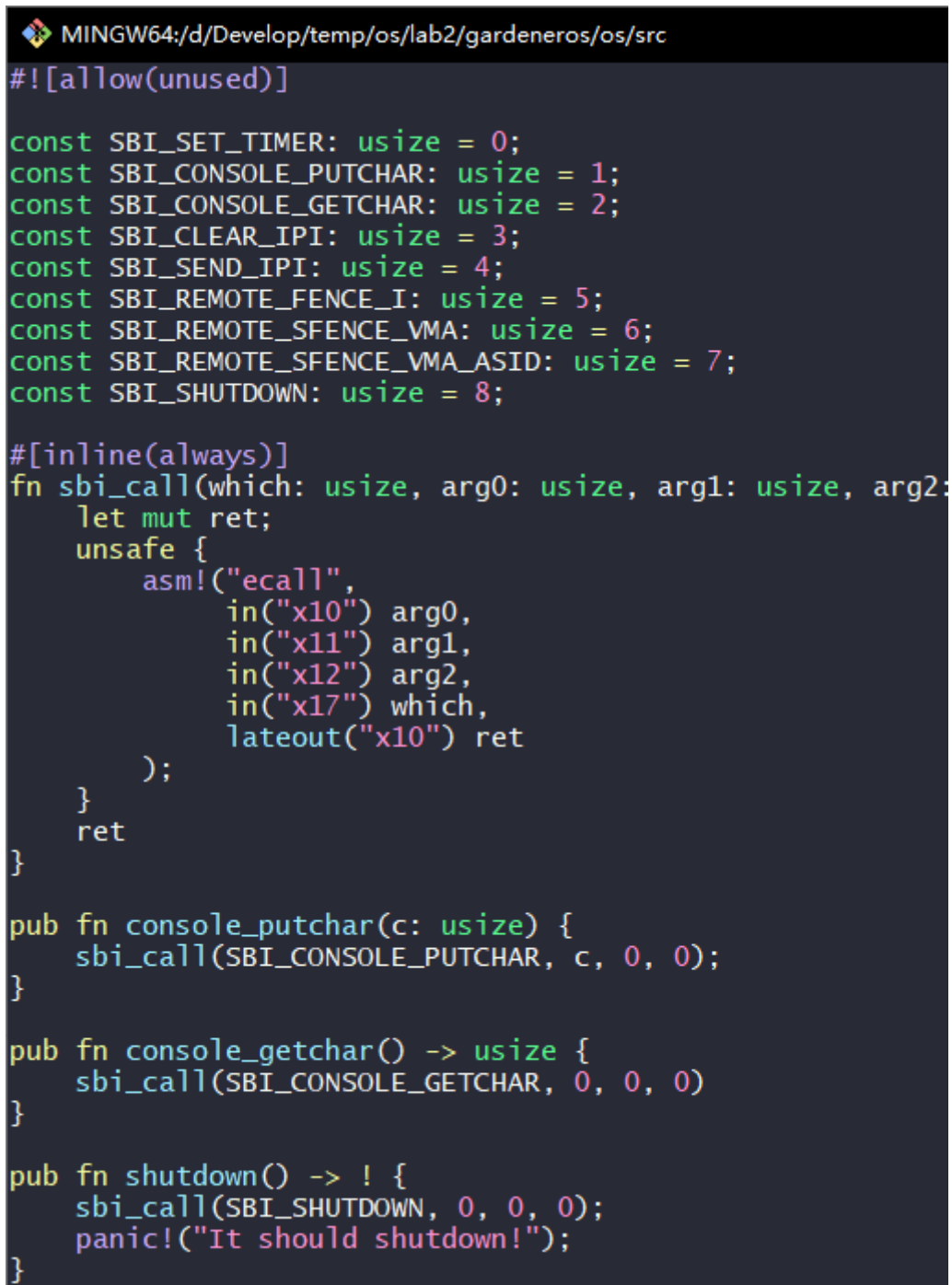
```

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!");
}

```



The screenshot shows a terminal window with the title bar "MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/src". The code is written in Rust and defines constants for SBI calls, an inline assembly function for `sbi_call`, and public functions for `console_putchar`, `console_getchar`, and `shutdown`.

```

#![allow(unused)]

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;

#[inline(always)]
fn sbi_call(which: usize, arg0: usize, arg1: usize, arg2: 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!");
}

```

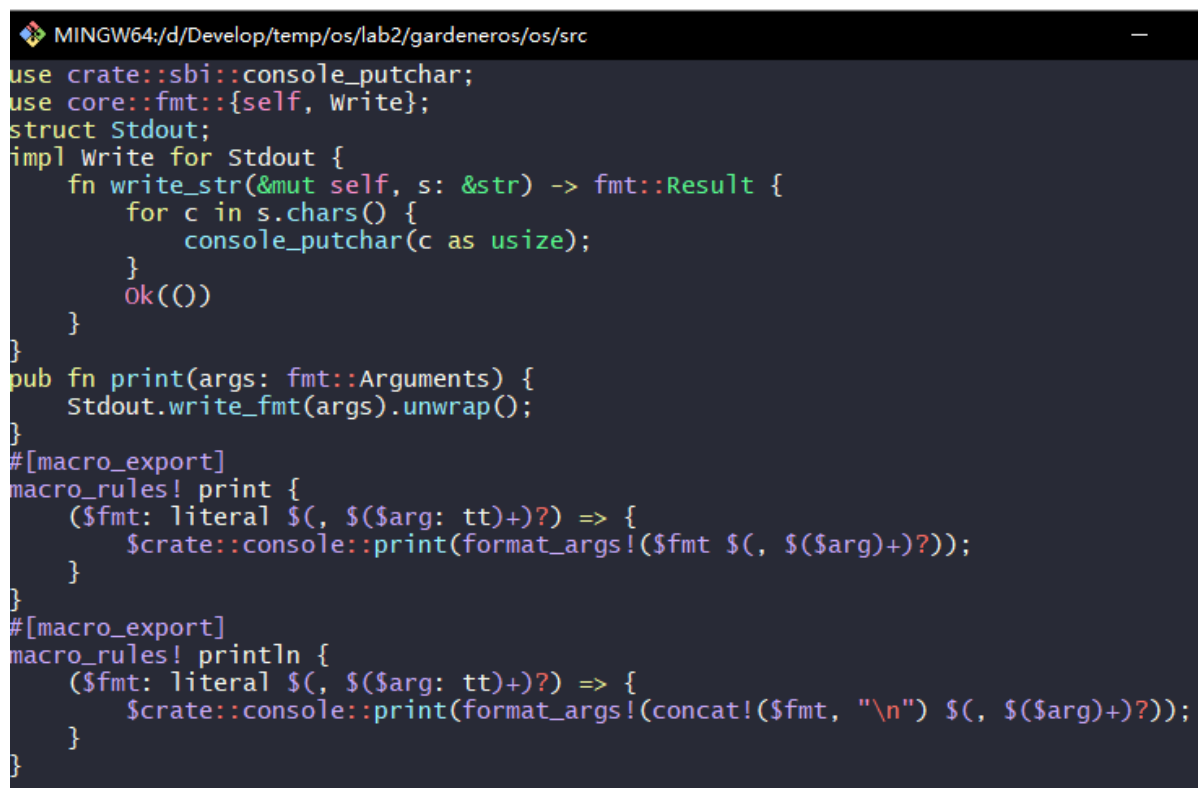
5.2 console.rs

On the basis of the interface provided by "**sbi.rs**", according to the realization of print function in the previous section, we implement print function on bare computer. Hence, we need to modify it in file "**console.rs**".

```

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)+)?));
    }
}

```



```

MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/src
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)+)?));
    }
}

```

6. 异常处理输出

To output the exception messages, we need to get it some restrictions. Make a file "lang_item.rs" and modify it.


```

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()
}

```



```

MINGW64:/d/Develop/temp/os/lab2/gardeneros/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()
}

```

7. 测试信息输出

Modify file "**main.rs**" to output test messages. Then recompile and regenerate the binary files.

```

#![no_std]
#![no_main]
#![feature(asm)]
#![feature(global_asm)]
#![feature(panic_info_message)]
#[macro_use]
mod console;
mod lang_items;
mod sbi;
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!");
}

```

```

MINGW64:/d/Develop/temp/os/lab2/gardeneros/os/src
#[macro_use]
mod console;
mod lang_items;
mod sbi;
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(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!");
}

```

To compile and generate more convenient, we can make a **"Makefile"** file to help us.

MINGW64:/d/Develop/temp/os/lab2/gardeneros/os

```
# Building
TARGET := riscv64gc-unknown-none-elf
MODE := release
KERNEL_ELF := target/$(TARGET)/$(MODE)/os
KERNEL_BIN := $(KERNEL_ELF).bin
DISASM_TMP := target/$(TARGET)/$(MODE)/asm

# BOARD
SBI ?= rustsbi
BOOTLOADER := ../bootloader/$(SBI).bin

# KERNEL ENTRY
KERNEL_ENTRY_PA := 0x80200000

# Binutils
OBJDUMP := rust-objdump --arch-name=riscv64
OBJCOPY := rust-objcopy --binary-architecture=riscv64

# Disassembly
DISASM ?= -x

build: $(KERNEL_BIN)

env:
    (rustup target list | grep "riscv64gc-unknown-none-elf" && rustup target add $(TARGET))
    cargo install cargo-binutils
    rustup component add rust-src
    rustup component add llvm-tools-preview

$(KERNEL_BIN): kernel
    @$$(OBJCOPY) $(KERNEL_ELF) --strip-all -O binary $@

kernel:
    @cargo build --release

clean:
    @cargo clean

disasm: kernel
```

```
root@iZuf6ci3vt7rdcc1658exhZ:~/os# cargo build
   Compiling os v0.1.0 (/root/os)
   Finished dev [unoptimized + debuginfo] target(s) in 2.90s
root@iZuf6ci3vt7rdcc1658exhZ:~/os# cargo run
   Finished dev [unoptimized + debuginfo] target(s) in 0.00s
   Running `target/riscv64gc-unknown-none-elf/debug/os`
```