19271169-张东植-实验报告3

19271169-张东植-实验报告3
1. 设计与实现程序
    1.1 System call
    1.2 Format Output
    1.3 Senmatic Support
    1.4 Memory Layout
    1.5 Runtime Library
    1.6 Template
    1.7 Multiple programs
    1.8 Makefile
2. 内核链接
3. 加载程序二进制码
4. 实现用户栈与内核栈
5. 实现trap管理
    5.1 Context Store and Resume
    5.2 Trap Distribution and Handling
    5.3 System call handling
6. 执行程序
7. 批处理

- **Gitlab repo: http://202.205.102.126:88/ZhangDongZhi/os-lab.git**

The main objective of this experiment is to implement a simple batch operating system and understand the concept of privileges. Through this experiment, we can realize a simple batch operating system and understand the concept of privilege level. It includes designing and implementing application program, linking application program to kernel, finding and loading application binary code, realizing user stack and kernel stack, realizing Trap management, etc.

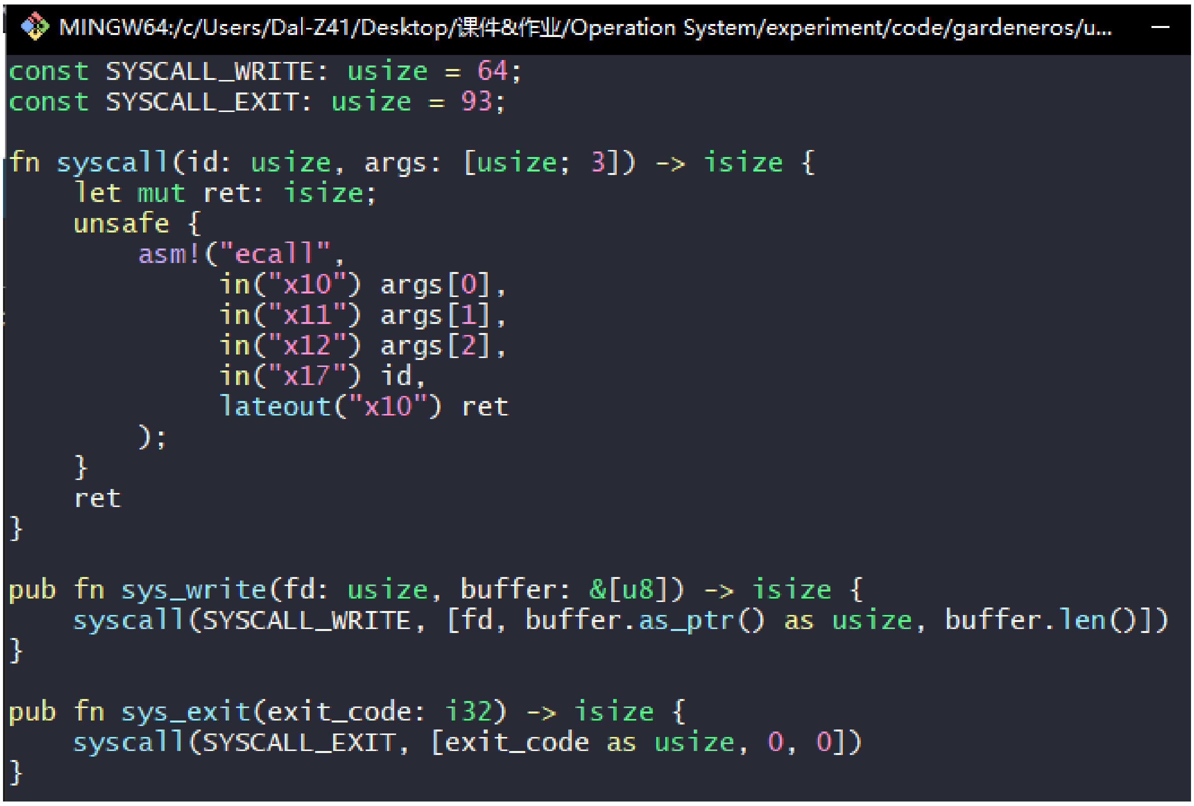
1. 设计与实现程序

To implement a simple batch operating system, we need to implement the application first, which requires the application to run in user mode. The application and its corresponding library files are placed in the user directory under the root directory.

1.1 System call

Implement two system calls in file **"syscall.rs"** as below.

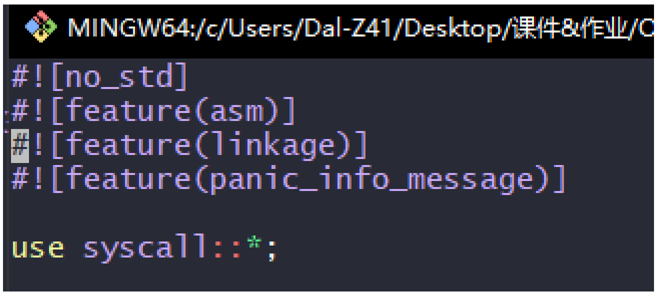
```
const SYSCALL_WRITE: usize = 64;
const SYSCALL_EXIT: usize = 93;

fn syscall(id: usize, args: [usize; 3]) -> isize {
    let mut ret: isize;
    unsafe {
        asm!("ecall",
             in("x10") arg[0],
             in("x11") arg[1],
             in("x12") arg[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 sys_exit(exit_code: i32) -> isize {
    syscall(SYSCALL_EXIT, [exit_code as usize, 0, 0])
}
```


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/u...

```
const SYSCALL_WRITE: usize = 64;
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_write(fd: usize, buffer: &[u8]) -> isize {
    syscall(SYSCALL_WRITE, [fd, buffer.as_ptr() as usize, buffer.len()])
}

pub fn sys_exit(exit_code: i32) -> isize {
    syscall(SYSCALL_EXIT, [exit_code as usize, 0, 0])
}
```

Besides, encapsulate it in file **"lib.rs"**.

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

use syscall::*;

pub fn write(fd: usize, buf: &[u8]) -> isize { sys_write(fd, buf) }
pub fn exit(exit_code: i32) -> isize { sys_exit(exit_code) }
```

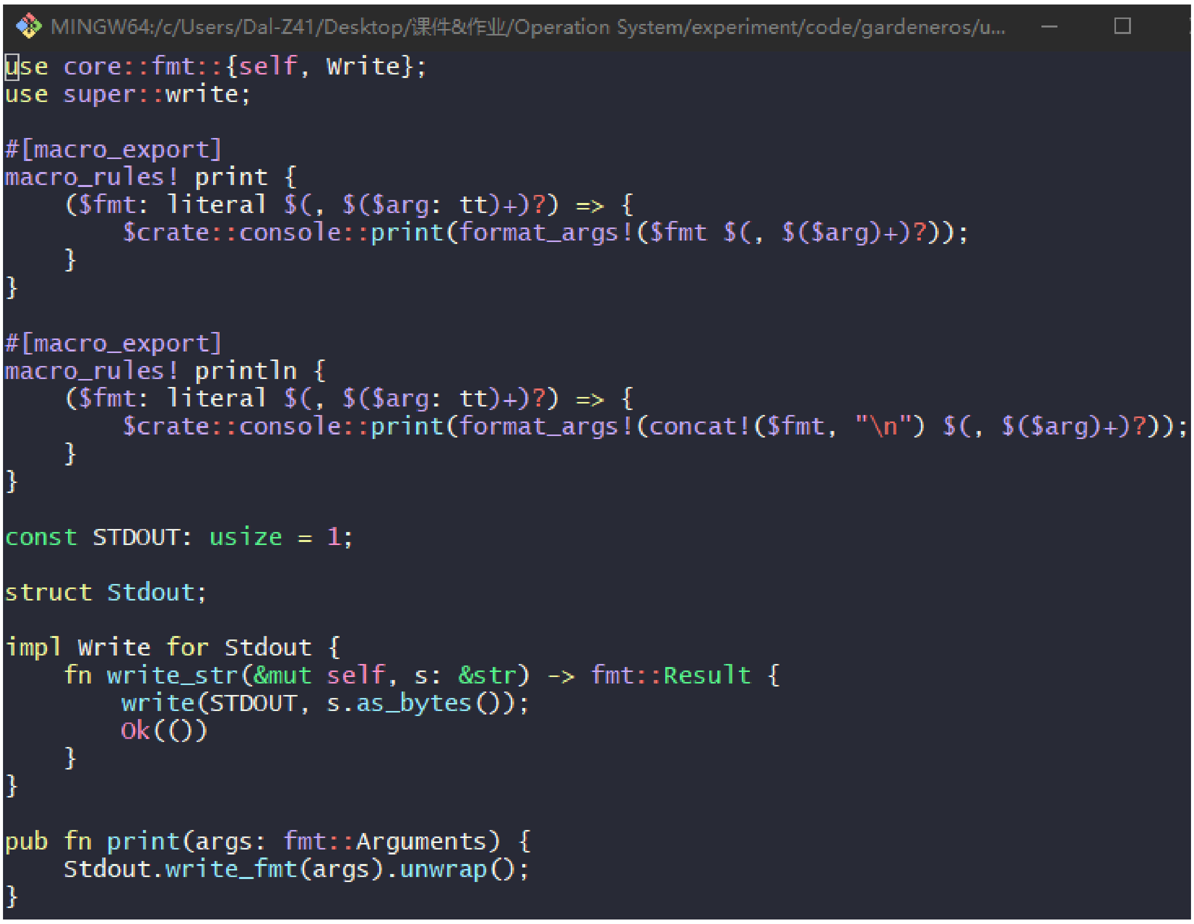


```
#![no_std]
#![feature(asm)]
#![feature(linkage)]
#![feature(panic_info_message)]

use syscall::*;
```

1.2 Format Output

To format output, we also need to change **stdout::write_str** to a write-based implementation and pass in the "fd" argument to 1, which represents standard output, that is, output to the screen. Hence, we need to modify file **"console.rs"**.

```rust
use core::fmt::{self, Write};
use super::write;

const STDOUT: usize = 1;

struct Stdout;

impl Write for Stdout {
    fn write_str(&mut self, s: &str) -> fmt::Result {
        write(STDOUT, s.as_bytes());
        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)+)?));
    }
}
```

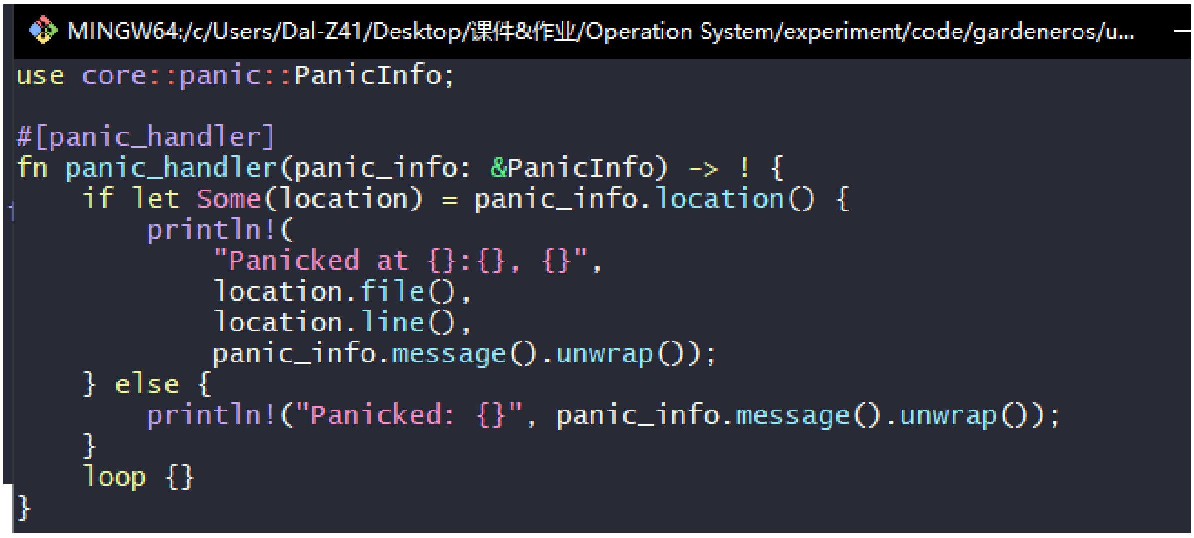

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/u...

```rust
use core::fmt::{self, Write};
use super::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)+)?) => {
        $crate::console::print(format_args!(concat!($fmt, "\n") $(, $($arg)+)?));
    }
}

const STDOUT: usize = 1;

struct Stdout;

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

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

1.3 Senmatic Support

To have the senmatic support, we need to implement **panic**. Hence, we need to modify file **"lang_items.rs"**.

```rust
use core::panic::PanicInfo;

#[panic_handler]
fn panic_handler(panic_info: &PanicInfo) -> ! {
    if let Some(location) = panic_info.location() {
        println!(
            "Panicked at {}:{}, {}",
            location.file(),
            location.line(),
            panic_info.message().unwrap());
    } else {
        println!("Panicked: {}", panic_info.message().unwrap());
    }
    loop {}
}
```

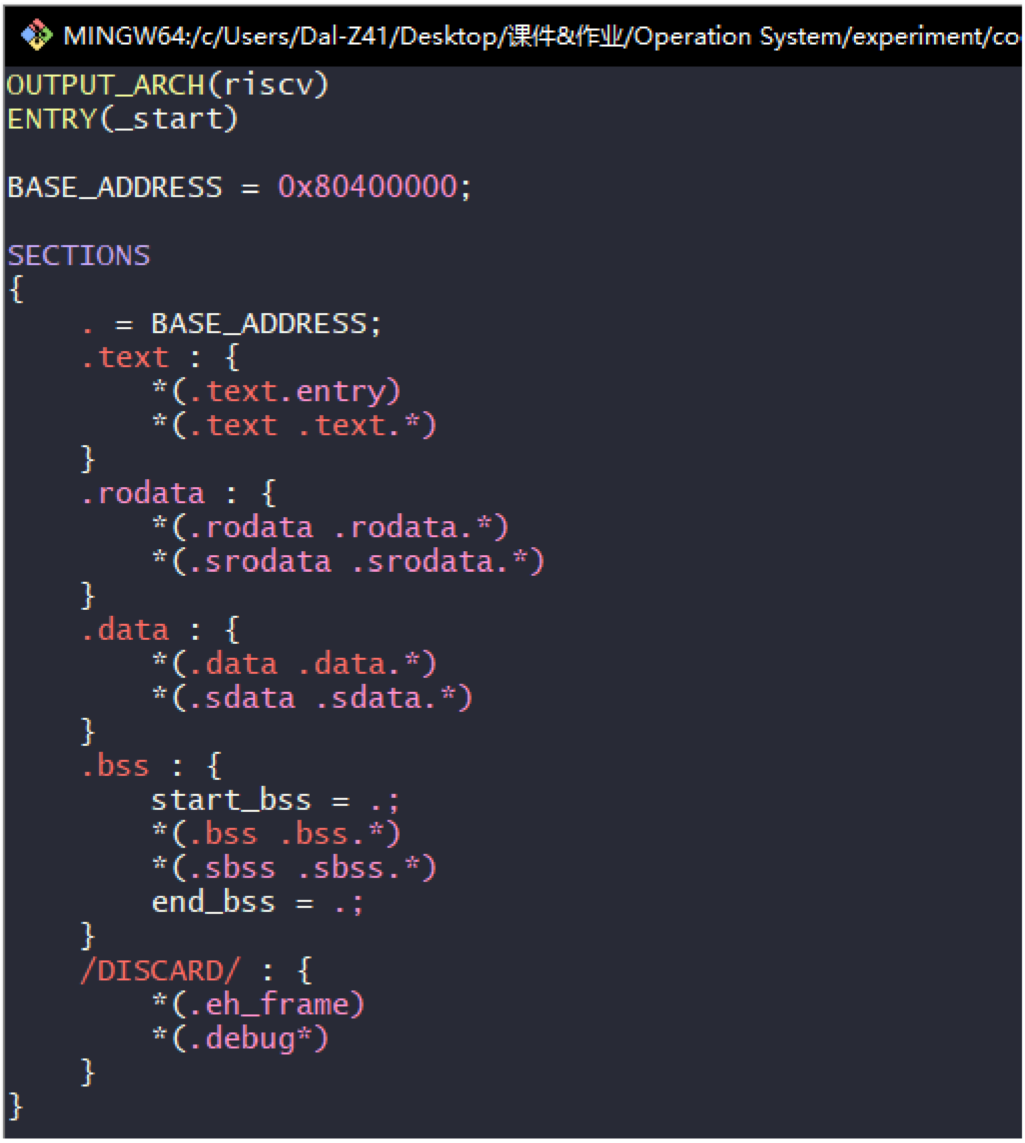

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/u...

```rust
use core::panic::PanicInfo;

#[panic_handler]
fn panic_handler(panic_info: &PanicInfo) -> ! {
    if let Some(location) = panic_info.location() {
        println!(
            "Panicked at {}:{}, {}",
            location.file(),
            location.line(),
            panic_info.message().unwrap());
    } else {
        println!("Panicked: {}", panic_info.message().unwrap());
    }
    loop {}
}
```

1.4 Memory Layout

We need to set the starting physical address of the application to 0x80400000, so that the application will be loaded to run at this physical address, entering the entry point of the user library, and jumping to the main application logic after initialization. Hence, we need to modify file **"linker.ld"**.

```
OUTPUT_ARCH(riscv)
ENTRY(_start)

BASE_ADDRESS = 0x80400000;

SECTIONS
{
    . = BASE_ADDRESS;
    .text : {
        *(.text.entry)
        *(.text .text.*)
    }
    .rodata : {
        *(.rodata .rodata.*)
        *(.srodata .srodata.*)
    }
    .data : {
        *(.data .data.*)
        *(.sdata .sdata.*)
    }
    .bss : {
        start_bss = .;
        *(.bss .bss.*)
        *(.sbss .sbss.*)
        end_bss = .;
    }
    /DISCARD/ : {
        *(.eh_frame)
        *(.debug*)
    }
}
```



MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/co

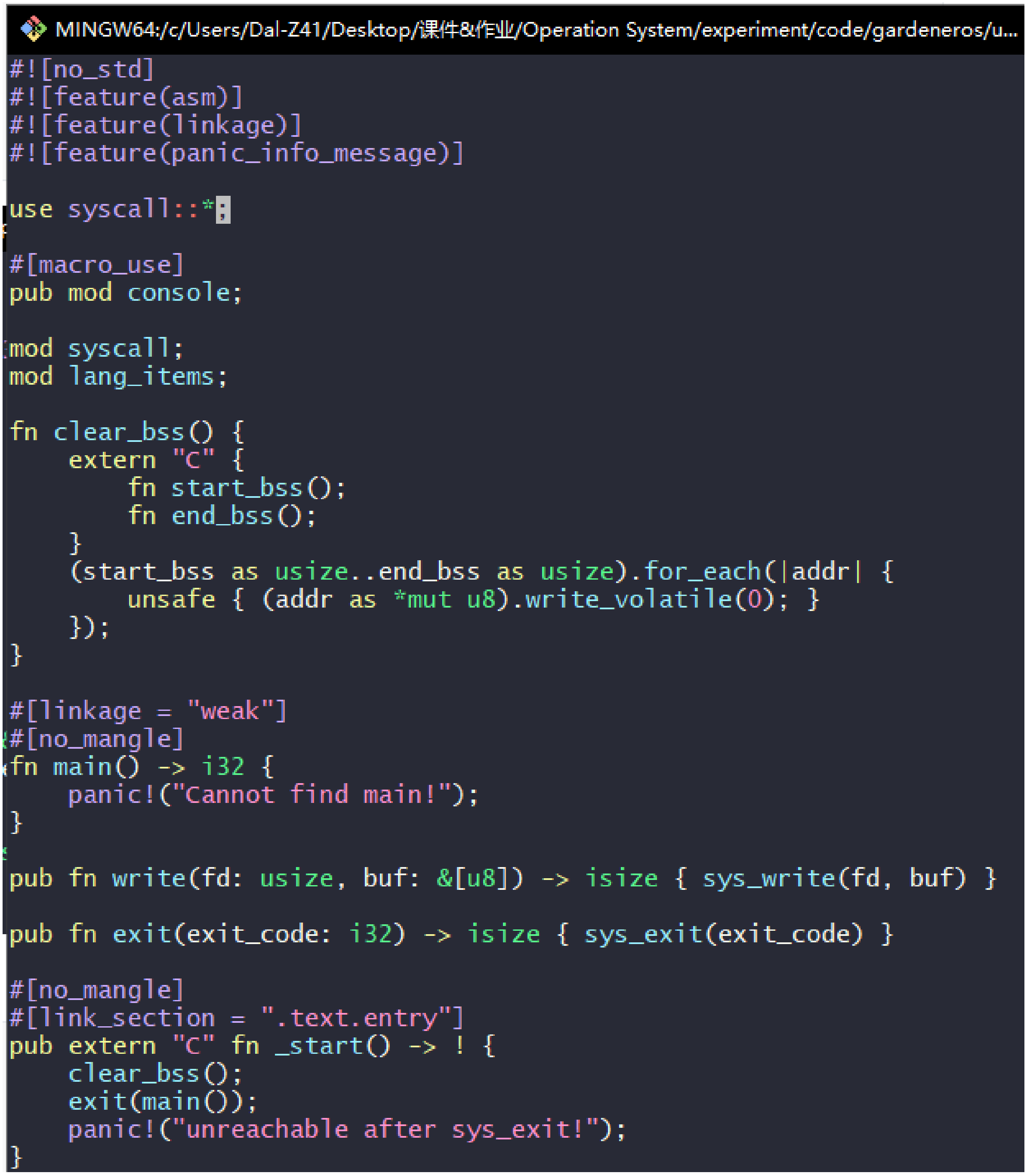
```
OUTPUT_ARCH(riscv)
ENTRY(_start)

BASE_ADDRESS = 0x80400000;

SECTIONS
{
    . = BASE_ADDRESS;
    .text : {
        *(.text.entry)
        *(.text .text.*)
    }
    .rodata : {
        *(.rodata .rodata.*)
        *(.srodata .srodata.*)
    }
    .data : {
        *(.data .data.*)
        *(.sdata .sdata.*)
    }
    .bss : {
        start_bss = .;
        *(.bss .bss.*)
        *(.sbss .sbss.*)
        end_bss = .;
    }
    /DISCARD/ : {
        *(.eh_frame)
        *(.debug*)
    }
}
```

1.5 Runtime Library

We need to define the entry point of the user library "**_start**", which is compiled and stored in the.text.entry code segment. In addition, use **#[linkage = "weak"]** to ensure that **"lib.rs"** and bin compilations with main on them will pass.

```
#![feature(linkage)]
#![feature(panic_info_message)]

#[macro_use]
pub mod console;
mod syscall;
mod lang_items;

fn clear_bss() {
    extern "C" {
        fn start_bss();
        fn end_bss();
    }
    (start_bss as usize..end_bss as usize).for_each(|addr| {
        unsafe { (addr as *mut u8).write_volatile(0); }
    });
}
```

```
#[no_mangle]
#[link_section = ".text.entry"]
pub extern "C" fn _start() -> ! {
    clear_bss();
    exit(main());
    panic!("unreachable after sys_exit!");
}

#[linkage = "weak"]
#[no_mangle]
fn main() -> i32 {
    panic!("Cannot find main!");
}
```


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/u...

```
#![no_std]
#![feature(asm)]
#![feature(linkage)]
#![feature(panic_info_message)]

use syscall::*;

#[macro_use]
pub mod console;

mod syscall;
mod lang_items;

fn clear_bss() {
    extern "C" {
        fn start_bss();
        fn end_bss();
    }
    (start_bss as usize..end_bss as usize).for_each(|addr| {
        unsafe { (addr as *mut u8).write_volatile(0); }
    });
}

#[linkage = "weak"]
#[no_mangle]
fn main() -> i32 {
    panic!("Cannot find main!");
}

pub fn write(fd: usize, buf: &[u8]) -> isize { sys_write(fd, buf) }

pub fn exit(exit_code: i32) -> isize { sys_exit(exit_code) }

#[no_mangle]
#[link_section = ".text.entry"]
pub extern "C" fn _start() -> ! {
    clear_bss();
    exit(main());
    panic!("unreachable after sys_exit!");
}
```

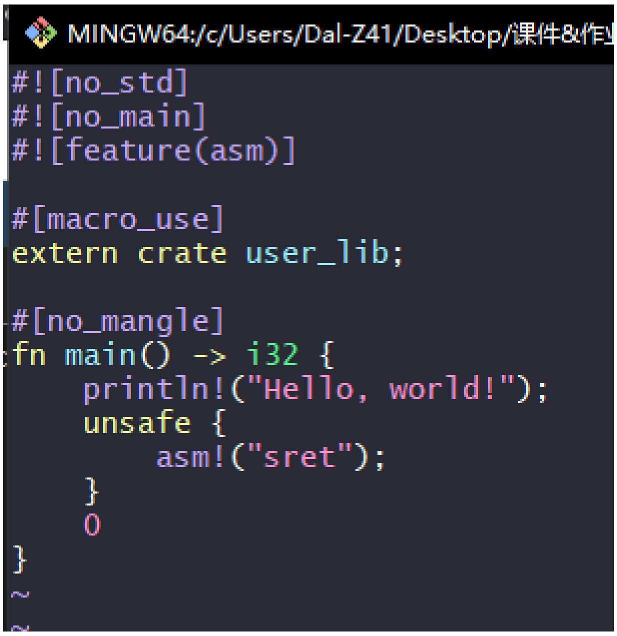
1.6 Template

The applications are stored in **"usr/src/bin"** with the following template. An external library is introduced in the code, which is the **"lib.rs"** definition and the submodules it references.

```rust
#![no_std]
#![no_main]

#[macro_use]
extern crate user_lib;

#[no_mangle]
fn main() -> i32 {
    0
}
```

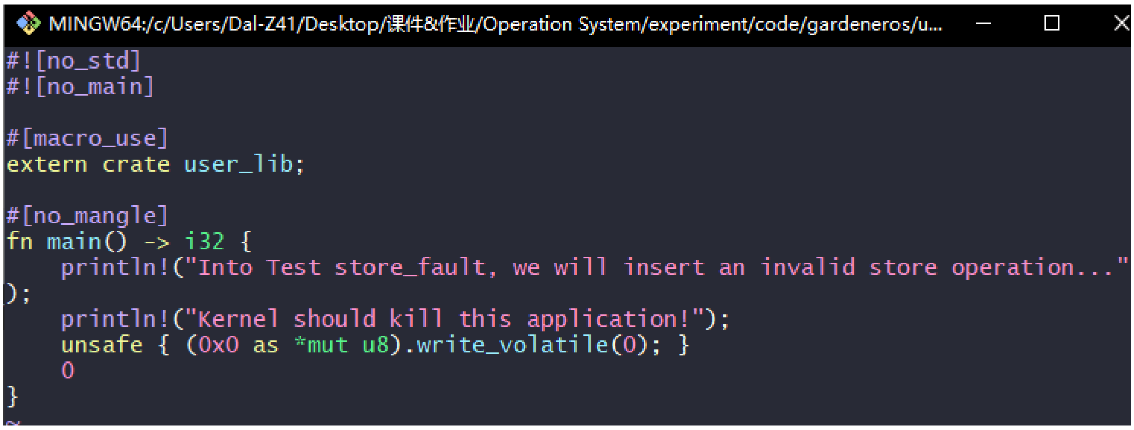
1.7 Multiple programs

- **00hello_world.rs :**



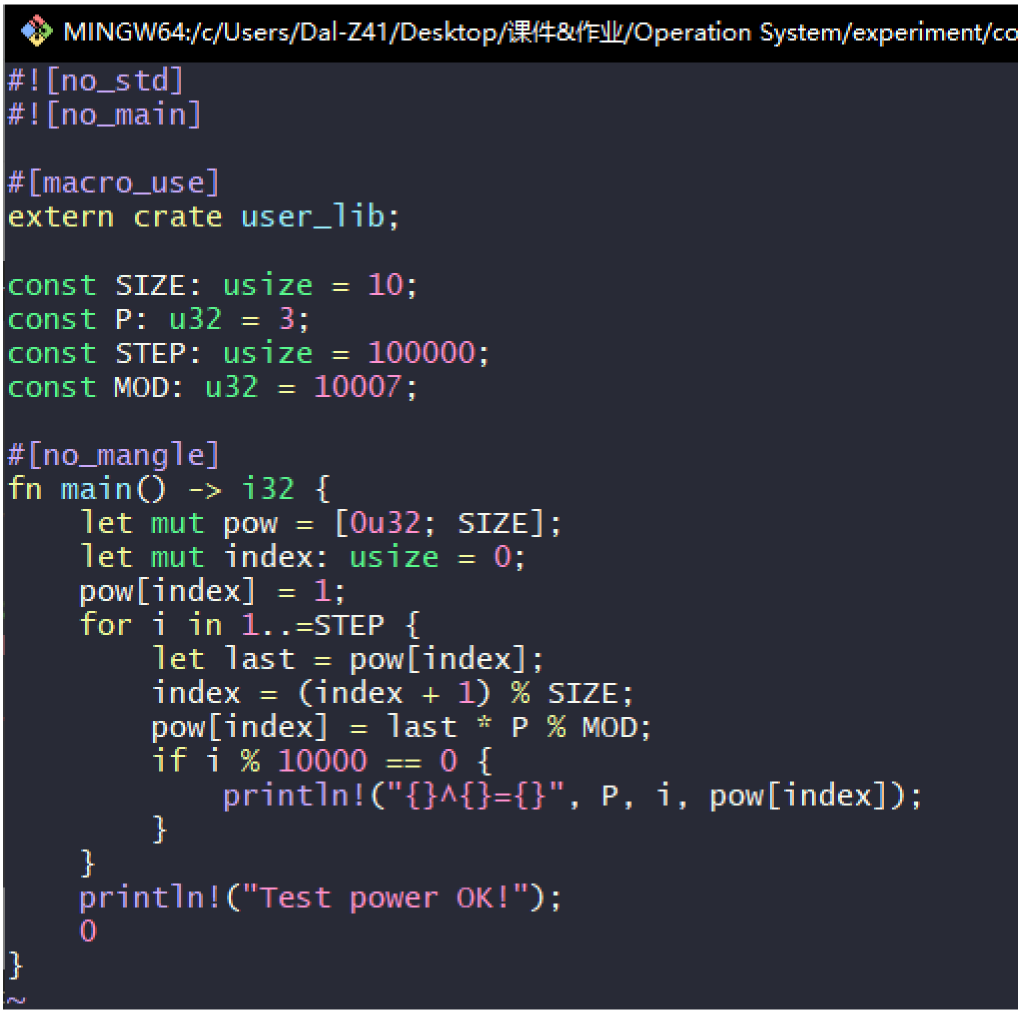
```rust
#![no_std]
#![no_main]
#![feature(asm)]

#[macro_use]
extern crate user_lib;

#[no_mangle]
fn main() -> i32 {
    println!("Hello, world!");
    unsafe {
        asm!("sret");
    }
    0
}
~
~
```

- **01store_fault.rs :**


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/u...

```rust
#![no_std]
#![no_main]

#[macro_use]
extern crate user_lib;

#[no_mangle]
fn main() -> i32 {
    println!("Into Test store_fault, we will insert an invalid store operation..."
);
    println!("Kernel should kill this application!");
    unsafe { (0x0 as *mut u8).write_volatile(0); }
    0
}
```

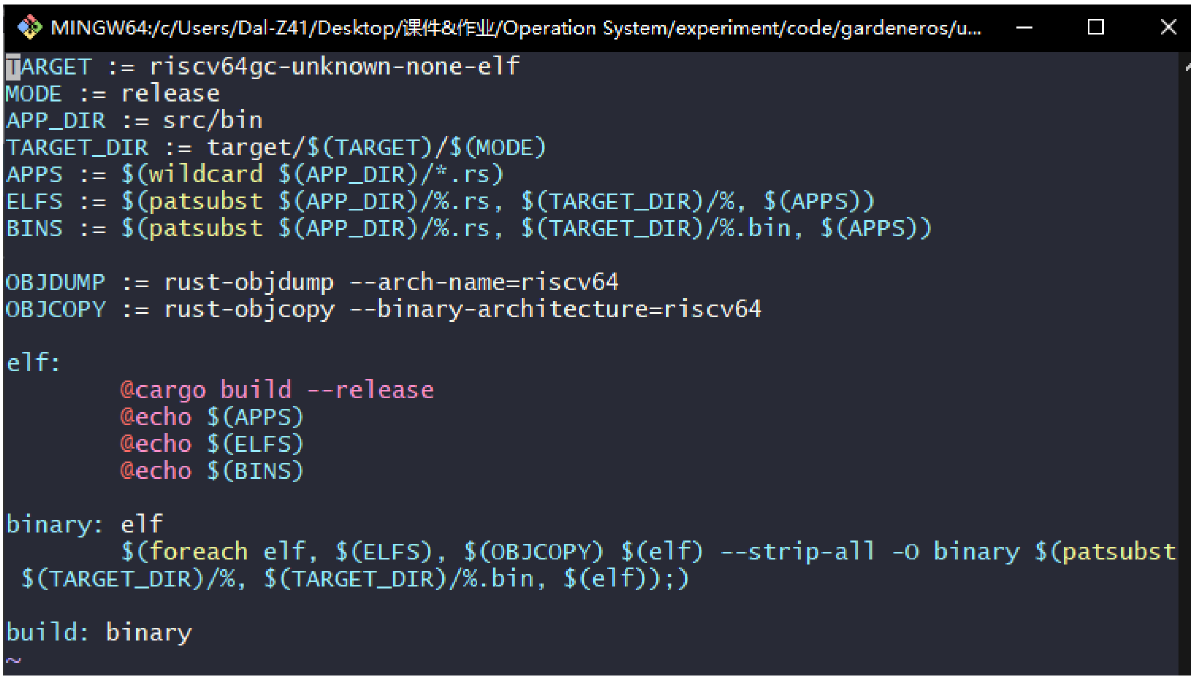
- **02power.rs :**


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/co

```rust
#![no_std]
#![no_main]

#[macro_use]
extern crate user_lib;

const SIZE: usize = 10;
const P: u32 = 3;
const STEP: usize = 100000;
const MOD: u32 = 10007;

#[no_mangle]
fn main() -> i32 {
    let mut pow = [0u32; SIZE];
    let mut index: usize = 0;
    pow[index] = 1;
    for i in 1..=STEP {
        let last = pow[index];
        index = (index + 1) % SIZE;
        pow[index] = last * P % MOD;
        if i % 10000 == 0 {
            println!("{}^{}={}", P, i, pow[index]);
        }
    }
    println!("Test power OK!");
    0
}
~
```

1.8 Makefile

File "user/Makefile" is as below.

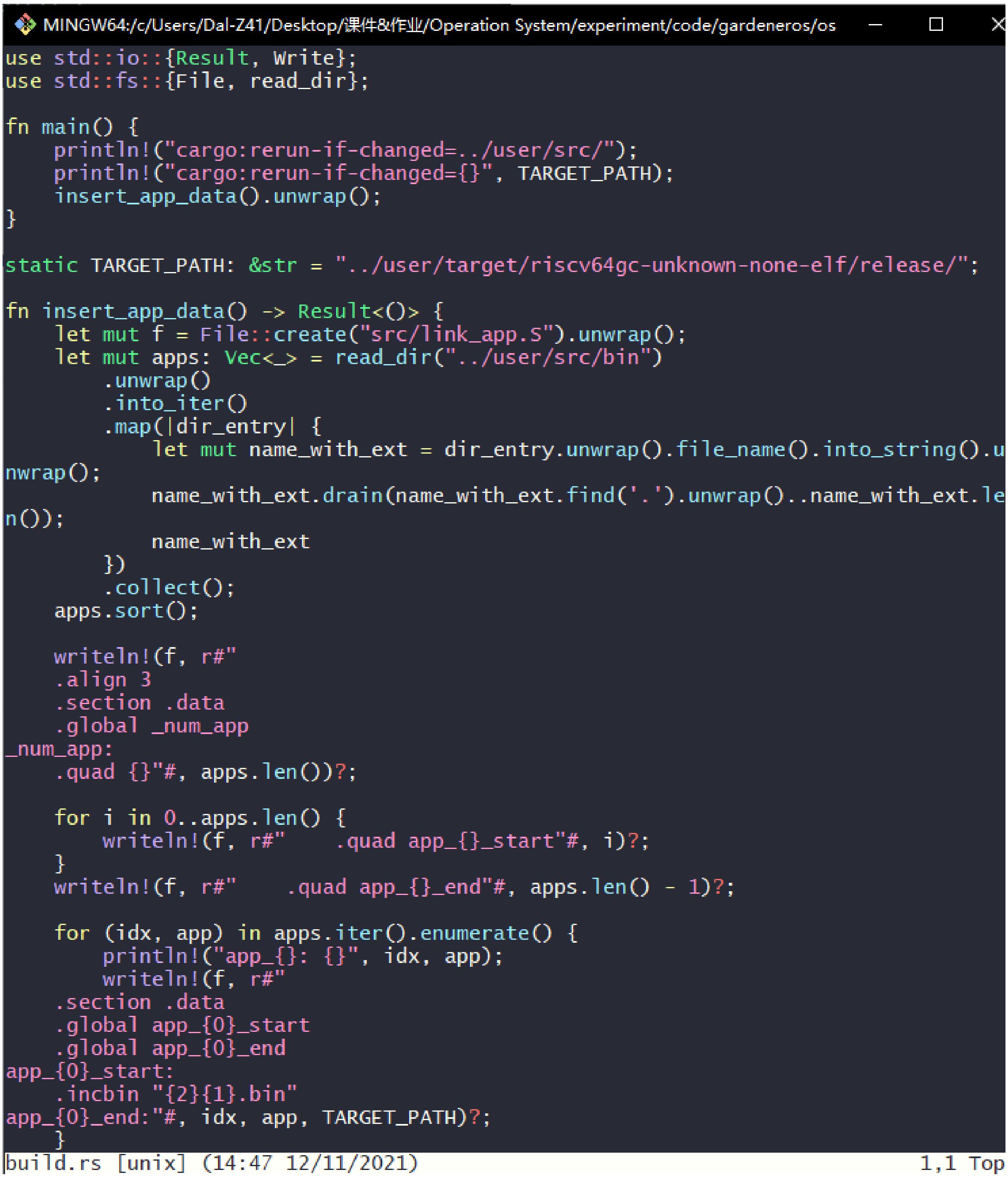

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/u...

```makefile
TARGET := riscv64gc-unknown-none-elf
MODE := release
APP_DIR := src/bin
TARGET_DIR := target/$(TARGET)/$(MODE)
APPS := $(wildcard $(APP_DIR)/*.rs)
ELFS := $(patsubst $(APP_DIR)/%.rs, $(TARGET_DIR)/%, $(APPS))
BINS := $(patsubst $(APP_DIR)/%.rs, $(TARGET_DIR)/%.bin, $(APPS))

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

elf:
	@cargo build --release
	@echo $(APPS)
	@echo $(ELFS)
	@echo $(BINS)

binary: elf
	$(foreach elf, $(ELFS), $(OBJCOPY) $(elf) --strip-all -O binary $(patsubst $(TARGET_DIR)/%, $(TARGET_DIR)/%.bin, $(elf));)

build: binary
~
```

2. 内核链接

To dynamically link the compiled application binaries into the content, we need to write a compilation script **os /build.rs** to generate the **link_app.S** script file specifically for linking. The details in file "build.rs" is as below.

```
use std::io::{Result, Write};
use std::fs::{File, read_dir};

fn main() {
    println!("cargo:rerun-if-changed=../user/src/");
    println!("cargo:rerun-if-changed={}", TARGET_PATH);
    insert_app_data().unwrap();
}

static TARGET_PATH: &str = "../user/target/riscv64gc-unknown-none-elf/release/";

fn insert_app_data() -> Result<()> {
    let mut f = File::create("src/link_app.S").unwrap();
    let mut apps: Vec<_> = read_dir("../user/src/bin")
        .unwrap()
        .into_iter()
        .map(|dir_entry| {
            let mut name_with_ext =
dir_entry.unwrap().file_name().into_string().unwrap();

 name_with_ext.drain(name_with_ext.find('.').unwrap()..name_with_ext.len());
            name_with_ext
        })
        .collect();
    apps.sort();

    writeln!(f, r#"
    .align 3
    .section .data
    .global _num_app
_num_app:
    .quad {}"#, apps.len())?;

    for i in 0..apps.len() {
        writeln!(f, r#"    .quad app_{}_start"#, i)?;
    }
    writeln!(f, r#"    .quad app_{}_end"#, apps.len() - 1)?;

    for (idx, app) in apps.iter().enumerate() {
        println!("app_{}: {}", idx, app);
        writeln!(f, r#"
    .section .data
    .global app_{0}_start
    .global app_{0}_end
app_{0}_start:
    .incbin "{2}{1}.bin"
app_{0}_end:"#, idx, app, TARGET_PATH)?;
    }
    Ok(())
}
```



MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/os

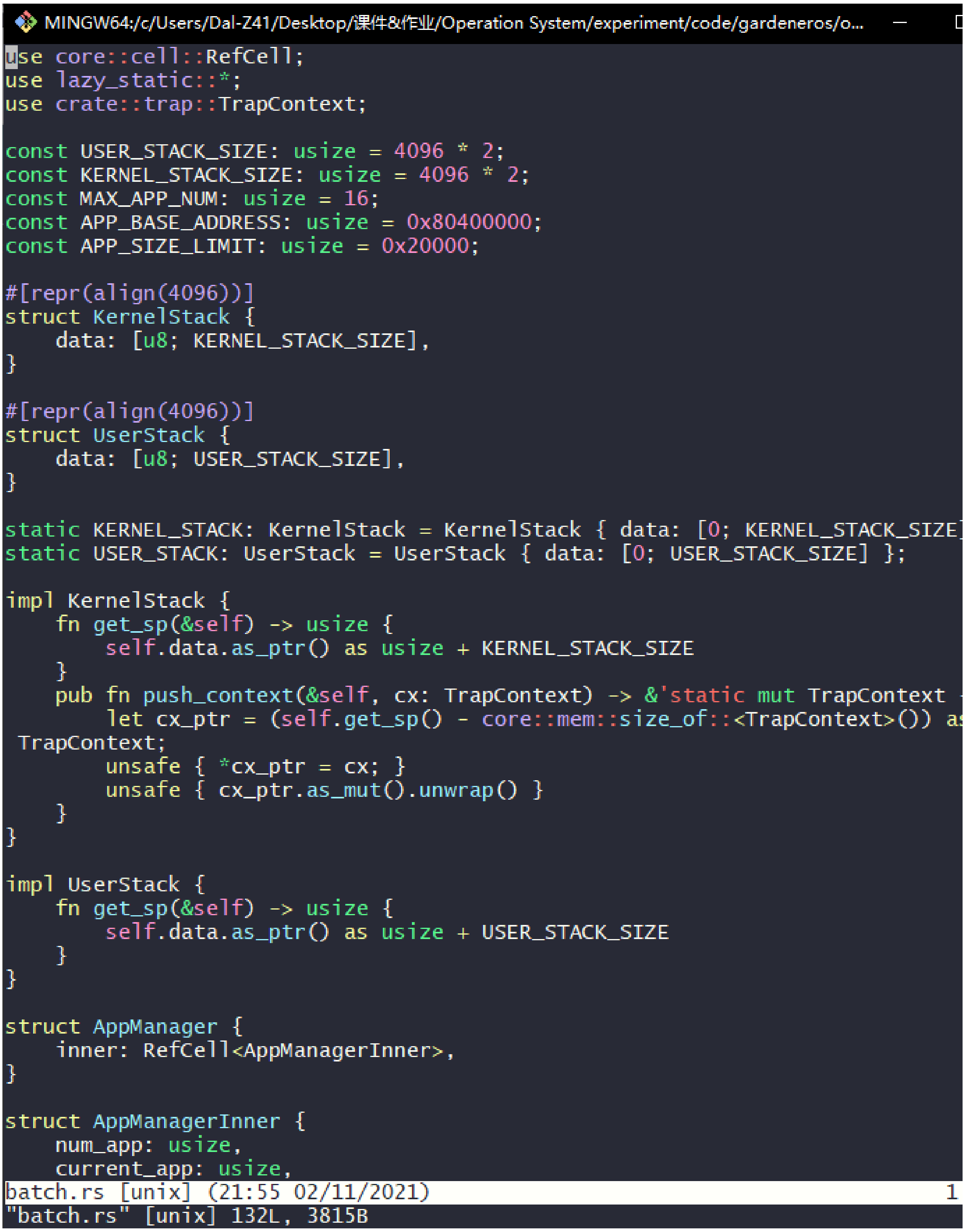
```
use std::io::{Result, Write};
use std::fs::{File, read_dir};

fn main() {
    println!("cargo:rerun-if-changed=../user/src/");
    println!("cargo:rerun-if-changed={}", TARGET_PATH);
    insert_app_data().unwrap();
}

static TARGET_PATH: &str = "../user/target/riscv64gc-unknown-none-elf/release/";

fn insert_app_data() -> Result<()> {
    let mut f = File::create("src/link_app.S").unwrap();
    let mut apps: Vec<_> = read_dir("../user/src/bin")
        .unwrap()
        .into_iter()
        .map(|dir_entry| {
            let mut name_with_ext = dir_entry.unwrap().file_name().into_string().u
nwrap();
            name_with_ext.drain(name_with_ext.find('.').unwrap()..name_with_ext.le
n());
            name_with_ext
        })
        .collect();
    apps.sort();

    writeln!(f, r#"
    .align 3
    .section .data
    .global _num_app
_num_app:
    .quad {}"#, apps.len())?;

    for i in 0..apps.len() {
        writeln!(f, r#"    .quad app_{}_start"#, i)?;
    }
    writeln!(f, r#"    .quad app_{}_end"#, apps.len() - 1)?;

    for (idx, app) in apps.iter().enumerate() {
        println!("app_{}: {}", idx, app);
        writeln!(f, r#"
    .section .data
    .global app_{0}_start
    .global app_{0}_end
app_{0}_start:
    .incbin "{2}{1}.bin"
app_{0}_end:"#, idx, app, TARGET_PATH)?;
    }
build.rs [unix] (14:47 12/11/2021)                                   1,1 Top
```

3. 加载程序二进制码

To implement a batch operating system, we implement a Batch submodule in the OS directory. Its main function is to save the application data and the corresponding location information, as well as the current implementation of the number of applications. At the same time, the memory required by the application is initialized and the execution application is loaded. Hence, here we need a file **"batch.rs"**.

```
use core::cell::RefCell;
use lazy_static::*;

const MAX_APP_NUM: usize = 16;
const APP_BASE_ADDRESS: usize = 0x80400000;
const APP_SIZE_LIMIT: usize = 0x20000;
```

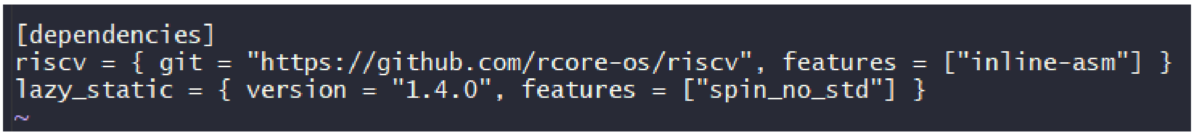
```
struct AppManager {
    inner: RefCell<AppManagerInner>,
}

struct AppManagerInner {
    num_app: usize,
    current_app: usize,
    app_start: [usize; MAX_APP_NUM + 1],
}

unsafe impl Sync for AppManager {}

impl AppManagerInner {
    pub fn print_app_info(&self) {
        println!("[kernel] num_app = {}", self.num_app);
        for i in 0..self.num_app {
            println!("[kernel] app_{} [{:#x}, {:#x})", i, self.app_start[i],
self.app_start[i + 1]);
        }
    }

    unsafe fn load_app(&self, app_id: usize) {
        if app_id >= self.num_app {
            panic!("All applications completed!");
        }

        println!("[kernel] Loading app_{}", app_id);
        // clear icache
        asm!("fence.i");
        // clear app area
        (APP_BASE_ADDRESS..APP_BASE_ADDRESS + APP_SIZE_LIMIT).for_each(|addr| {
            (addr as *mut u8).write_volatile(0);
        });
        let app_src = core::slice::from_raw_parts(
            self.app_start[app_id] as *const u8,
            self.app_start[app_id + 1] - self.app_start[app_id]
        );
        let app_dst = core::slice::from_raw_parts_mut(
            APP_BASE_ADDRESS as *mut u8,
            app_src.len()
        );
        app_dst.copy_from_slice(app_src);
    }

    pub fn get_current_app(&self) -> usize { self.current_app }

    pub fn move_to_next_app(&mut self) {
        self.current_app += 1;
    }
}

lazy_static! {
    static ref APP_MANAGER: AppManager = AppManager {
        inner: RefCell::new({
            extern "C" { fn _num_app(); }
            let num_app_ptr = _num_app as usize as *const usize;
            let num_app = unsafe { num_app_ptr.read_volatile() };
            let mut app_start: [usize; MAX_APP_NUM + 1] = [0; MAX_APP_NUM + 1];
            let app_start_raw: &[usize] = unsafe {
                core::slice::from_raw_parts(num_app_ptr.add(1), num_app + 1)
            };
            app_start[..=num_app].copy_from_slice(app_start_raw);
            AppManagerInner {
                num_app,
                current_app: 0,
                app_start,
            }
        }),
    };
}

pub fn init() {
    print_app_info();
}

pub fn print_app_info() {
    APP_MANAGER.inner.borrow().print_app_info();
}

pub fn run_next_app() -> ! {
    let current_app = APP_MANAGER.inner.borrow().get_current_app();
    unsafe {
        APP_MANAGER.inner.borrow().load_app(current_app);
    }

    APP_MANAGER.inner.borrow_mut().move_to_next_app();
    extern "C" { fn __restore(cx_addr: usize); }
    unsafe {
        __restore(KERNEL_STACK.push_context(
            TrapContext::app_init_context(APP_BASE_ADDRESS, USER_STACK.get_sp())
        ) as *const _ as usize);
    }
    panic!("Unreachable in batch::run_current_app!");
}
```


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/o...

```
use core::cell::RefCell;
use lazy_static::*;
use crate::trap::TrapContext;

const USER_STACK_SIZE: usize = 4096 * 2;
const KERNEL_STACK_SIZE: usize = 4096 * 2;
const MAX_APP_NUM: usize = 16;
const APP_BASE_ADDRESS: usize = 0x80400000;
const APP_SIZE_LIMIT: usize = 0x20000;

#[repr(align(4096))]
struct KernelStack {
    data: [u8; KERNEL_STACK_SIZE],
}

#[repr(align(4096))]
struct UserStack {
    data: [u8; USER_STACK_SIZE],
}

static KERNEL_STACK: KernelStack = KernelStack { data: [0; KERNEL_STACK_SIZE]
static USER_STACK: UserStack = UserStack { data: [0; USER_STACK_SIZE] };

impl KernelStack {
    fn get_sp(&self) -> usize {
        self.data.as_ptr() as usize + KERNEL_STACK_SIZE
    }
    pub fn push_context(&self, cx: TrapContext) -> &'static mut TrapContext
        let cx_ptr = (self.get_sp() - core::mem::size_of::<TrapContext>()) as
 TrapContext;
        unsafe { *cx_ptr = cx; }
        unsafe { cx_ptr.as_mut().unwrap() }
    }
}

impl UserStack {
    fn get_sp(&self) -> usize {
        self.data.as_ptr() as usize + USER_STACK_SIZE
    }
}

struct AppManager {
    inner: RefCell<AppManagerInner>,
}

struct AppManagerInner {
    num_app: usize,
    current_app: usize,
batch.rs [unix] (21:55 02/11/2021)                                  1
"batch.rs" [unix] 132L, 3815B
```

Since we use lazy_static provided by the external library macro **lazy_static!**, we need to add dependencies to **"cargo.toml"**. Macro lazy_static! provide runtime initialization of global variables with the help of lazy_static! A global instance of the **AppManager** structure, APP_MANAGER, is declared so that the actual initialization takes place only when it is first used. Hence, we need to modify config file "cargo.toml" and add dependencies to it.

```
lazy_static = { version = "1.4.0", features = ["spin_no_std"] }
```

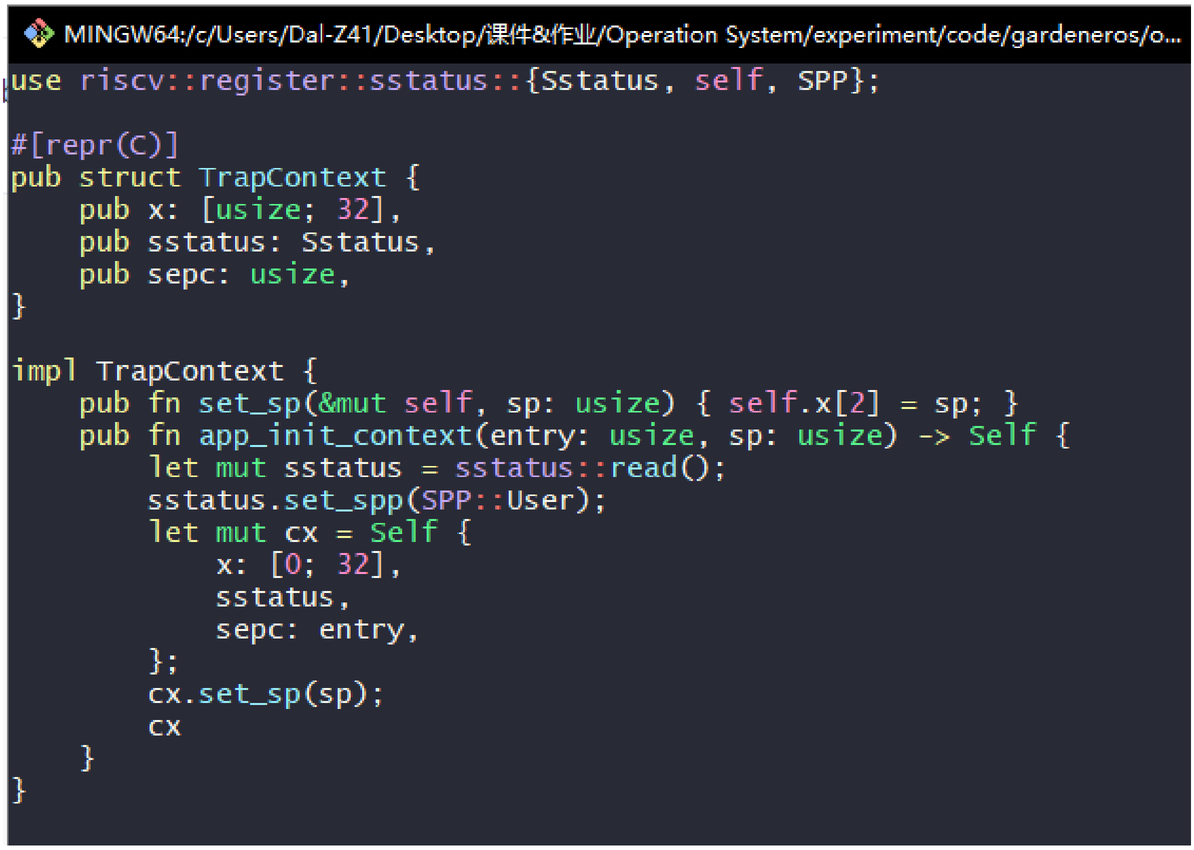


```
[dependencies]
riscv = { git = "https://github.com/rcore-os/riscv", features = ["inline-asm"] }
lazy_static = { version = "1.4.0", features = ["spin_no_std"] }
~
```

4. 实现用户栈与内核栈

In order to switch privilege levels, user **stacks** and **kernel stacks** need to be implemented. Hence ,we need to add the following implementation to file **"batch.rs"**.

```
use crate::trap::TrapContext;

const USER_STACK_SIZE: usize = 4096 * 2;
const KERNEL_STACK_SIZE: usize = 4096 * 2;

#[repr(align(4096))]
struct KernelStack {
    data: [u8; KERNEL_STACK_SIZE],
}

#[repr(align(4096))]
struct UserStack {
    data: [u8; USER_STACK_SIZE],
}

static KERNEL_STACK: KernelStack = KernelStack { data: [0; KERNEL_STACK_SIZE] };
static USER_STACK: UserStack = UserStack { data: [0; USER_STACK_SIZE] };

impl KernelStack {
    fn get_sp(&self) -> usize {
        self.data.as_ptr() as usize + KERNEL_STACK_SIZE
    }
    pub fn push_context(&self, cx: TrapContext) -> &'static mut TrapContext {
        let cx_ptr = (self.get_sp() - core::mem::size_of::<TrapContext>()) as
*mut TrapContext;
        unsafe { *cx_ptr = cx; }
        unsafe { cx_ptr.as_mut().unwrap() }
    }
}

impl UserStack {
    fn get_sp(&self) -> usize {
        self.data.as_ptr() as usize + USER_STACK_SIZE
    }
}
```


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/o...

```
use riscv::register::sstatus::{Sstatus, self, SPP};

#[repr(C)]
pub struct TrapContext {
    pub x: [usize; 32],
    pub sstatus: Sstatus,
    pub sepc: usize,
}

impl TrapContext {
    pub fn set_sp(&mut self, sp: usize) { self.x[2] = sp; }
    pub fn app_init_context(entry: usize, sp: usize) -> Self {
        let mut sstatus = sstatus::read();
        sstatus.set_spp(SPP::User);
        let mut cx = Self {
            x: [0; 32],
            sstatus,
            sepc: entry,
        };
        cx.set_sp(sp);
        cx
    }
}
```

5. 实现trap管理

The process of Trap processing is as follows: First, save the Trap context on the kernel stack, and then switch to the Trap handler function to complete the distribution and processing of Trap. When the handler returns, the registers are recovered from the Trap context stored on the kernel stack. Finally, an SRET instruction returns to the application to continue execution.

5.1 Context Store and Resume

First, modify the **STVEC** register to point to the correct Trap processing entry point. Hence, we need to modify file **"mod.rs"**.

```
global_asm!(include_str!("trap.S"));

pub fn init() {
    extern "C" { fn __alltraps(); }
    unsafe {
        stvec::write(__alltraps as usize, TrapMode::Direct);
    }
}
```

5.2 Trap Distribution and Handling

We can use "trap_handler" to implement the distribution and handling of **"trap"**. And since we introduce library **"nscv"**, we need to add dependencies to file "cargo.toml".

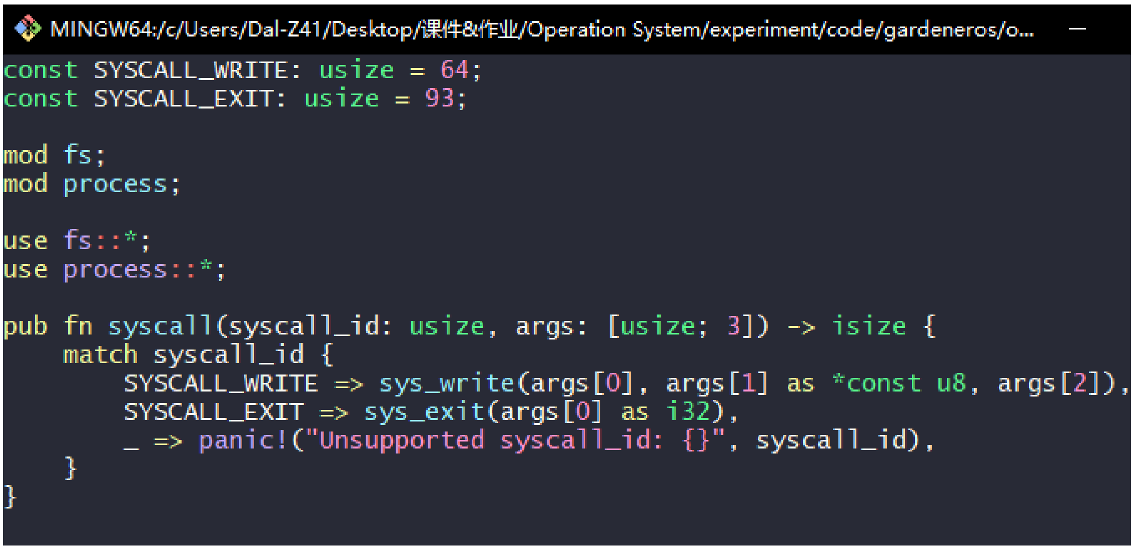
```
// os/src/trap/mod.rs

mod context;
```

```
use riscv::register::{
    mtvec::TrapMode,
    stvec,
    scause::{
        self,
        Trap,
        Exception,
    },
    stval,
};

use crate::syscall::syscall;
use crate::batch::run_next_app;

#[no_mangle]
pub fn trap_handler(cx: &mut TrapContext) -> &mut TrapContext {
    let scause = scause::read();
    let stval = stval::read();
    match scause.cause() {
        Trap::Exception(Exception::UserEnvCall) => {
            cx.sepc += 4;
            cx.x[10] = syscall(cx.x[17], [cx.x[10], cx.x[11], cx.x[12]]) as
usize;
        }
        Trap::Exception(Exception::StoreFault) |
        Trap::Exception(Exception::StorePageFault) => {
            println!("[kernel] PageFault in application, core dumped.");
            run_next_app();
        }
        Trap::Exception(Exception::IllegalInstruction) => {
            println!("[kernel] IllegalInstruction in application, core
dumped.");
            run_next_app();
        }
        _ => {
            panic!("Unsupported trap {:?}, stval = {:#x}!", scause.cause(),
stval);
        }
    }
    cx
}

pub use context::TrapContext;

// Add dependency
riscv = { git = "https://github.com/rcore-os/riscv", features = ["inline-asm"] }
```

5.3 System call handling

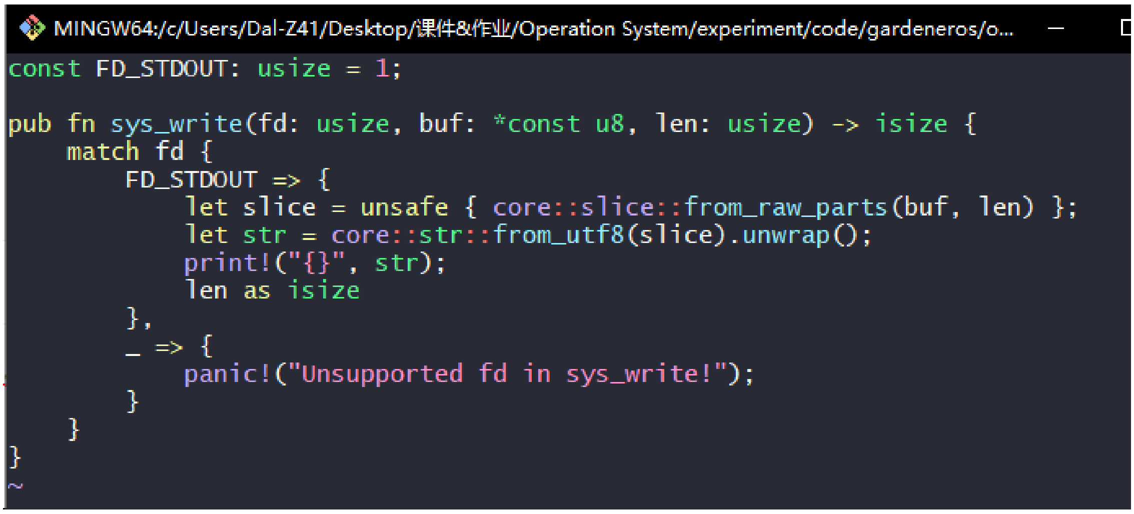
To implement system call processing, we also need to implement the Syscall module. The syscall function does not actually handle system calls, but rather is distributed to specific handlers based on the Syscall ID.

- **mod.rs :**


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/o...

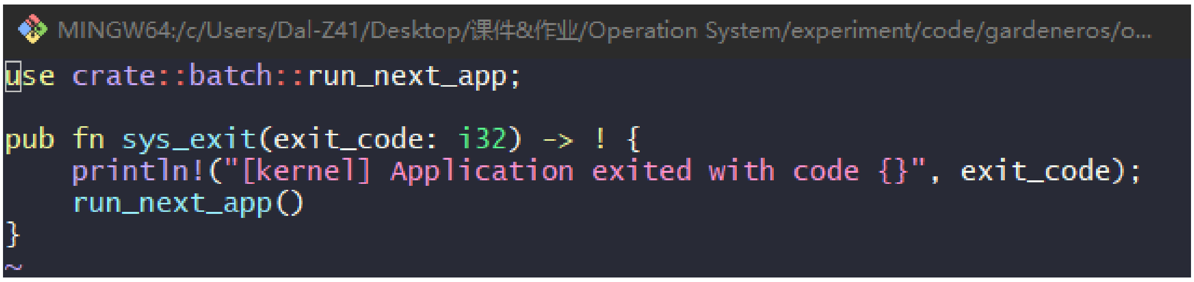
```
const SYSCALL_WRITE: usize = 64;
const SYSCALL_EXIT: usize = 93;

mod fs;
mod process;

use fs::*;
use process::*;

pub fn syscall(syscall_id: usize, args: [usize; 3]) -> isize {
    match syscall_id {
        SYSCALL_WRITE => sys_write(args[0], args[1] as *const u8, args[2]),
        SYSCALL_EXIT => sys_exit(args[0] as i32),
        _ => panic!("Unsupported syscall_id: {}", syscall_id),
    }
}
```

- **fs.rs :**


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/o...

```
const FD_STDOUT: usize = 1;

pub fn sys_write(fd: usize, buf: *const u8, len: usize) -> isize {
    match fd {
        FD_STDOUT => {
            let slice = unsafe { core::slice::from_raw_parts(buf, len) };
            let str = core::str::from_utf8(slice).unwrap();
            print!("{}", str);
            len as isize
        },
        _ => {
            panic!("Unsupported fd in sys_write!");
        }
    }
}
~
```

- **process.rs :**

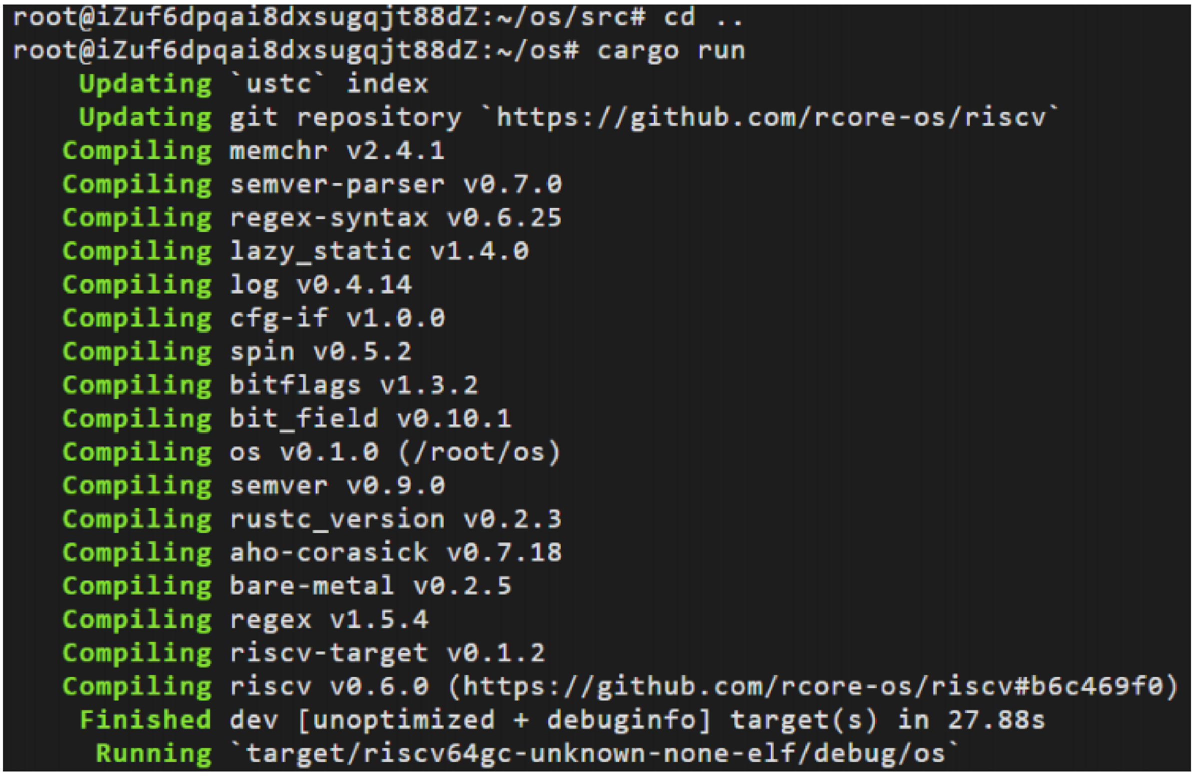

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/o...

```
use crate::batch::run_next_app;

pub fn sys_exit(exit_code: i32) -> ! {
    println!("[kernel] Application exited with code {}", exit_code);
    run_next_app()
}
~
```

6. 执行程序

Before executing the application, jump to the application entry 0x80400000, switch to the user stack, set **sscratch** to point to the kernel stack, and switch to the **U** privilege level with the **S** privilege level. We can do this by reusing **__restore** code. In this way, a Trap file specially constructed to start the application is pushed onto the kernel stack, and the context state needed to start the application can be stored via the __restore function. Hence, we need to implement "app_init_context" for TrapContext.

```
// os/src/trap/context.rs
use riscv::register::sstatus::{Sstatus, self, SPP};

impl TrapContext {
    pub fn set_sp(&mut self, sp: usize) { self.x[2] = sp; }
    pub fn app_init_context(entry: usize, sp: usize) -> Self {
        let mut sstatus = sstatus::read();
        sstatus.set_spp(SPP::User);
        let mut cx = Self {
            x: [0; 32],
            sstatus,
            sepc: entry,
        };
        cx.set_sp(sp);
        cx
    }
}
```



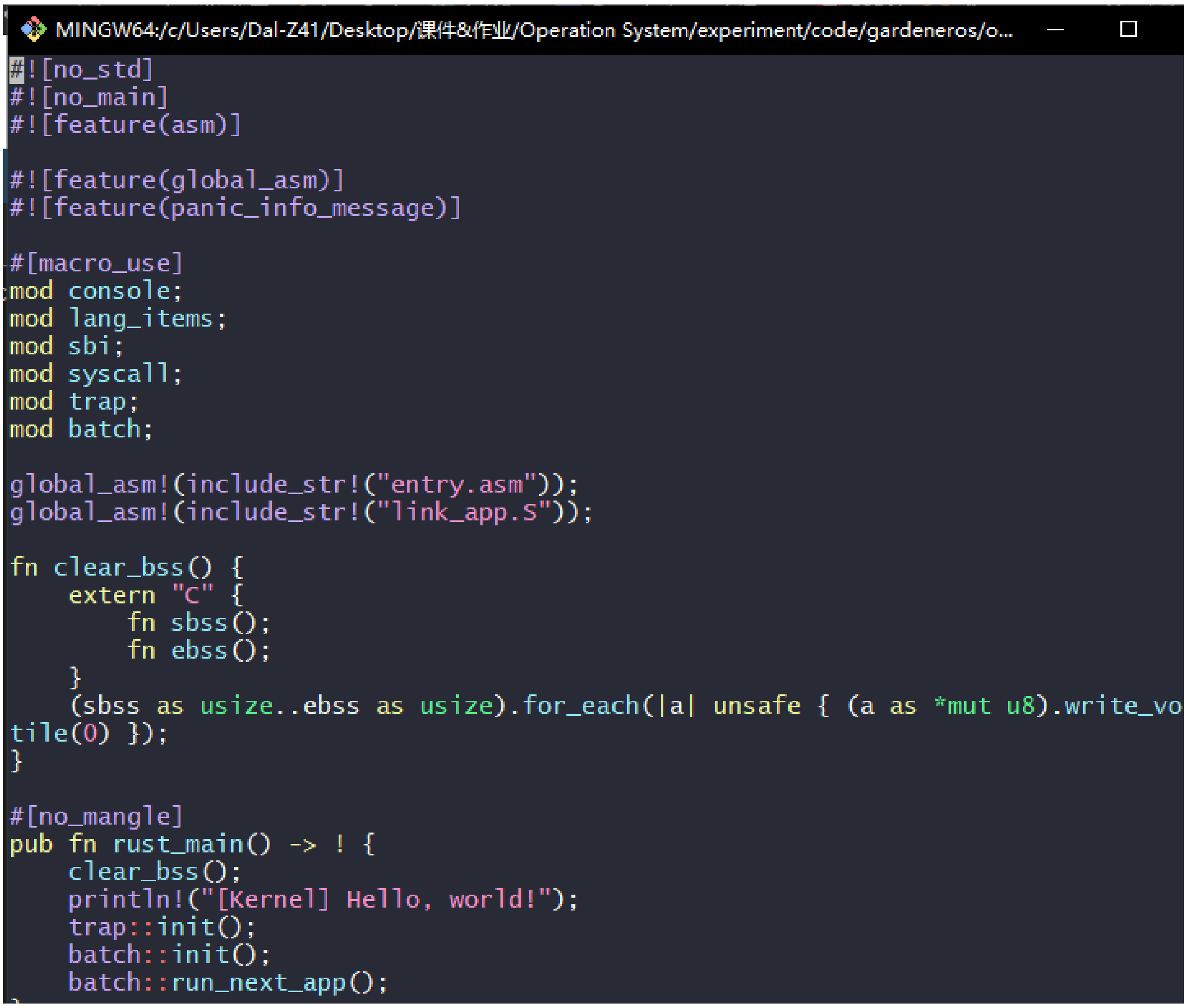
```
root@iZuf6dpqai8dxsugqjt88dZ:~/os/src# cd ..
root@iZuf6dpqai8dxsugqjt88dZ:~/os# cargo run
    Updating `ustc` index
    Updating git repository `https://github.com/rcore-os/riscv`
   Compiling memchr v2.4.1
   Compiling semver-parser v0.7.0
   Compiling regex-syntax v0.6.25
   Compiling lazy_static v1.4.0
   Compiling log v0.4.14
   Compiling cfg-if v1.0.0
   Compiling spin v0.5.2
   Compiling bitflags v1.3.2
   Compiling bit_field v0.10.1
   Compiling os v0.1.0 (/root/os)
   Compiling semver v0.9.0
   Compiling rustc_version v0.2.3
   Compiling aho-corasick v0.7.18
   Compiling bare-metal v0.2.5
   Compiling regex v1.5.4
   Compiling riscv-target v0.1.2
   Compiling riscv v0.6.0 (https://github.com/rcore-os/riscv#b6c469f0)
    Finished dev [unoptimized + debuginfo] target(s) in 27.88s
     Running `target/riscv64gc-unknown-none-elf/debug/os`
```

7. 批处理

Finally, modify "**main.rs**" to add the newly implemented module, and call the **batch** submodule to initialize and batch execute the application.

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

#[macro_use]
mod console;
mod lang_items;
mod sbi;
mod syscall;
mod trap;
mod batch;
```

```
global_asm!(include_str!("entry.asm"));
global_asm!(include_str!("link_app.S"));

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() -> ! {
    clear_bss();
    println!("[Kernel] Hello, world!");
    trap::init();
    batch::init();
    batch::run_next_app();
}
```


MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/o...

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

#![feature(global_asm)]
#![feature(panic_info_message)]

#[macro_use]
mod console;
mod lang_items;
mod sbi;
mod syscall;
mod trap;
mod batch;

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

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

#[no_mangle]
pub fn rust_main() -> ! {
    clear_bss();
    println!("[Kernel] Hello, world!");
    trap::init();
    batch::init();
    batch::run_next_app();
```