華中師範大學
CENTRAL CHINA NORMAL UNIVERSITY

操作系统实验报告

第二次实验

姓名 赖景康

学号 2023214564

课程 操作系统

学院 计算机学院

2025 年 11 月 20 日

1 实验名称

进程 PCB 的组织、管理与调度模拟——实验二

2 实验目的

- 理解并实现 PCB（进程控制块）的数据结构与常见操作（创建、撤销、挂起、激活）。
- 实现并比较若干调度策略（FCFS、SJF、RR、优先级），掌握调度选择与时间片处理。
- 了解并实现一种简单的内存管理（伙伴系统），观察进程创建/结束对内存的分配与回收。
- 熟悉 Rust 语言实现操作系统模拟模块的基本方法与调试技巧。

3 实验环境

- 操作系统：macOS
- 开发工具：VS Code
- 编程语言：Rust (stable)
- 构建与运行：Cargo

4 问题描述

实现一个用于教学验证的进程管理模拟器，要求：

- 维护 PCB 池与若干队列（就绪/运行/等待/挂起），支持进程的增删查改操作；
- 提供多种调度算法（FCFS、SJF、RR、Priority），支持基于时间片的抢占与非抢占调度策略；
- 实现简化的伙伴系统内存分配与回收；
- 提供交互式菜单与自动模拟运行，用于观察系统快照与调度过程。

5 设计思想与模块划分

本程序用模块化设计组织，主要模块与数据结构如下：

- **PCB (struct PCB)**：保存进程 pid、名称、状态、优先级、到达时间、运行时间、剩余时间、等待/周转时间、内存需求与分配地址等字段；提供状态更新、时间片执行、等待时间累加等方法。
- **MemoryBlock / BuddySystem**：实现伙伴式内存管理（简化版），以块为单位分配/释放，支持块分割与相邻空闲块合并，便于模拟进程对内存的占用与回收。
- **ProcessQueueManager**：管理 PCB 池与各种队列（就绪/等待/运行/挂起），提供创建、撤销、挂起、激活、时间片过期等操作。
- **ProcessScheduler**：封装调度算法选择与时间片执行逻辑，提供 select_next_process、schedule、execute_time_slice 等方法，支持 FCFS、SJF、RR 和基于优先级的选择。
- **InteractiveMenu**：交互式菜单与自动模拟逻辑，初始化示例进程，负责读取用户输入并调用管理器/调度器完成任务。

6 实现要点（摘录自源码）

下面列出程序中的关键点与简要说明：

6.1 PCB 与进程操作

PCB 提供构造、新增、状态变更与按时间片执行的能力，核心函数签名（简化）如下：

```
struct PCB { pid: u32, name: String, state: ProcessState, priority: u8, ... }
impl PCB {
  fn new(pid: u32, name: String, priority: u8, arrival_time: u32, burst_time: u32, mem
  fn update_state(&mut self, new_state: ProcessState)
  fn execute_time_slice(&mut self, time_slice: u32) -> bool // 返回是否完成
}
```

6.2 伙伴系统（简化）

伙伴系统维护若干内存块（MemoryBlock），按需要分割或合并以分配/释放。该实现使用线性向量存放块信息，并通过遍历查找首次适配块：

```
struct BuddySystem { blocks: Vec<MemoryBlock>, total_memory: u32, min_block_size: u32
impl BuddySystem {
  fn allocate(&mut self, size: u32, pid: u32) -> Option<u32>
  fn deallocate(&mut self, start_addr: u32, pid: u32) -> bool
}
```

该实现为教学简化版本，分割策略与真正的二进制幂次伙伴系统有所不同，但能反映分配/回收与合并的基本思想。

6.3 调度器与调度逻辑

ProcessScheduler 支持四种算法：

- FCFS: 选择就绪队列首个进程；
- SJF: 选择剩余时间最短的就绪进程；
- RR: 按就绪队列顺序，使用固定时间片；
- Priority: 选择优先级最高的进程。

调度步骤（高层）：

1. select_next_process 从就绪队列选取候选；
2. 若选中则尝试为其分配内存；
3. 分配成功后将其状态置为 Running 并放入运行队列；
4. 调用 execute_time_slice 执行时间片：若进程完成则释放内存并终止，否则将其回到就绪队列等待下一次调度。

7 使用方法与运行说明

在项目根目录下（包含 'Cargo.toml' 的目录），使用下面命令构建并运行：

```
# 构建并运行交互式模拟
cargo run --manifest-path ../task2/Cargo.toml --bin task2
# 或在 task2 目录下直接运行
cd task2
cargo run
```

运行后会出现交互菜单：创建进程、撤销、挂起/激活、执行时间片、显示系统快照、切换调度算法、查看内存状态或运行自动模拟等。

8 示例输出（片段）

下面给出一次典型运行的片段（仅示意，运行环境与随机输入会影响结果）：

说明 下面预留位置用于插入本地运行得到的真实输出（建议在本地运行程序并将输出粘贴到此处以完成报告）：

```
<在此粘贴运行程序后的控制台输出>
例如：
=== 进程PCB组织、管理与调度模拟实验 ===
欢迎使用进程管理系统!
创建初始测试进程...
创建进程: PID: 1, Name: 进程A, State: Ready, Priority: 3, Arrival: 0, Burst: 5, Remai
...
--- 时间片 1 ---
时间片执行完成，进程 0 已完成
=== 系统快照 (时间: 5) ===
... (输出省略)
```

如何在本地获取输出 在仓库根目录或 'task2' 目录下运行（需安装 Rust + Cargo）：

```
cd task2
cargo run
```

然后在交互菜单中选择“9”（自动模拟）以运行一段时间片并打印系统快照，结束后选择“0”退出。如果希望一次性运行并自动执行选项，可在类 Unix shell 中使用：

```
printf "9\n0\n" | cargo run
```

（注意：本次编辑环境无法直接调用 Cargo 来运行程序，因此未把真实输出嵌入本报告，请在本地运行后将输出粘贴到上面的占位块中。）

请在本地运行并将真实输出粘贴到此处以便做更精确的结果统计与分析。

9 验证与注意事项

- 程序为教学模拟，未实现完整的幂次伙伴系统的位运算分裂/合并逻辑；若用于更严格的对比测试，应将伙伴算法改为基于幂次的分块实现。
- 当前内存分配策略为首次适配并作简单分割，可能在某些顺序下导致较多小块碎片。
- PCB 池大小在 'ProcessQueueManager::new(pool$_size$)' 10 PCB

10 结论与心得

本次实验实现了一个功能完整的进程管理教学模拟，包括 PCB 池管理、队列管理、若干调度算法和简化的内存管理（伙伴式）。通过交互式和自动模拟可以直观观察调度策略对周转时间与等待时间的影响。用 Rust 实现能较好地体现内存与所有权管理，以及使用枚举/结构体来清晰表达系统状态。

11 附录：关键代码片段

主要实现位于 ‘task2/src/main.rs‘，可将该文件作为附录代码。以下为节选：

```
// PCB 结构体与基本方法
pub struct PCB { pid: u32, name: String, state: ProcessState, priority: u8, arrival_t

impl PCB {
  pub fn new(...) -> Self { ... }
  pub fn execute_time_slice(&mut self, time_slice: u32) -> bool { ... }
}

// BuddySystem: allocate / deallocate / merge_blocks
// ProcessScheduler: select_next_process / schedule / execute_time_slice
```

12 参考文献

- Silberschatz, Galvin, Gagne. Operating System Concepts.
- Tanenbaum. Modern Operating Systems.
- Rust 官方文档：https://www.rust-lang.org/
- Github: https://github.com/shufufufu/OS-TASKS