一. 实验步骤

1. 在内核中支持动态内存分配

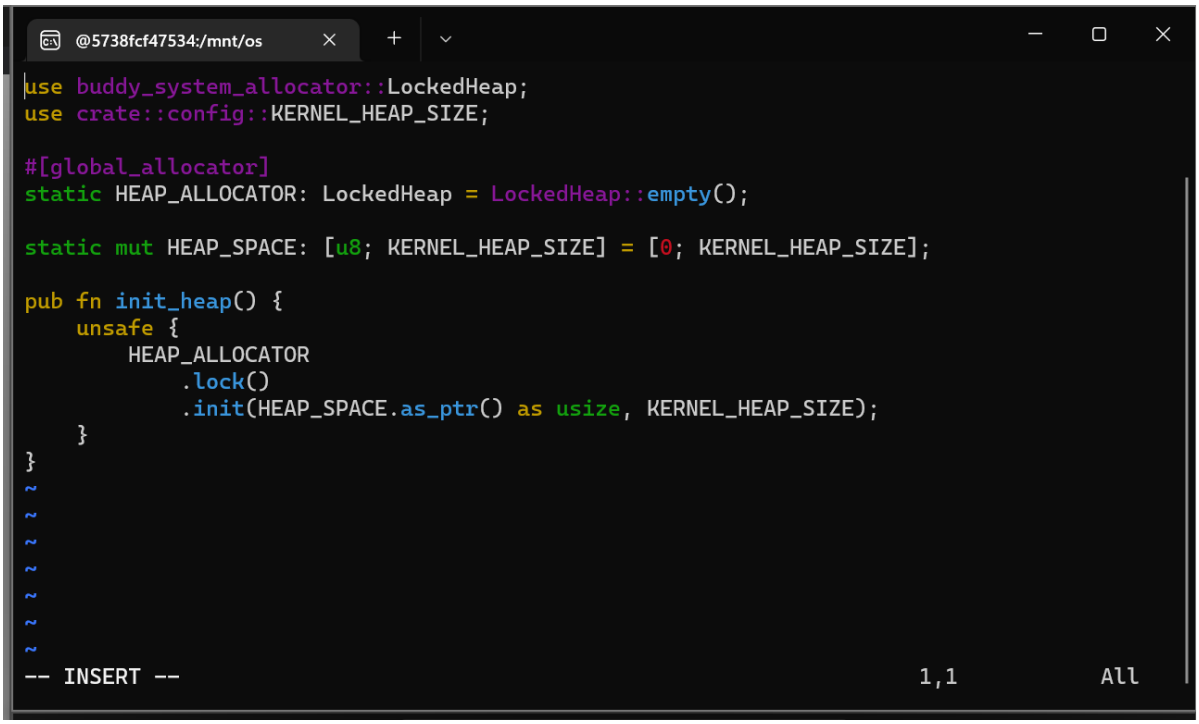
- 增加依赖

```
C:\ProgramData\Microsoft\W
[package]
name = "os"
version = "0.1.0"
edition = "2021"
buddy_system_allocator = "0.6"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/mani
fest.html

[dependencies]
lazy_static = { version = "1.4.0", features = ["spin_no_std"] }
riscv = { git = "https://github.com/rcore-os/riscv", features = ["inline-asm"] }
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                    5,31          All
```

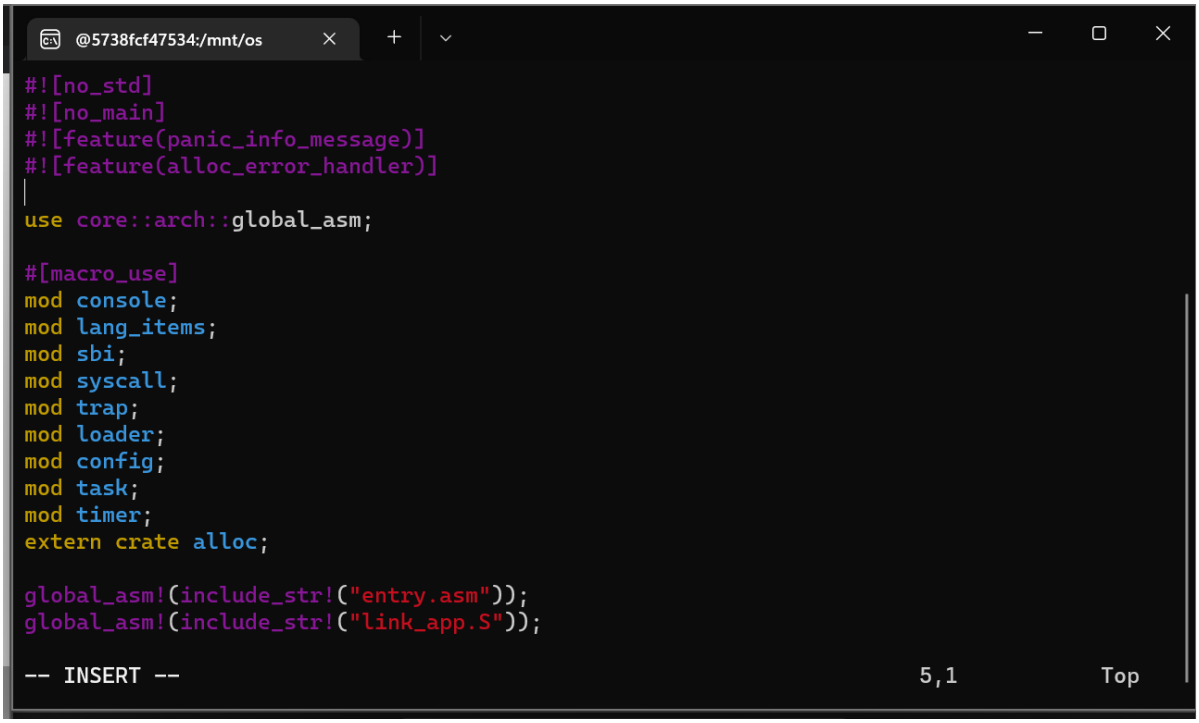
- 在`main.rs`中引入`alloc`依赖

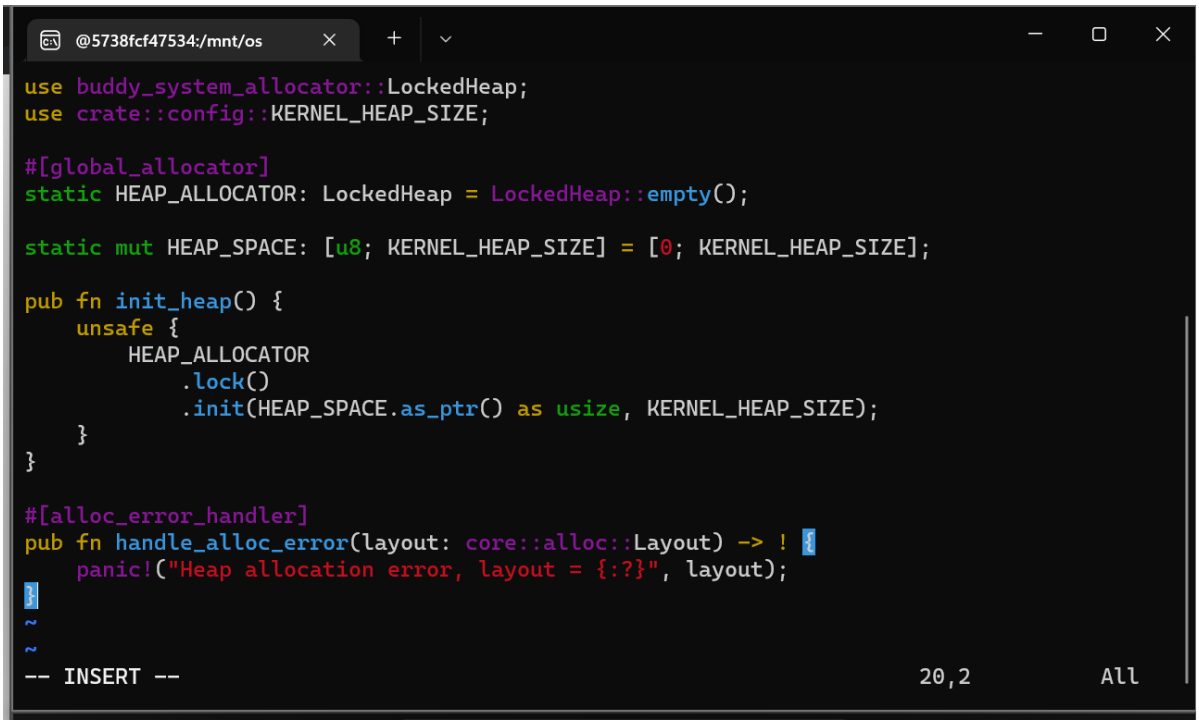
```
@5738fcf47534:/mnt/os
use core::arch::global_asm;

#[macro_use]
mod console;
mod lang_items;
mod sbi;
mod syscall;
mod trap;
mod loader;
mod config;
mod task;
mod timer;
extern crate alloc;

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

fn clear_bss() {
    extern "C" {
        fn sbss();
        fn ebss();
    }
-- INSERT --                                                    10,9          20%
```

- 实现全局动态内存分配器

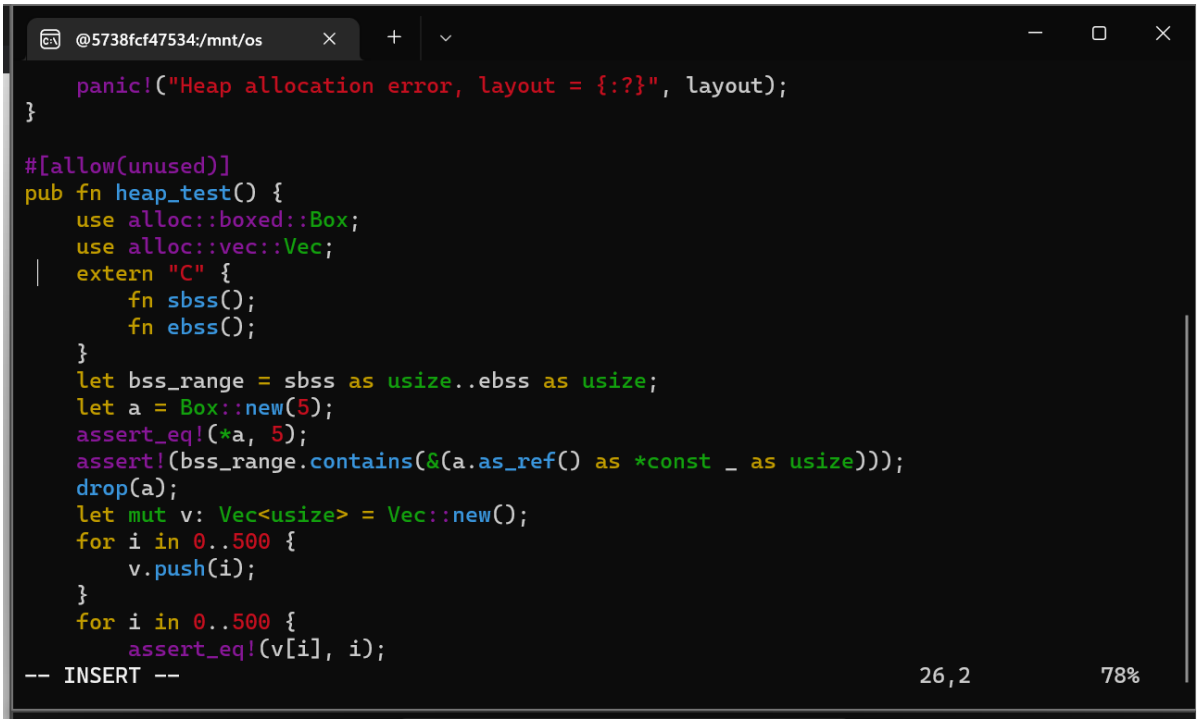


- 处理动态内存分配失败的情况

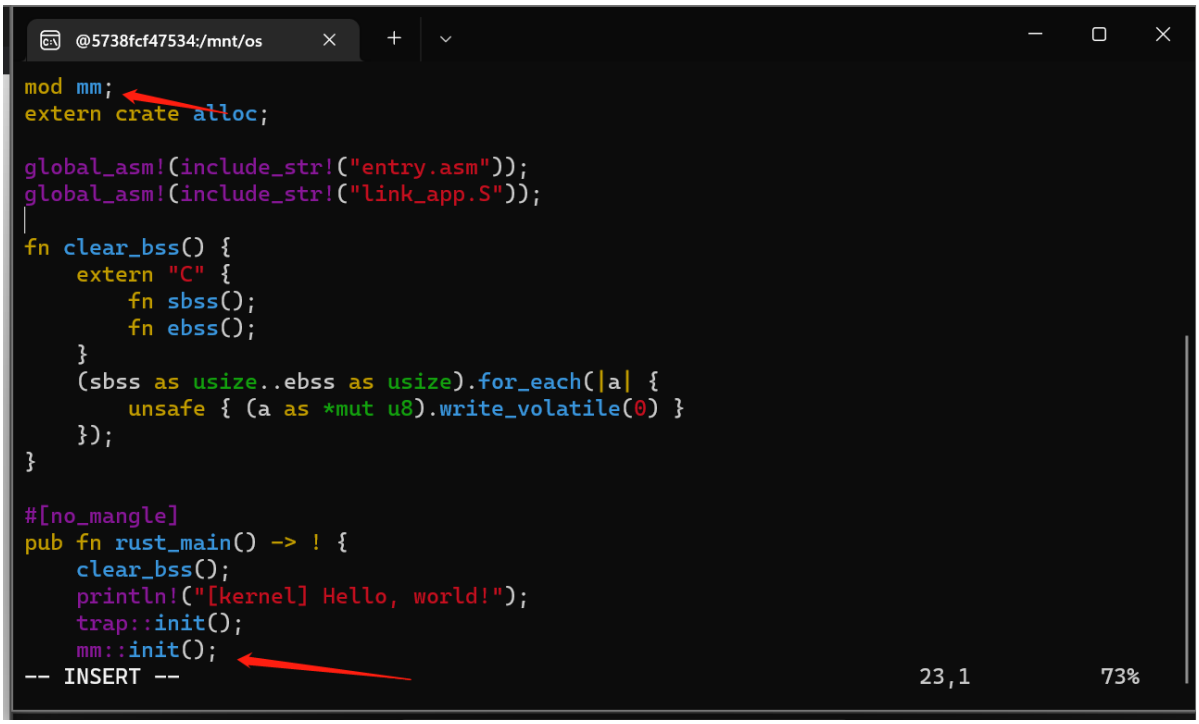




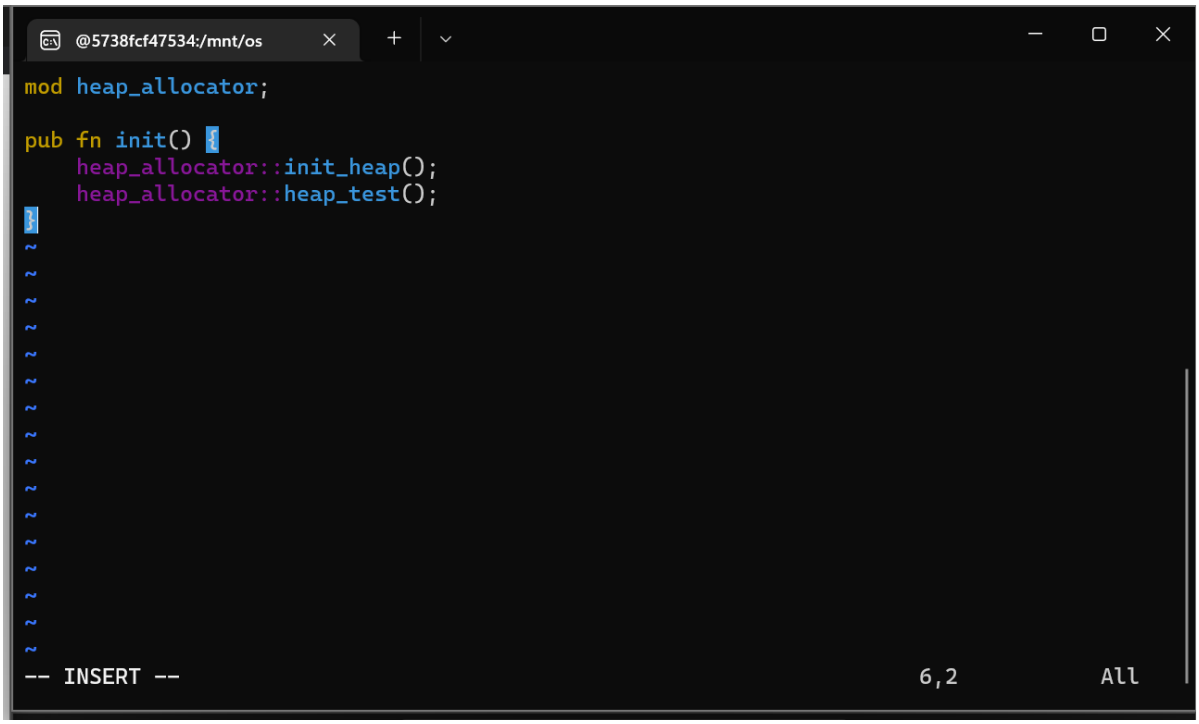
- 测试动态内存分配



- 修改`main.rs` `config.rs`中代码

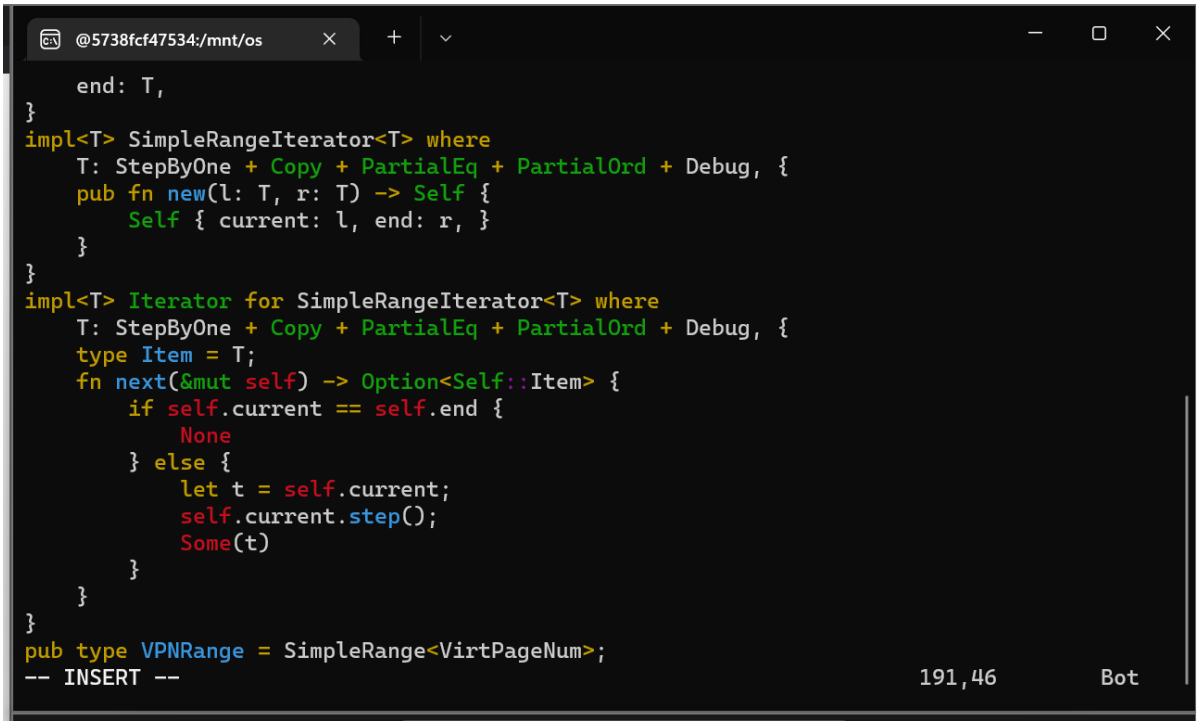


- 实现`os/src/mod.rs`



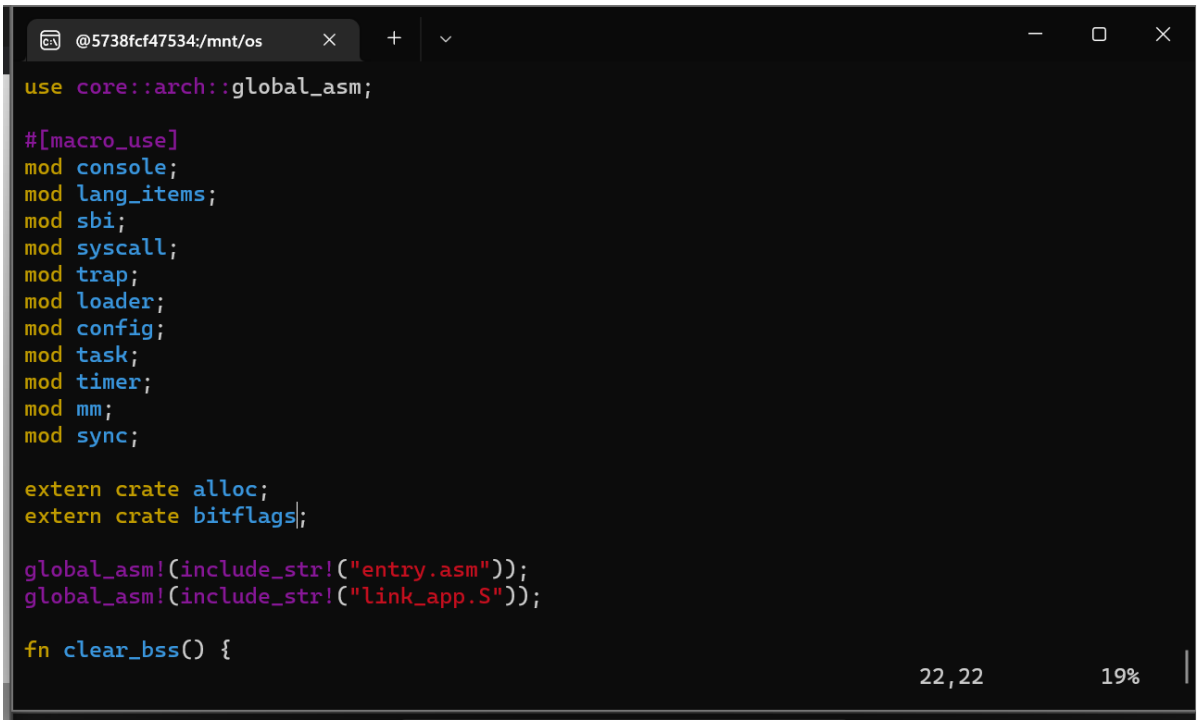
2. 实现虚拟地址与物理地址的基本定义

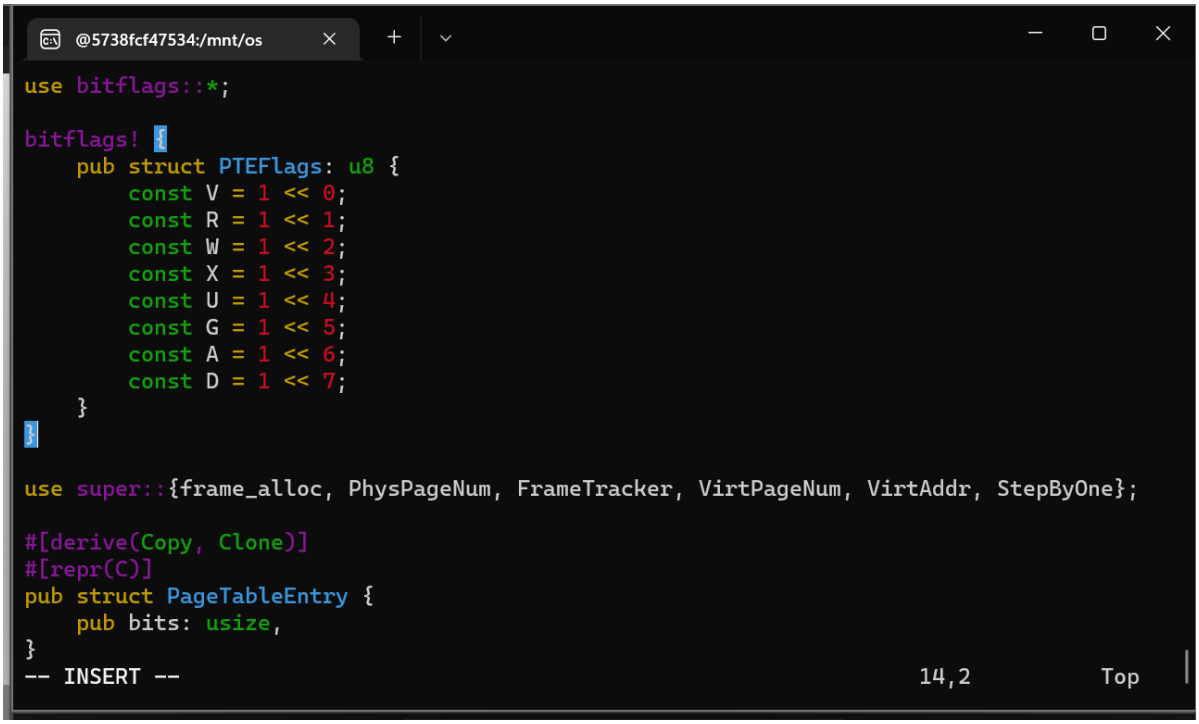
> 总之复制很多代码到`os/src/mm/address.rs`中，实现最基本的数据结构和相关操作



3. 定义页表项数据结构

- 实现标志位`PTEFlags`


@5738fcf47534:/mnt/os
use core::arch::global_asm;
#[macro_use]
mod console;
mod lang_items;
mod sbi;
mod syscall;
mod trap;
mod loader;
mod config;
mod task;
mod timer;
mod mm;
mod sync;
extern crate alloc;
extern crate bitflags;
global_asm!(include_str!("entry.asm"));
global_asm!(include_str!("link_app.S"));
fn clear_bss() {
22,22 19%


@5738fcf47534:/mnt/os
use bitflags::*;
bitflags! {
pub struct PTEFlags: u8 {
const V = 1 << 0;
const R = 1 << 1;
const W = 1 << 2;
const X = 1 << 3;
const U = 1 << 4;
const G = 1 << 5;
const A = 1 << 6;
const D = 1 << 7;
}
}
use super::{frame_alloc, PhysPageNum, FrameTracker, VirtPageNum, VirtAddr, StepByOne};
#[derive(Copy, Clone)]
#[repr(C)]
pub struct PageTableEntry {
pub bits: usize,
}
-- INSERT -- 14,2 Top

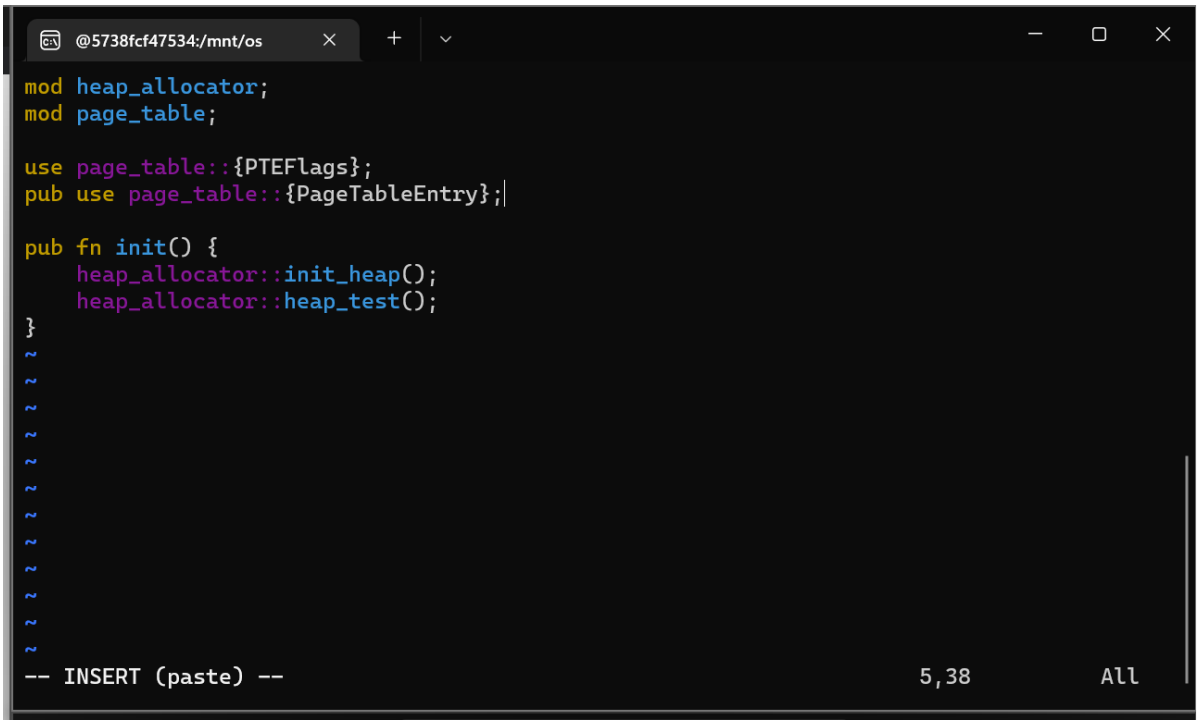
- 增加依赖

```
@5738fcf47534:/mnt/os
[package]
name = "os"
version = "0.1.0"
edition = "2021"
buddy_system_allocator = "0.6"
bitflags = "1.2.1"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
lazy_static = { version = "1.4.0", features = ["spin_no_std"] }
riscv = { git = "https://github.com/rcore-os/riscv", features = ["inline-asm"] }
~
~
~
~
~
~
~
~
~
-- INSERT --                                                    6,19          All
```

- 实现`PageTableEntry`

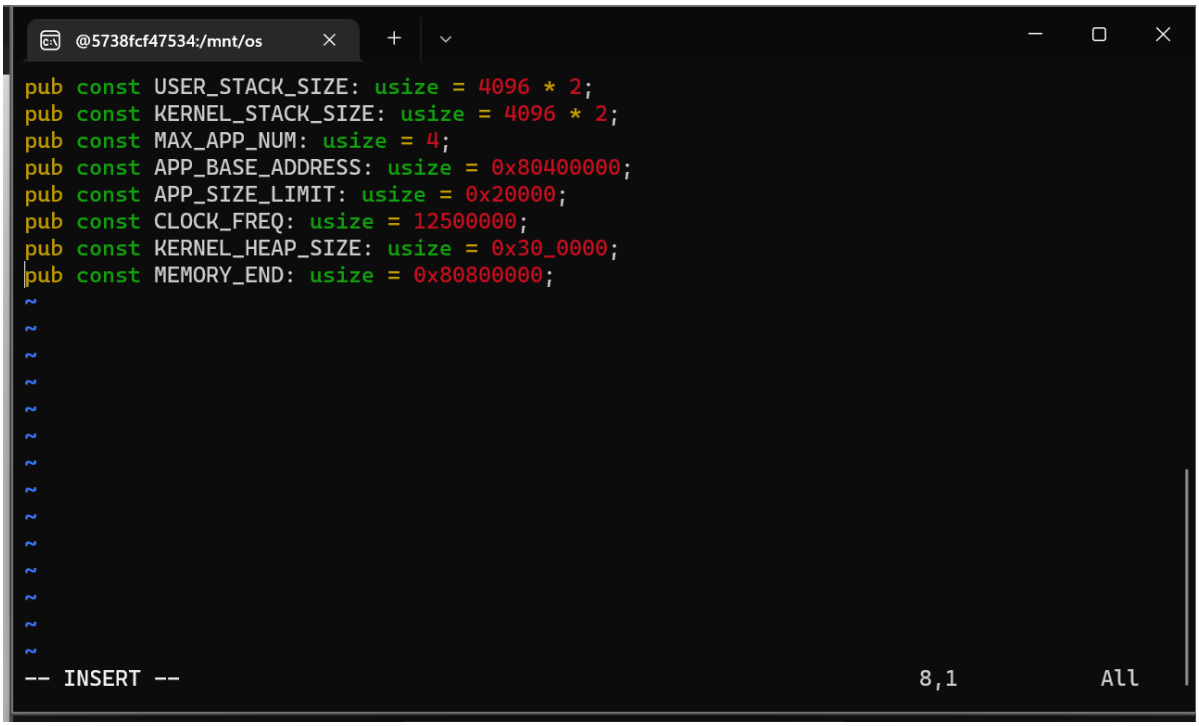
```
@5738fcf47534:/mnt/os
use super::{frame_alloc, PhysPageNum, FrameTracker, VirtPageNum, VirtAddr, StepByOne};

#[derive(Copy, Clone)]
#[repr(C)]
pub struct PageTableEntry {
    pub bits: usize,
}

impl PageTableEntry {
    pub fn new(ppn: PhysPageNum, flags: PTEFlags) -> Self {
        PageTableEntry {
            bits: ppn.0 << 10 | flags.bits as usize,
        }
    }
    pub fn empty() -> Self {
        PageTableEntry {
            bits: 0,
        }
    }
    pub fn ppn(&self) -> PhysPageNum {
        (self.bits >> 10 & ((1usize << 44) - 1)).into()
    }
                                                                 1,1           Top
```

- 修改`os/src/mm/mod.rs`



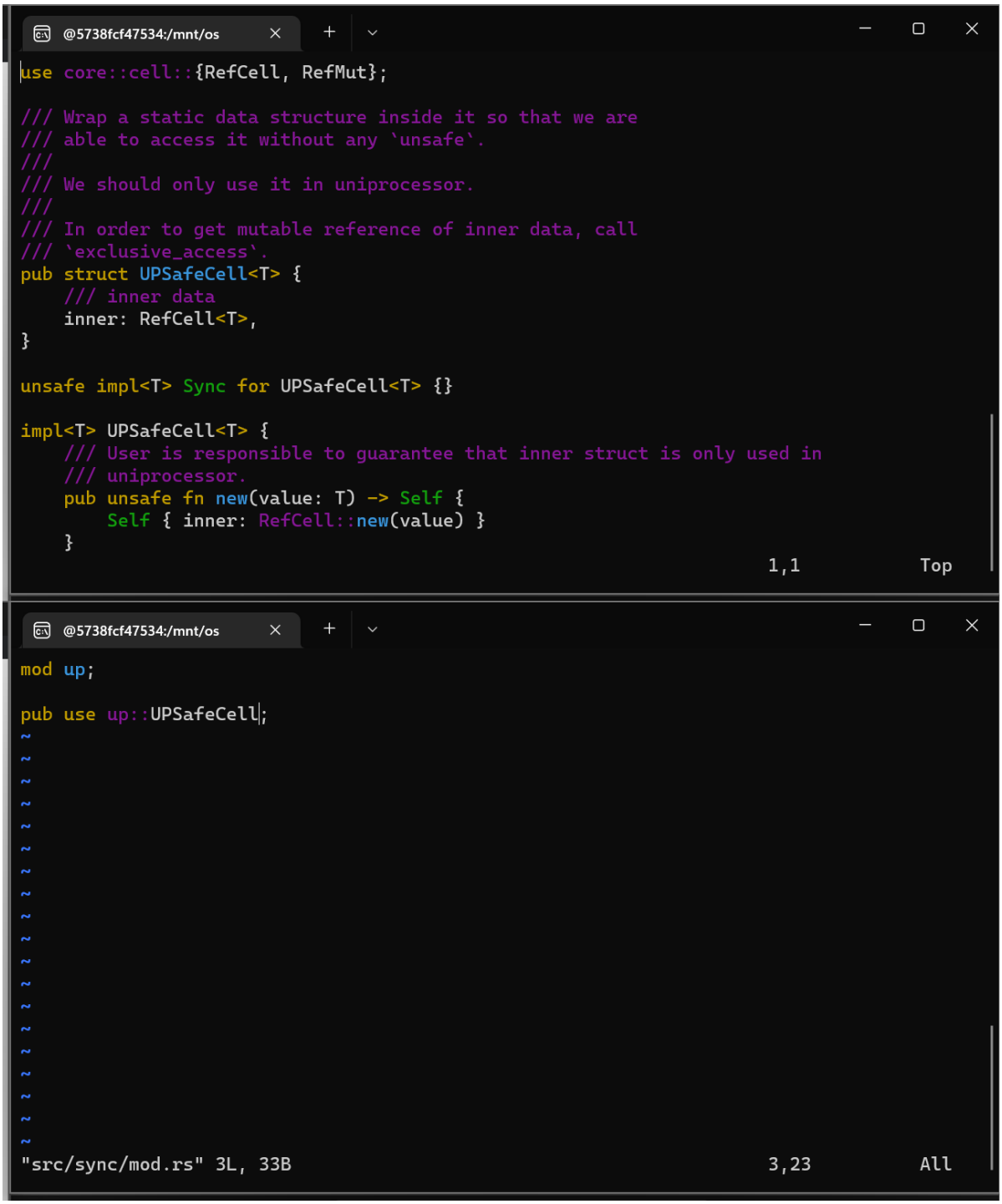
4. 实现物理帧的管理与分配

- 设置物理内存终止地址



- 实现物理帧管理

- 增加`sync`模块，实现`UPSafeCell`


@5738fcf47534:/mnt/os
use core::cell::{RefCell, RefMut};
/// Wrap a static data structure inside it so that we are
/// able to access it without any `unsafe`.
///
/// We should only use it in uniprocessor.
///
/// In order to get mutable reference of inner data, call
/// `exclusive_access`.
pub struct UPSafeCell<T> {
/// inner data
inner: RefCell<T>,
}
unsafe impl<T> Sync for UPSafeCell<T> {}
impl<T> UPSafeCell<T> {
/// User is responsible to guarantee that inner struct is only used in
/// uniprocessor.
pub unsafe fn new(value: T) -> Self {
Self { inner: RefCell::new(value) }
}
1,1 Top
@5738fcf47534:/mnt/os
mod up;
pub use up::UPSafeCell;
"src/sync/mod.rs" 3L, 33B
3,23 All

```
@5738fcf47534:/mnt/os
mod syscall;
mod trap;
mod loader;
mod config;
mod task;
mod timer;
mod mm;
mod sync;

extern crate alloc;
extern crate bitflags;

bitflags! {
    pub struct PTEFlags: u8 {
        const V = 1 << 0;
        const R = 1 << 1;
        const W = 1 << 2;
        const X = 1 << 3;
        const U = 1 << 4;
        const G = 1 << 5;
        const A = 1 << 6;
        const D = 1 << 7;
                                                    20,9          29%
```

- 物理帧管理测试

```
@5738fcf47534:/mnt/os
pub const USER_STACK_SIZE: usize = 4096 * 2;
pub const KERNEL_STACK_SIZE: usize = 4096 * 2;
pub const MAX_APP_NUM: usize = 4;
pub const APP_BASE_ADDRESS: usize = 0x80400000;
pub const APP_SIZE_LIMIT: usize = 0x20000;
pub const CLOCK_FREQ: usize = 12500000;
pub const KERNEL_HEAP_SIZE: usize = 0x30_0000;
pub const MEMORY_END: usize = 0x80800000;
pub const PAGE_SIZE: usize = 0x1000;
pub const PAGE_SIZE_BITS: usize = 0xc;
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                    10,39         All
```

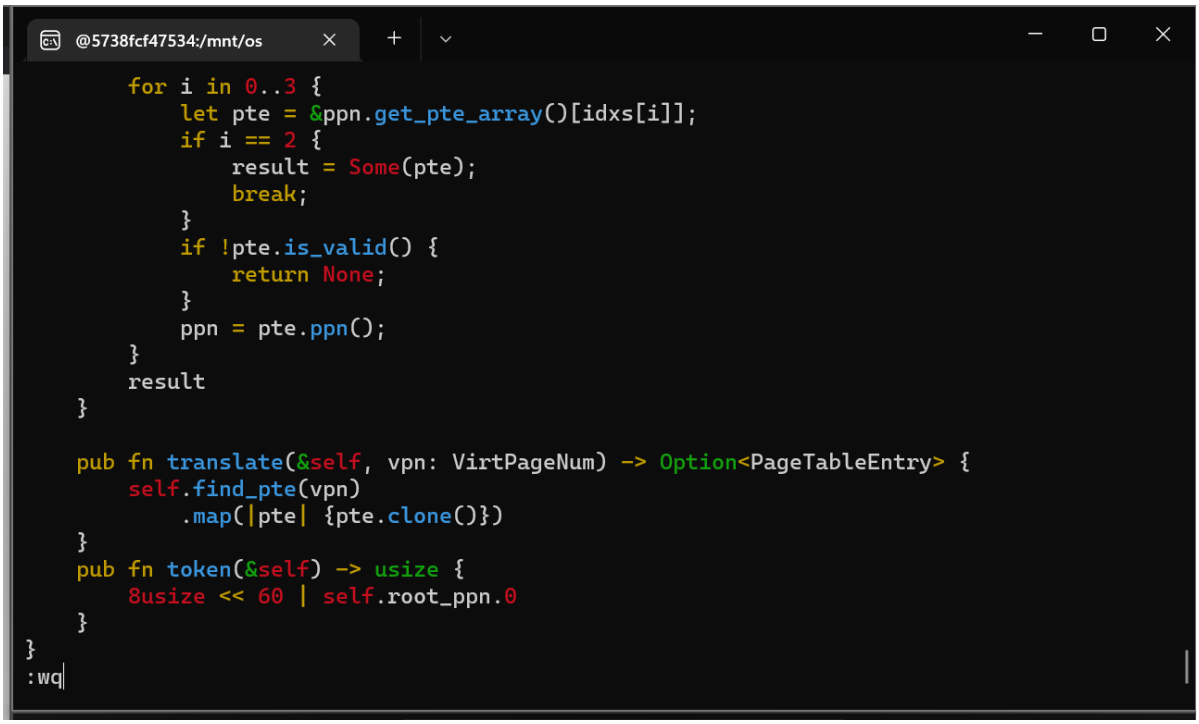
```
@5738fcf47534:/mnt/os
mod heap_allocator;
mod page_table;
mod address;
mod frame_allocator;

use address::{VPNRange, StepByOne};
pub use address::{PhysAddr, VirtAddr, PhysPageNum, VirtPageNum};
pub use frame_allocator::{FrameTracker, frame_alloc};
use page_table::{PTEFlags};
pub use page_table::{PageTableEntry};

pub fn init() {
    heap_allocator::init_heap();
    heap_allocator::heap_test();
    frame_allocator::init_frame_allocator();
    frame_allocator::frame_allocator_test();
}
~
~
~
~
~
"src/mm/mod.rs" 17L, 467B                                    10,37         All
```

- 测试成功

```
@5738fcf47534:/mnt/os
qemu-system-riscv64: clint: invalid write: 00000004
[rustsbi] enter supervisor 0x80200000
[kernel] Hello, world!
heap_test passed!
last 721 Physical Frames.
FrameTracker:PPN=0x8052f
FrameTracker:PPN=0x80530
FrameTracker:PPN=0x80531
FrameTracker:PPN=0x80532
FrameTracker:PPN=0x80533
FrameTracker:PPN=0x80533
FrameTracker:PPN=0x80532
FrameTracker:PPN=0x80531
FrameTracker:PPN=0x80530
FrameTracker:PPN=0x8052f
frame_allocator_test passed!
power_3 [10000/200000]
power_3 [20000power_5 [10000/200000]
power_5 [20000/200000]
power_5 [30000/200000]
power_5 [40000/200000]
power_5 [50000/200000]
power_5 [60000/200000]
```

5. 多级页表管理

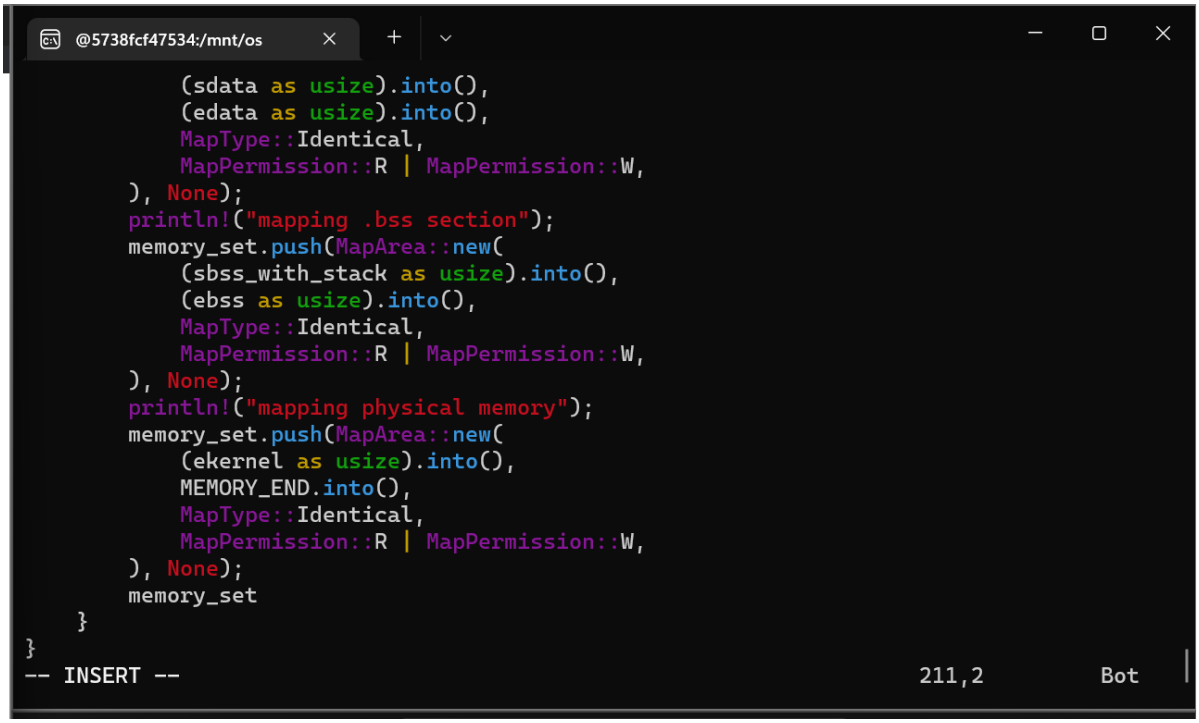
> 依然是粘贴各种代码，实现基本数据结构和操作



6. 内核与应用的地址空间

- 内核地址空间

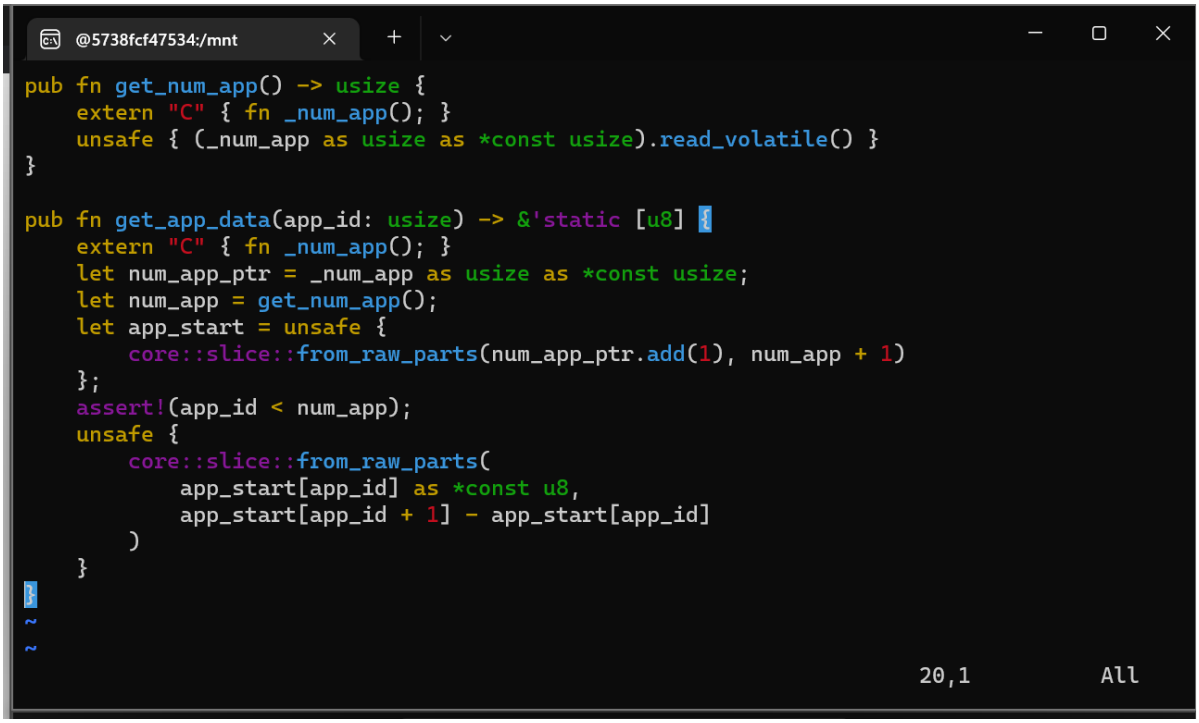
> 粘贴粘贴粘贴粘贴 主要实现了地址空间的抽象和创建内核地址空间



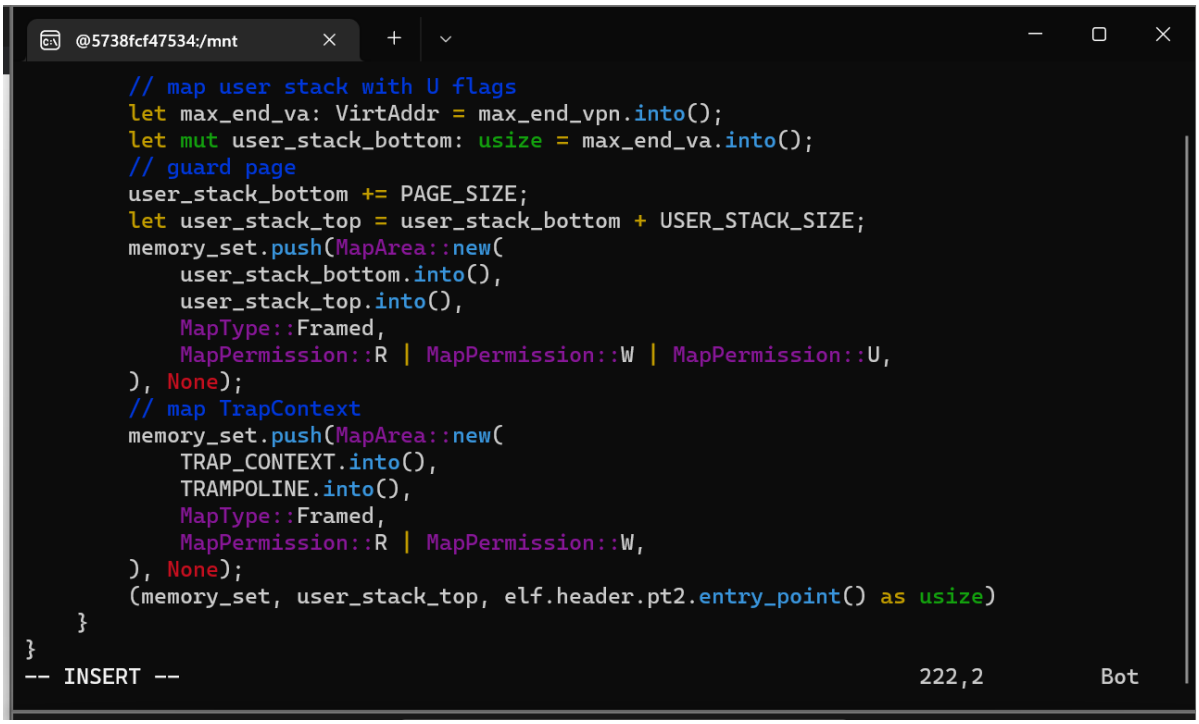
- 修改链接脚本

> 这里不知道是实验手册有问题还是我看漏了，选中的这行`sbss_with_stack`手册里好像是没有的，如果没有这一行最后build会失败

- 修改`loader`子模块

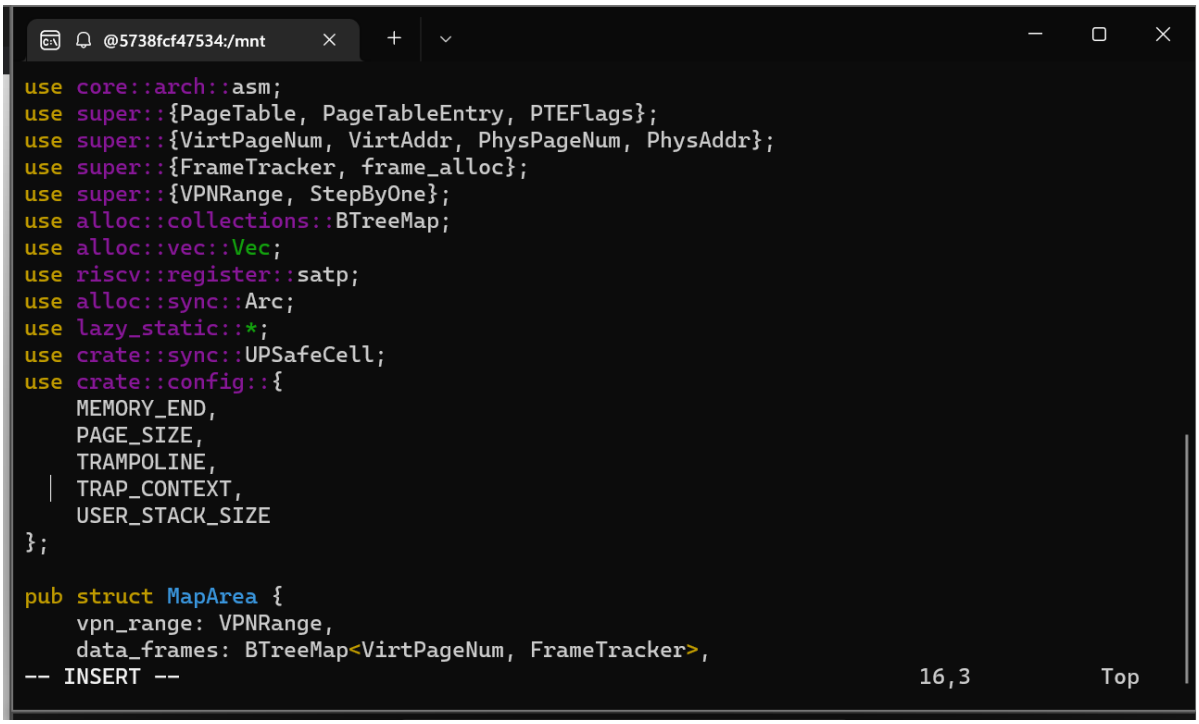


- 实现解析ELF格式的数据



- 增加`xmas-elf`依赖

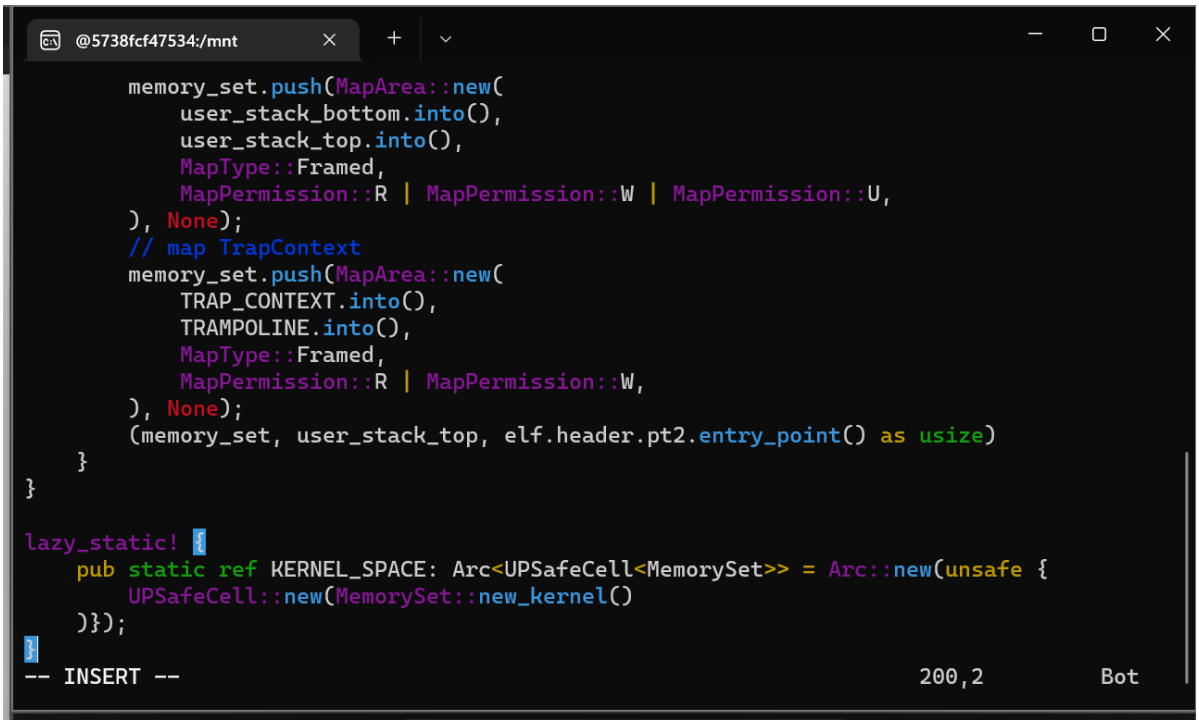
- 实现`memory_set`子模块



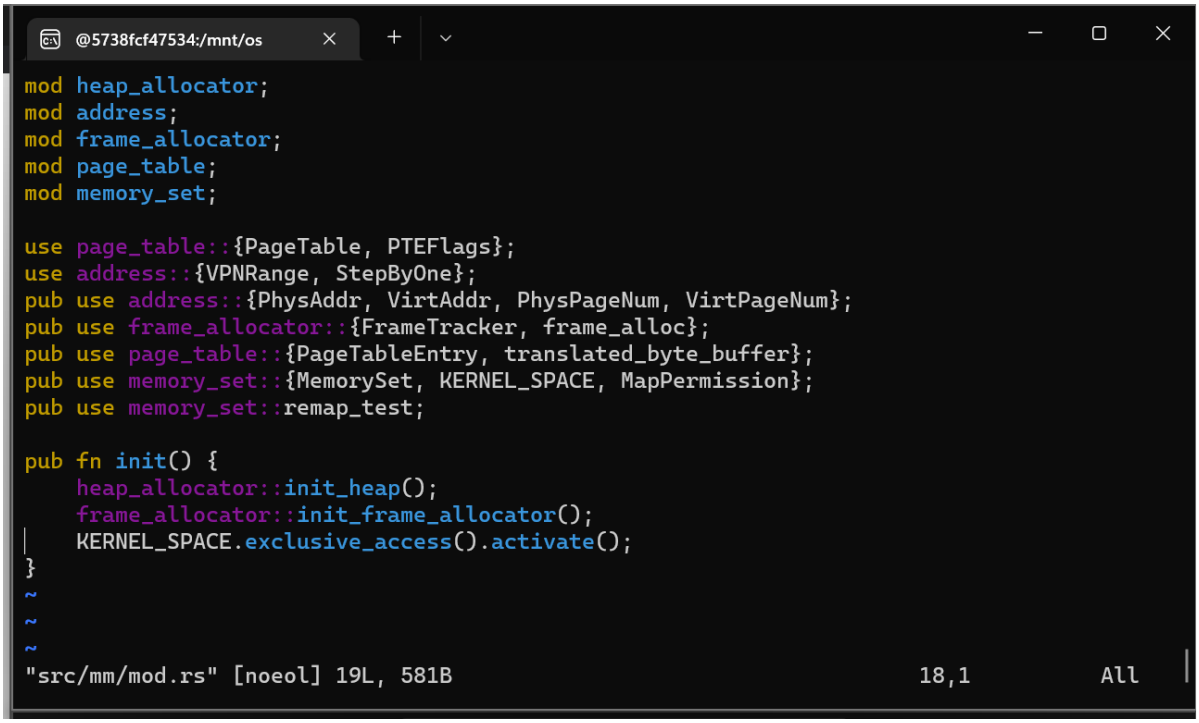
- 修改`config.rs`

7. 实现基于地址空间的分时多任务

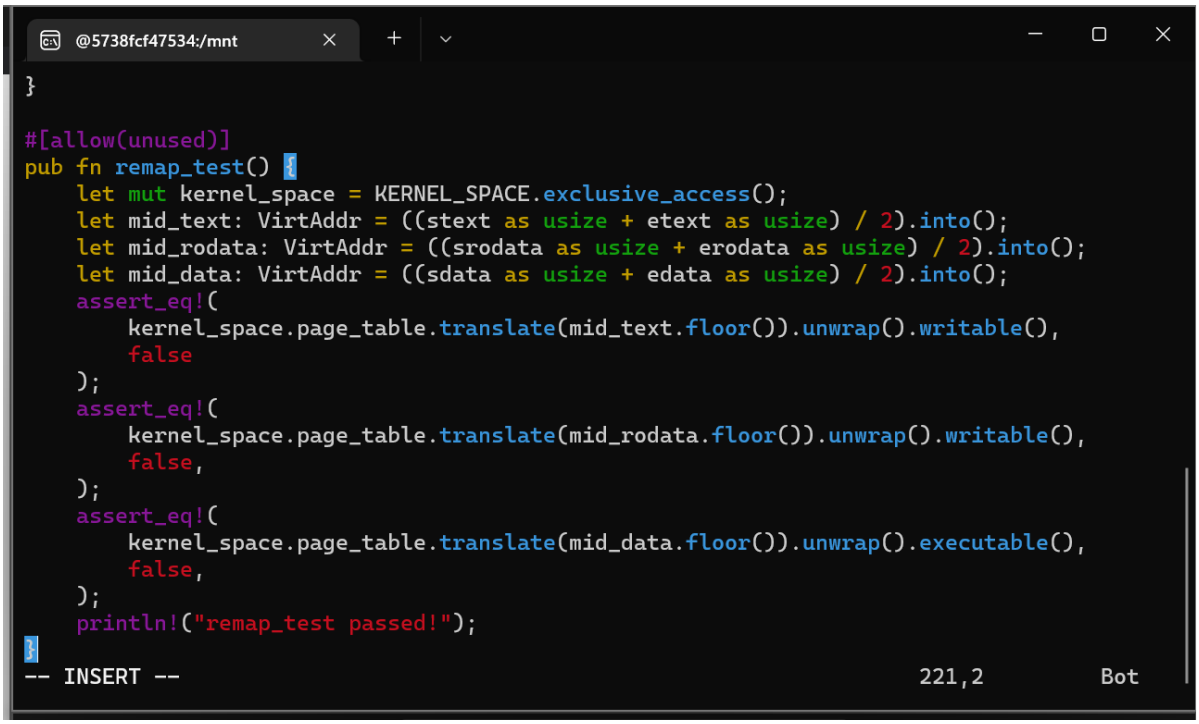
- 创建内核地址空间



- 进行内存管理子系统的初始化

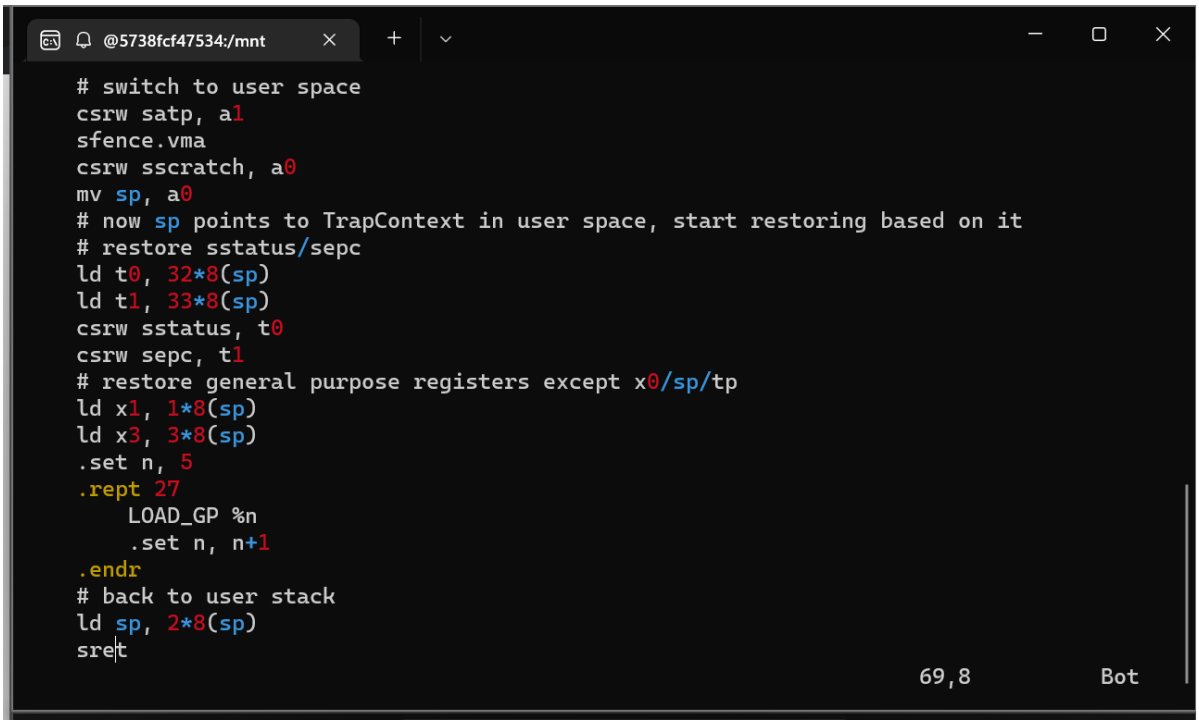


- 检查内核地址空间的多级页表设置

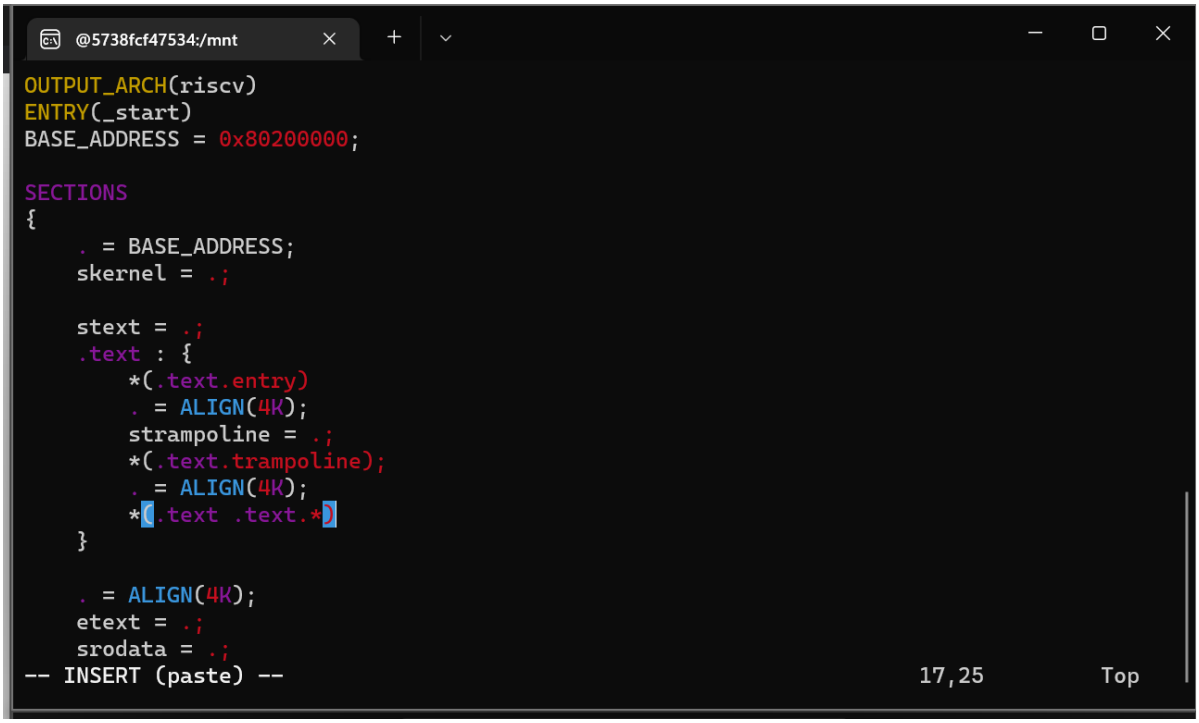


- 实现跳板机制
- 扩展`Trap`上下文

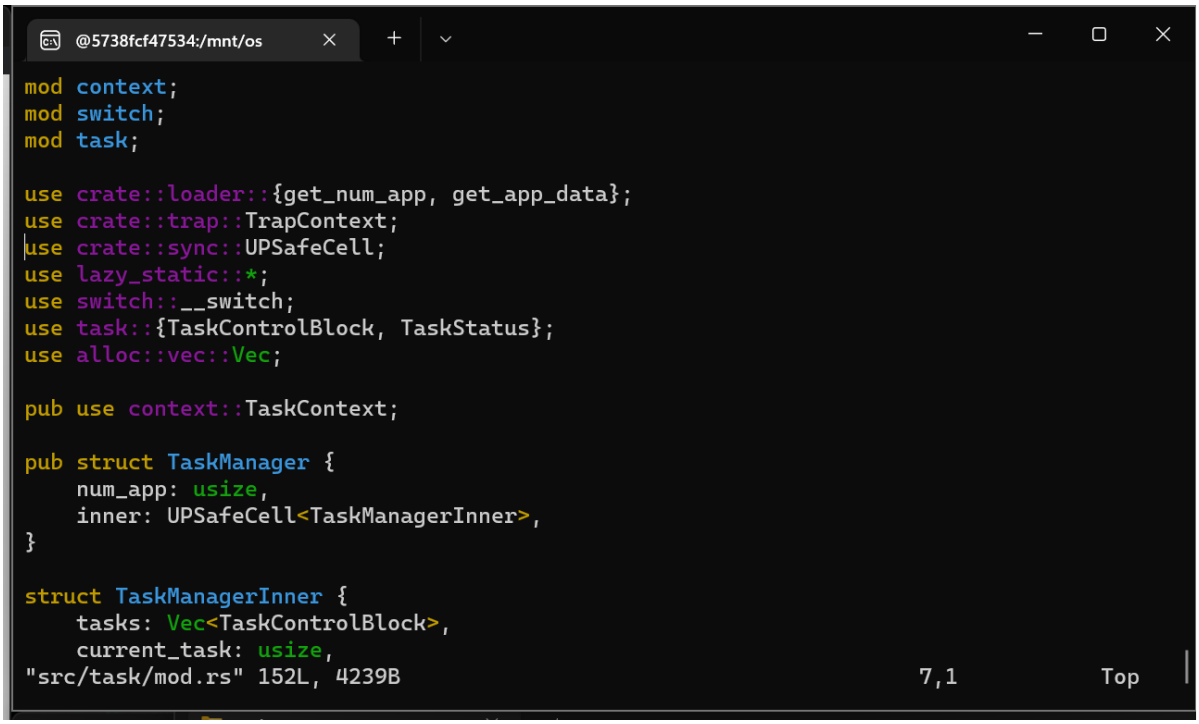
- 实现地址空间的切换



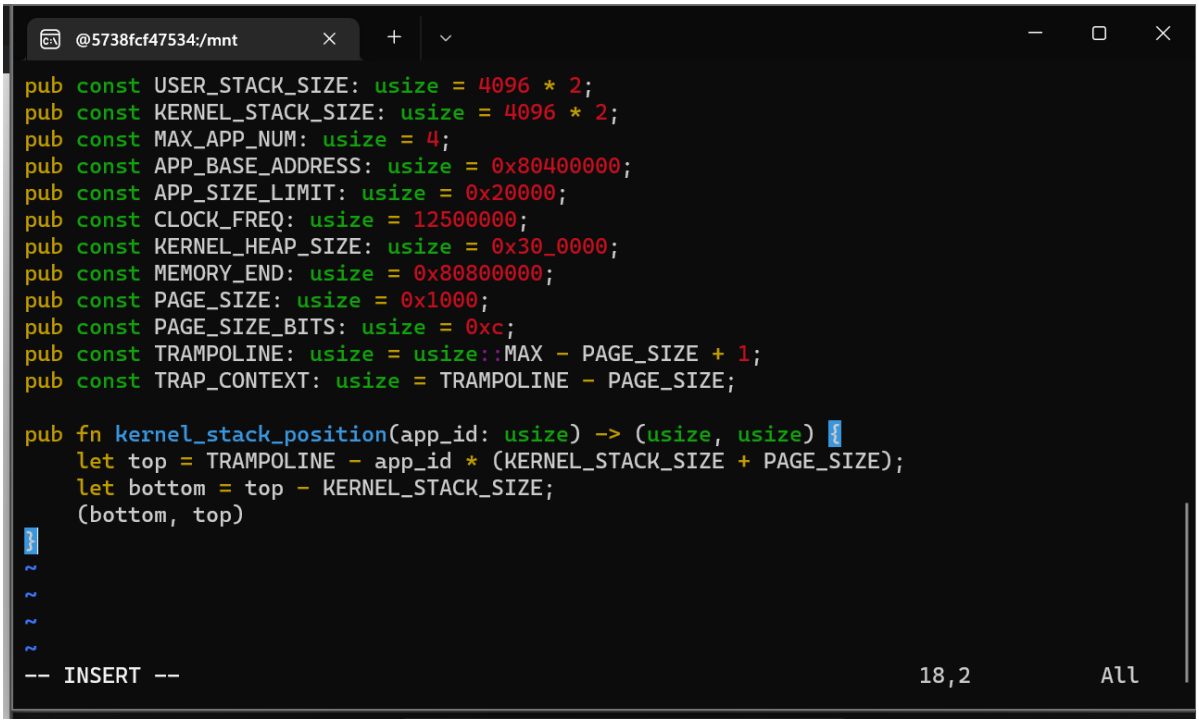
- 建立跳板页面



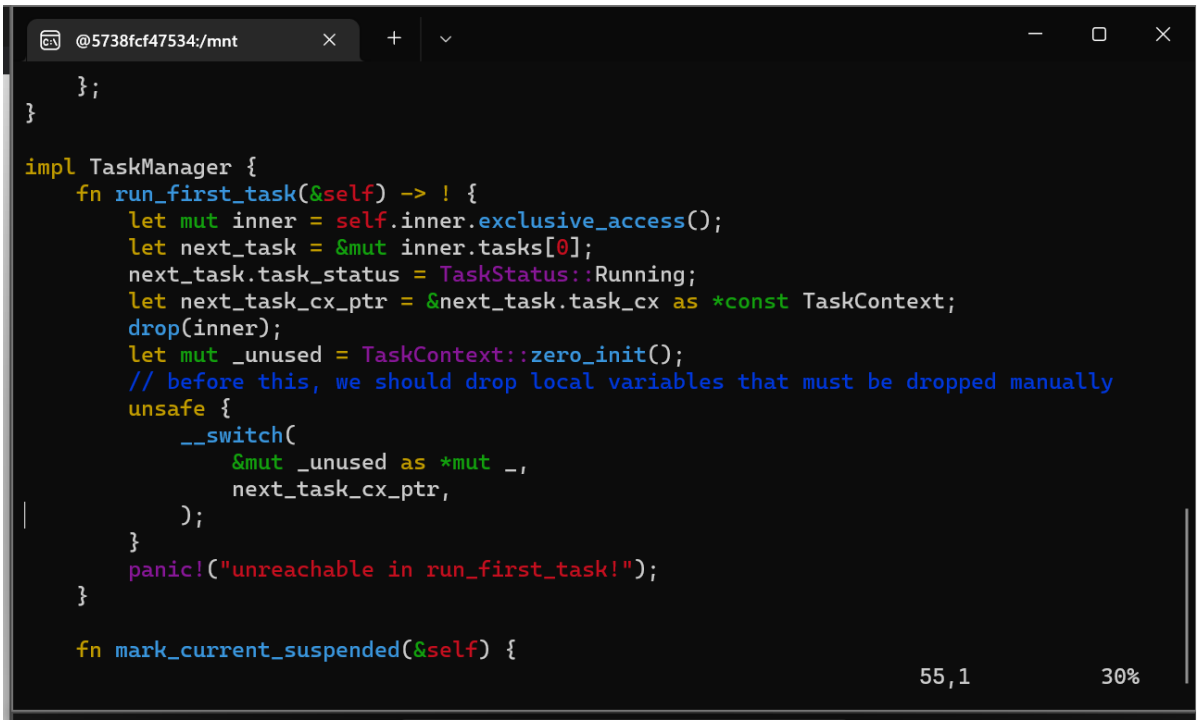
- 加载和执行应用程序
- 修改任务子模块，并更新任务控制块的管理



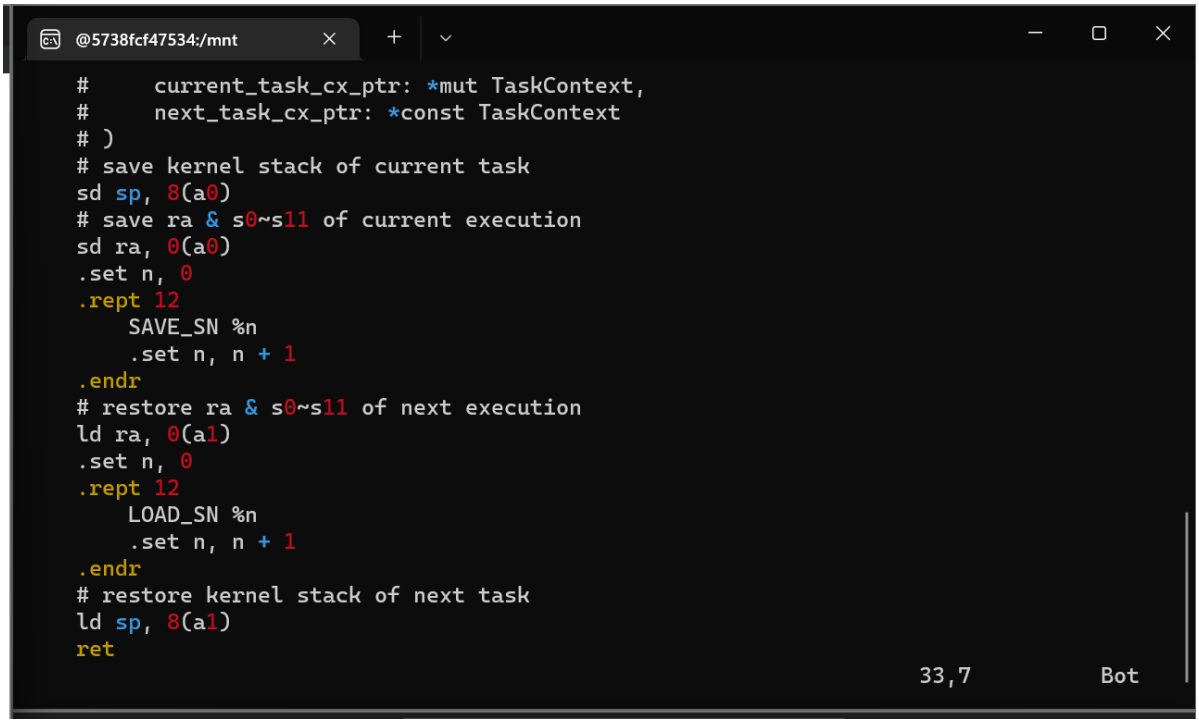
- 更新`config.rs`，在内核初始化时加载所有应用程序



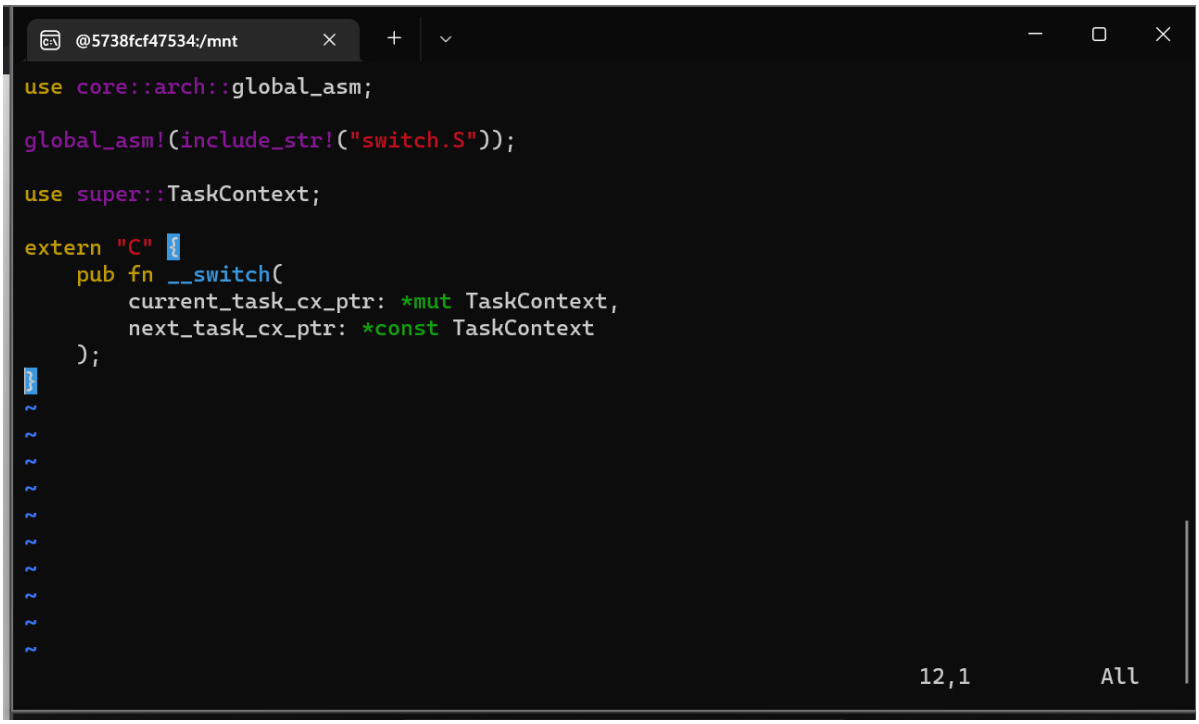
- 修改`TaskManager`的实现



- 修改`/os/src/task/switch.S`



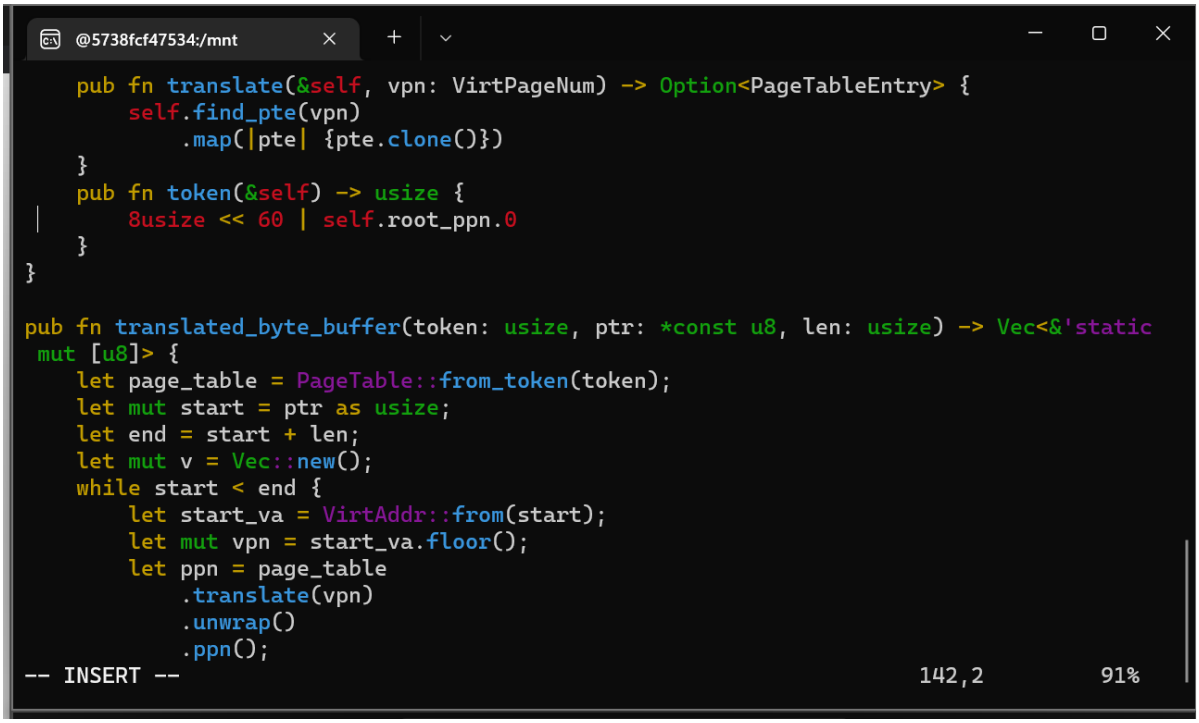
- 修改`switch.rs`



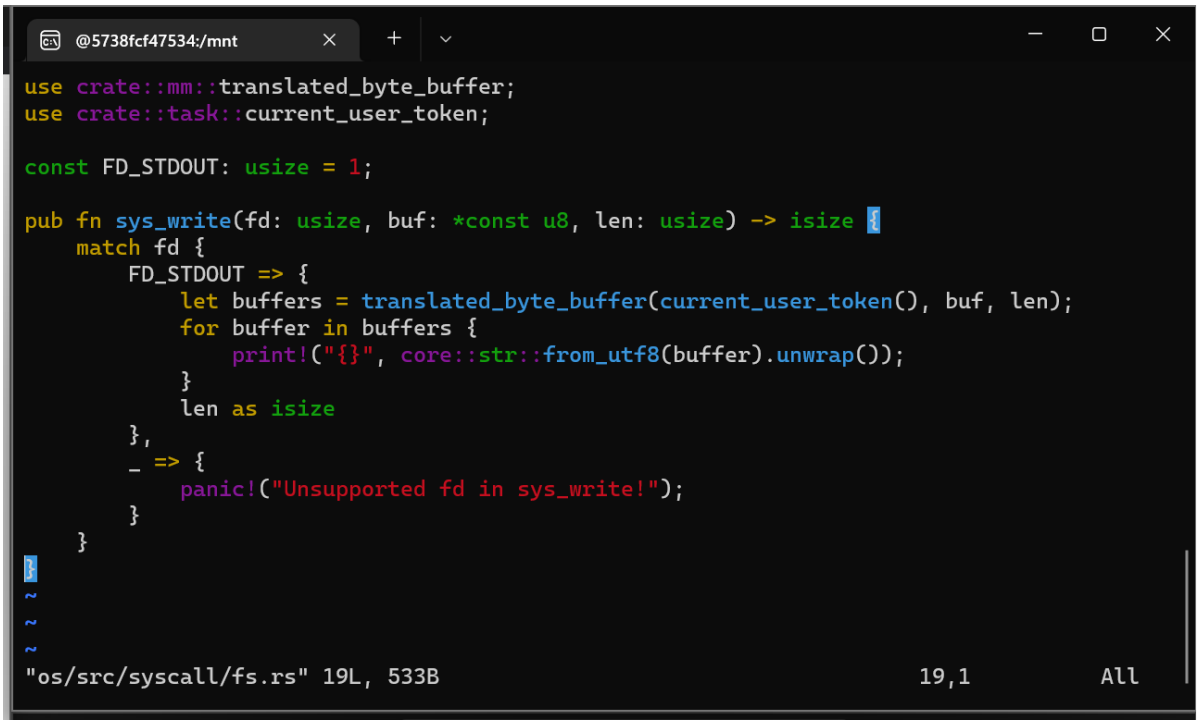
- 改进`Trap`的处理

- 在每一个应用程序第一次获得CPU权限时，内核栈顶放置在内核加载应用的时候构造的一个任务上下文

- 改进`sys_write`的实现



- 修改`sys_write`系统调用

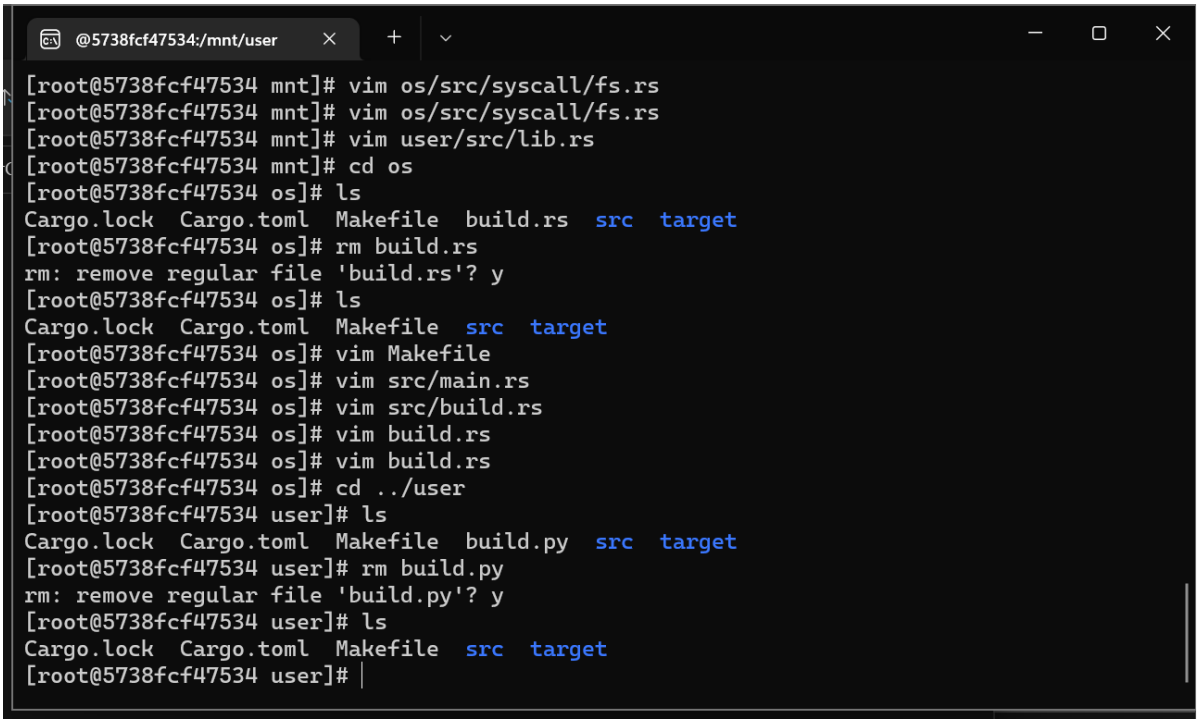


8. 修改应用程序

- 删除`user/src/lib.rs`中的`clear_bss()`

  除了删除`clear_bss()`的实现外，注意删除`_start()`中调用的`clear_bss()`。

- 删除`build.py`



- 修改`Makefile`文件

把`user/Makefile`文件中的`build.py`替换为`cargo build`

@5738fcf47534:/mnt/user

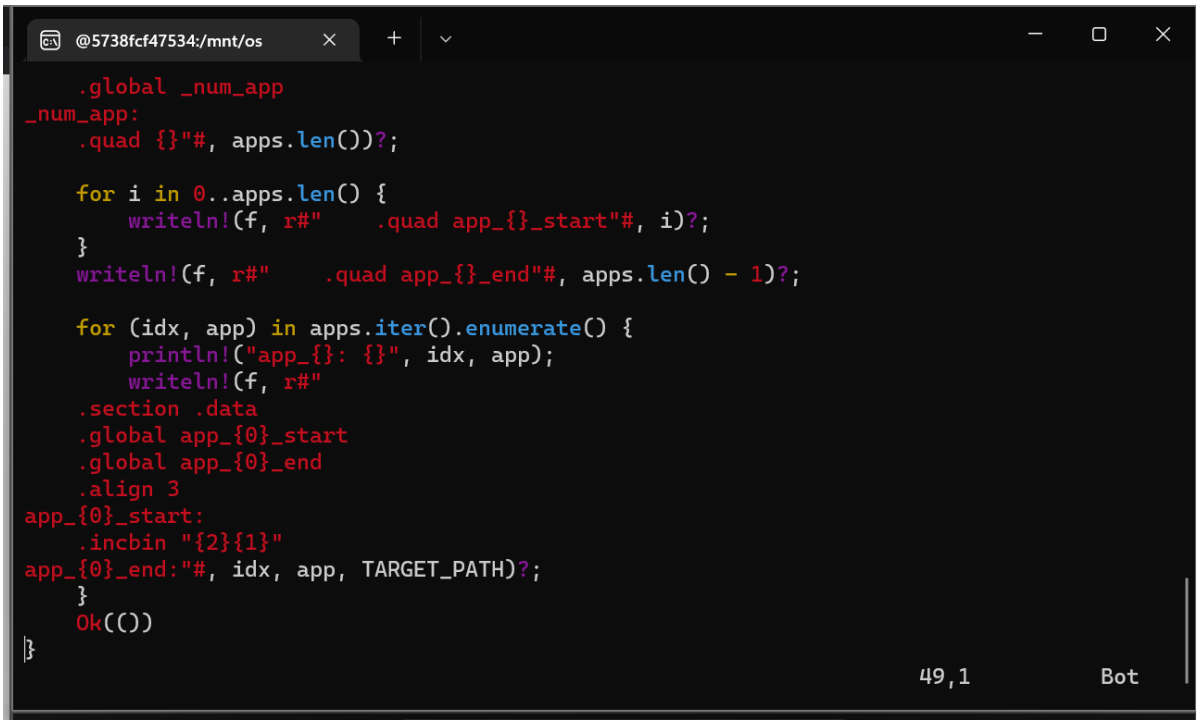
```
TARGET := riscv64gc-unknown-none-elf
MODE := release
APP_DIR := src/bin
TARGET_DIR := target/$(TARGET)/$(MODE)
APPS := $(wildcard $(APP_DIR)/*.rs)
ELFS := $(patsubst $(APP_DIR)/%.rs, $(TARGET_DIR)/%, $(APPS))
BINS := $(patsubst $(APP_DIR)/%.rs, $(TARGET_DIR)/%.bin, $(APPS))

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

elf: $(APPS)
        @cargo build --release

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

build: binary

clean:
        @cargo clean
"Makefile" 23L, 618B                                    1,1           Top
```

- 修改`main.rs`

@5738fcf47534:/mnt/os

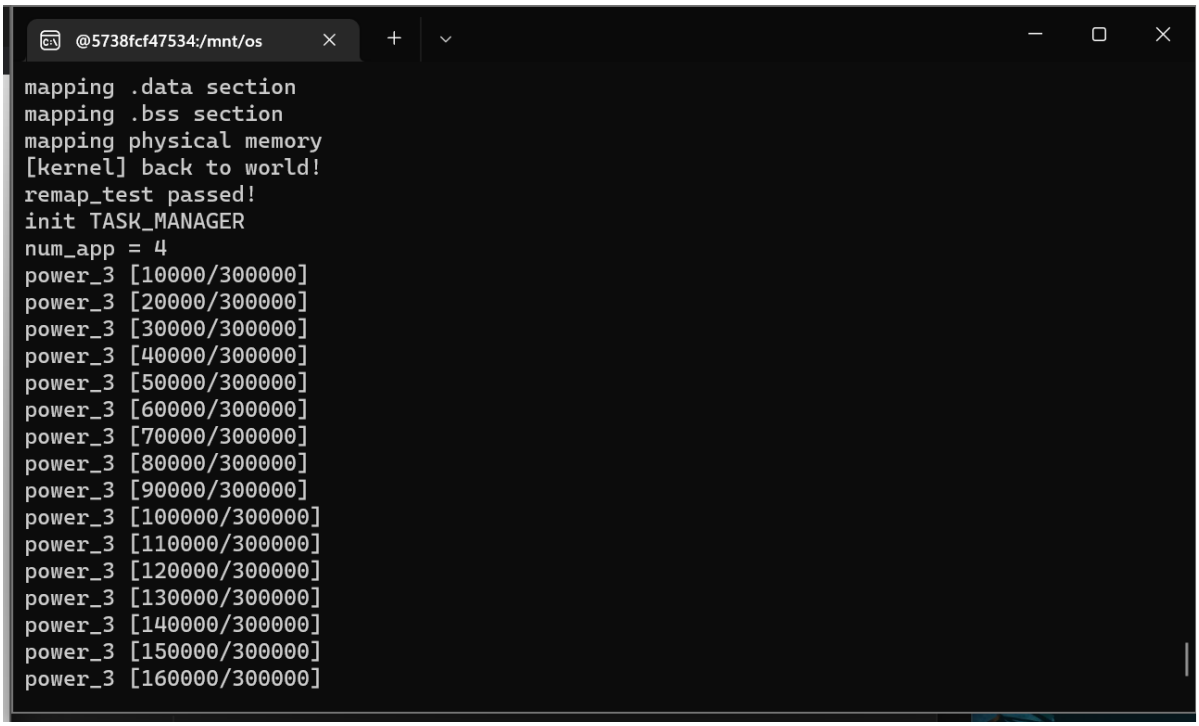
```
    }
    unsafe {
        core::slice::from_raw_parts_mut(
            sbss as usize as *mut u8,
            ebss as usize - sbss as usize,
        ).fill(0);
    }
}

#[no_mangle]
pub fn rust_main() -> ! {
    clear_bss();
    println!("[kernel] Hello, world!");
    mm::init();
    println!("[kernel] back to world!");
    mm::remap_test();
    trap::init();
    trap::enable_timer_interrupt();
    timer::set_next_trigger();
    task::run_first_task();
    panic!("Unreachable in rust_main!");
}
                                                        54,1          Bot
```

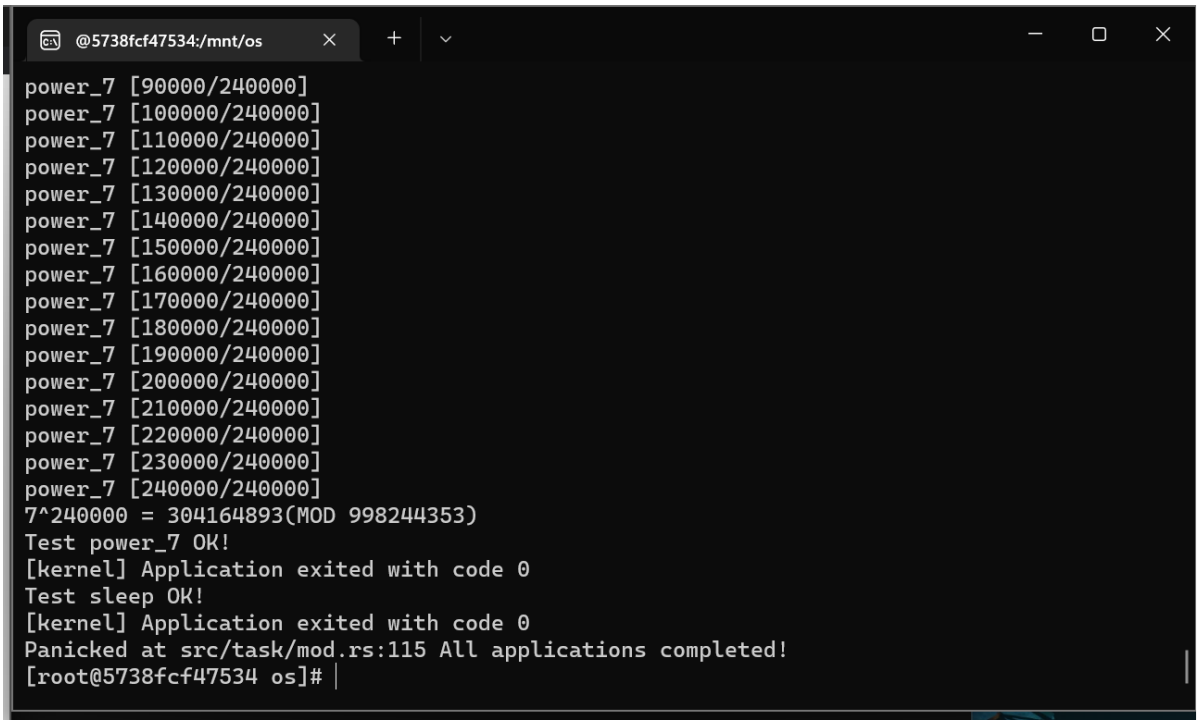
- 修改`build.rs`



8. 运行成功

应该算成功了吧，虽然有很多警告，好像是因为删代码没删干净





二. 思考题

1. 虚拟地址和物理地址的设计与实现

- **设计**: 虚拟地址（`VirtAddr`）和物理地址（`PhysAddr`）是为了方便内存管理和实现虚拟内存系统而设计的。在操作系统中，虚拟地址提供了一种抽象，允许每个程序感觉像是在使用一个大的、连续的内存空间，而实际上这些虚拟地址被映射到物理内存的不同部分。
- **实现**: 在您的代码中，这两种地址都被封装为含有一个 `usize` 字段的结构体。提供了从 `usize` 到这些类型的转换，以及这些类型之间的相互转换（例如，通过页号和页表来转换）。这种实现方式简化了地址的处理，并允许在物理和虚拟地址空间之间轻松转换。

2. 物理帧的管理与分配

- **管理**: 物理帧（内存的物理页）通过 `StackFrameAllocator` 来管理。这个分配器负责跟踪哪些物理页被使用，哪些是空闲的。
- **分配**: 通过 `frame_alloc` 函数分配物理帧。如果有可用的帧，它会返回一个包含物理页号的 `FrameTracker` 实例。当 `FrameTracker` 被丢弃时，对应的物理帧会被释放回分配器。

3. 内核和应用程序的地址空间实现

- **内核地址空间**: 通过 `MemorySet` 结构表示。在启动时，内核地址空间包括代码段、数据段、BSS段等。此地址空间始终激活并被所有进程共享。
- **应用程序地址空间**: 每个应用程序都有自己的 `MemorySet` 实例，表示其私有的地址空间。这些空间包括应用程序的代码、数据、堆栈等。在上下文切换时，会切换活跃的地址空间，确保每个应用程序只能访问自己的内存。

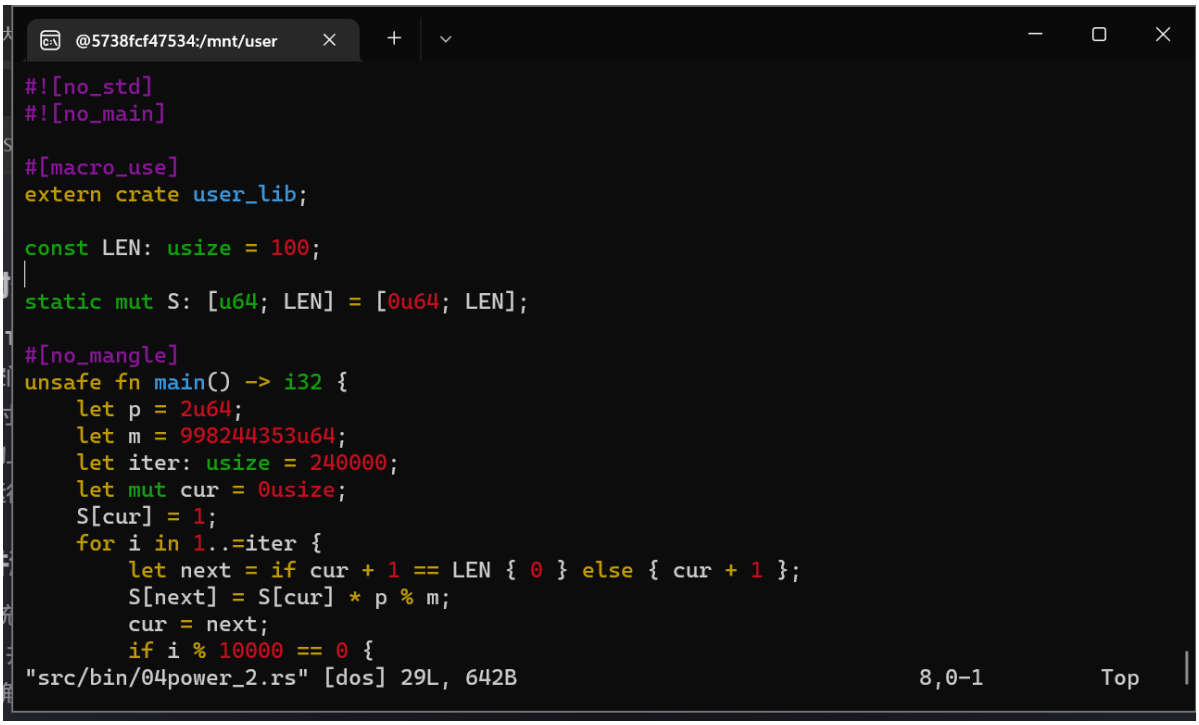
4. 基于地址空间的分时多任务实现

- **多任务**: 利用任务控制块（`TaskControlBlock`），系统维护了多个任务的状态。每个任务都有自己的虚拟地址空间（`MemorySet`），堆栈，以及其他必要的上下文信息。

- **上下文切换**: 当一个任务的时间片用完时，系统会保存其上下文（包括CPU寄存器等），并加载下一个任务的上下文。这个过程通过保存和恢复任务上下文来完成，确保每个任务在适当的时候运行。

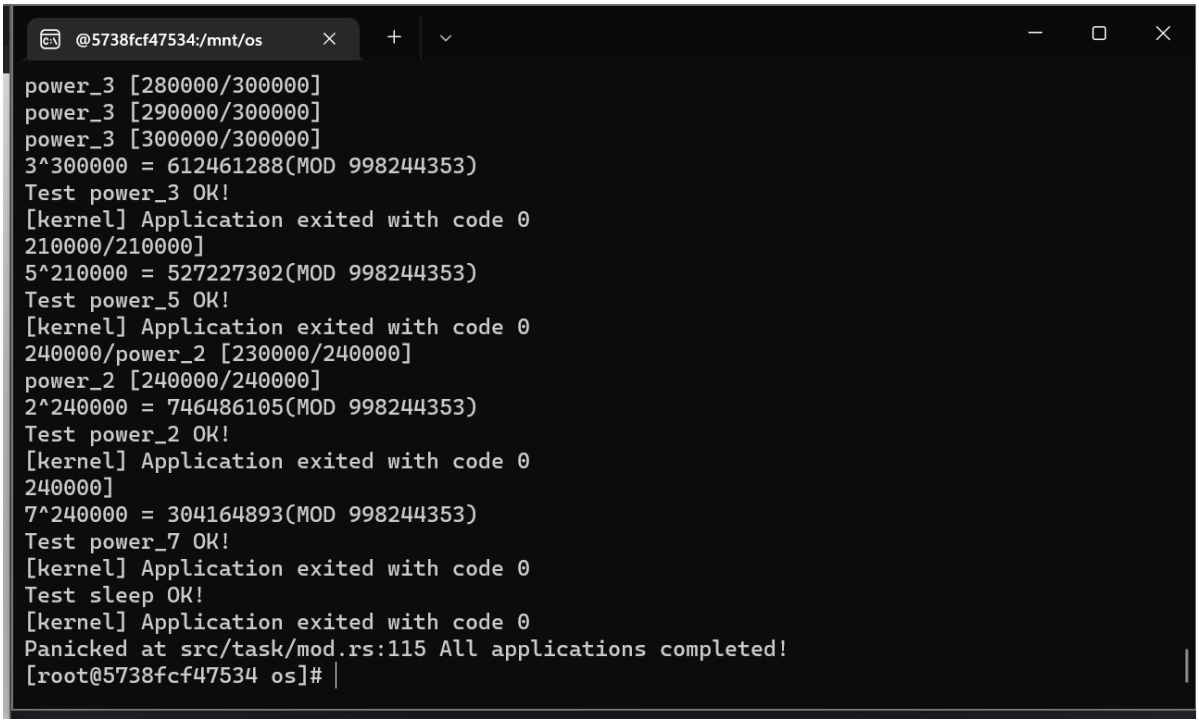
5. 编写新的应用程序并测试验证结果

- 实现了一个新的计算2次幂的程序



- 运行结果：

看起来没什么问题



三. Git提交截图