|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| 北京交通大学软件学院  **《操作系统》课程**  **实验报告** | | |

|  |
| --- |
| **姓名:韩熔** |
| **学号:20301036** |

目录

[1 实验目的 3](#_Toc118133029)

[2 实验过程设计 3](#_Toc118133030)

[3 源代码及注释 3](#_Toc118133031)

[4 运行结果与分析 4](#_Toc118133032)

[5 实验总结 4](#_Toc118133033)

# 1 实验目的

希望能在操作系统中提供动态申请和释放内存的能力，加强操作系统对各种以内存为基础的资源分配与管理。

# 2 实验过程设计

（1）了解静态与动态内存分配

（2）学习Rust 中的堆数据结构

（3）学习如何在内核中支持动态内存分配

（4）构建并运行代码，查看动态分配内存的功能是否已经实现

# 3 源代码及注释

表 3-1 源代码及注释

|  |  |
| --- | --- |
| **文件名:** | os/Cargo.toml |
| **主要功能:** | 添加依赖 |
| **源代码:**  **//添加 crate 依赖**  **buddy\_system\_allocator = "0.6"** | |

|  |  |
| --- | --- |
| **文件名:** | os/src/main.rs |
| **主要功能:** | 主模块和入口点 |
| **源代码:**  **//引入 alloc 库的依赖**  **extern crate alloc;**  **//定义alloc\_error\_handler这个函数**  **#![feature(alloc\_error\_handler)]**  **#[no\_mangle]**  **//在 rust\_main()调用mm::init() 函数**  **pub fn rust\_main() -> ! {**  **clear\_bss();**  **println!("[kernel] Hello, world!");**  **mm::init();**  **panic!("Shutdown machine!");**  **}** | |

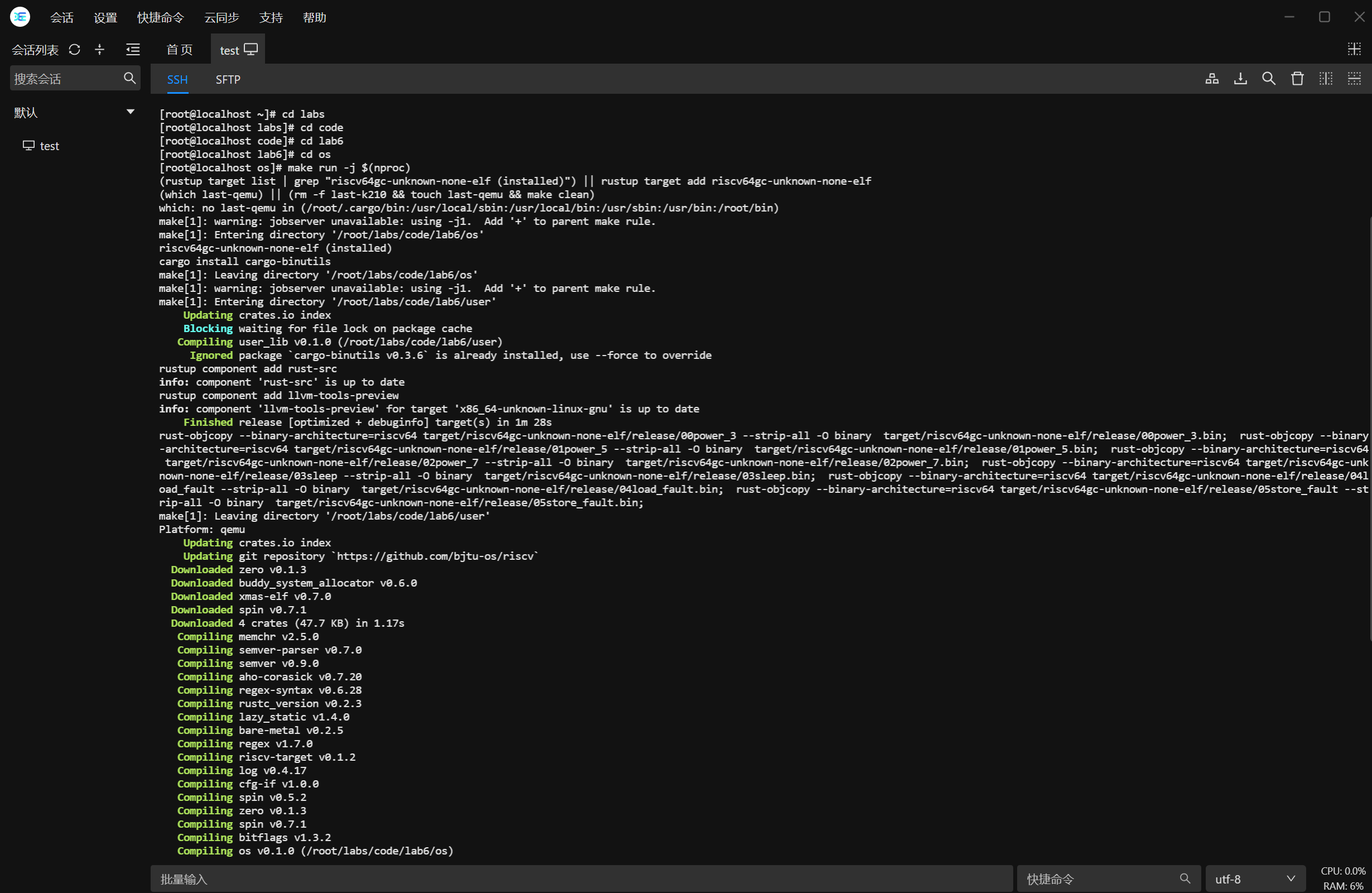
|  |  |
| --- | --- |
| **文件名:** | os/src/mm/heap\_allocator.rs |
| **主要功能:** | 实现[`FrameAllocator`]  控制操作系统中的所有frame |
| **源代码:**  **//根据 alloc 留好的接口提供全局动态内存分配器**  **// os/src/mm/heap\_allocator.rs**  **use buddy\_system\_allocator::LockedHeap;**  **use crate::config::KERNEL\_HEAP\_SIZE;**  **#[global\_allocator]**  **//将 buddy\_system\_allocator 中提供的 LockedHeap 实例化成一个全局变量，并使用 alloc 要求的 #[global\_allocator] 语义项进行标记**  **static HEAP\_ALLOCATOR: LockedHeap = LockedHeap::empty();**  **//这块内存是一个 static mut 且被零初始化的字节数组，位于内核的 .bss 段中**  **static mut HEAP\_SPACE: [u8; KERNEL\_HEAP\_SIZE] = [0; KERNEL\_HEAP\_SIZE];**  **//调用 init\_heap 函数来给我们的全局分配器一块内存用于分配**  **pub fn init\_heap() {**  **unsafe {**  **HEAP\_ALLOCATOR**  **.lock()**  **.init(HEAP\_SPACE.as\_ptr() as usize, KERNEL\_HEAP\_SIZE);**  **}**  **}**  **//处理动态内存分配失败的情形，直接 panic**  **#[alloc\_error\_handler]**  **pub fn handle\_alloc\_error(layout: core::alloc::Layout) -> ! {**  **panic!("Heap allocation error, layout = {:?}", layout);**  **}**  **#[allow(unused)]**  **//heap\_test() 函数**  **pub fn heap\_test() {**  **//使用智能指针 Box<T> 和向量 Vec<T> 在堆上分配数据并管理它们**  **use alloc::boxed::Box;**  **use alloc::vec::Vec;**  **extern "C" {**  **fn sbss();**  **fn ebss();**  **}**  **let bss\_range = sbss as usize..ebss as usize;**  **let a = Box::new(5);**  **assert\_eq!(\*a, 5);**  **//通过 as\_ref方法可以看到指向的数据的位置**  **assert!(bss\_range.contains(&(a.as\_ref() as \*const \_ as usize)));**  **drop(a);**  **let mut v: Vec<usize> = Vec::new();**  **for i in 0..500 {**  **v.push(i);**  **}**  **for i in 0..500 {**  **assert\_eq!(v[i], i);**  **}**  **//通过 as\_ptr方法可以看到指向的数据的位置**  **assert!(bss\_range.contains(&(v.as\_ptr() as usize)));**  **drop(v);**  **println!("heap\_test passed!");**  **}** | |

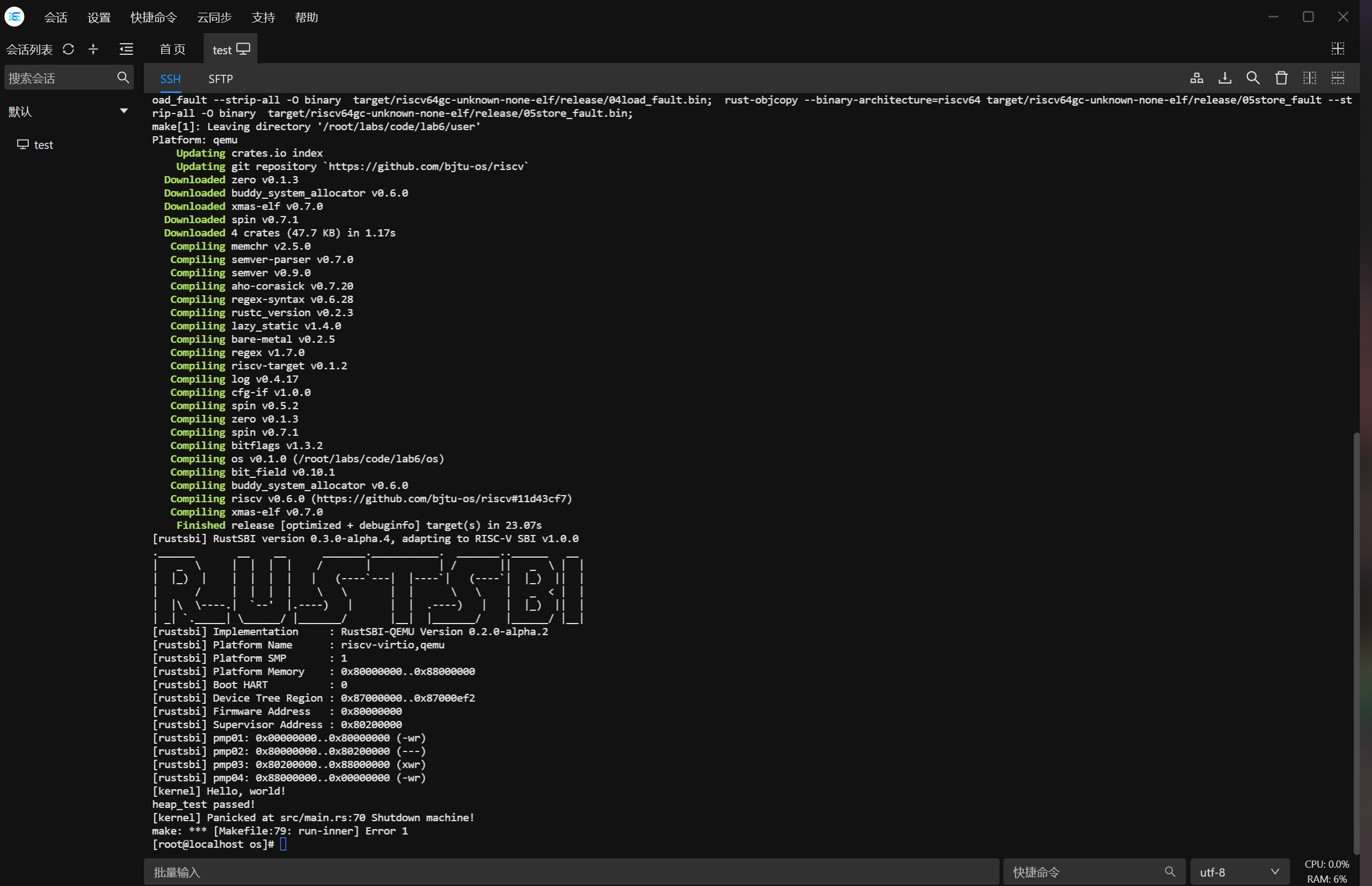
|  |  |
| --- | --- |
| **文件名:** | os/src/mm/mod.rs |
| **主要功能:** | 实现内存管理 |
| **源代码:**  **pub fn init() {**  **//初始化动态分配内存的功能**  **heap\_allocator::init\_heap();**  **//测试动态分配内存的功能**  **heap\_allocator::heap\_test();**  **}** | |

表 3-2 关键命令解释

|  |  |  |
| --- | --- | --- |
| **序号** | **命令名** | **主要作用** |
| **1** | **$ make run -j $(nproc)** | 构建并运行代码 |
| **2** |  |  |
| **3** |  |  |
| **…** |  |  |

# 4 运行结果与分析





heap\_test() 已经通过，说明我们的动态分配内存的功能已经实现

# 5 实验总结

本节我们尝试在内核中支持动态内存分配，以能使用 Rust 核心库中各种灵活的动态数据结构，如 Vec、HashMap 等，且不用考虑这些数据结构的动态内存释放的繁琐操作，充分利用 Rust 语言保证的内存安全能力。