華中師範大學
CENTRAL CHINA NORMAL UNIVERSITY

操作系统实验报告

第一次实验

姓名 赖景康

学号 2023214564

课程 操作系统

学院 计算机学院

2025 年 10 月 7 日

1 实验目的和要求

- 掌握周转时间、等待时间、平均周转时间等概念及其计算方法。
- 理解三种常用的进程调度算法，区分算法之间的差异性，并模拟实现各算法。
- 了解操作系统中高级调度、中级调度和低级调度的区别和联系

2 问题描述

- (1) 在单道环境下，已知 n 个作业的进入时间和估计运行时间（以分钟计），分别求出每一个作业的开始时间、结束时间、周转时间、带权周转时间，以及这些作业的平均周转时间和带权平均周转时间；
- (2) 在多道环境（如 2 道）下，已知 n 个作业的进入时间和估计运行时间（以分钟计），分别求出每一个作业的开始时间、结束时间、周转时间、带权周转时间，以及这些作业的平均周转时间和带权平均周转时间。

3 实验要求

- 分别用先来先服务调度算法（FCFS）、短作业优先调度算法（SJF）、响应比高者优先调度算法（HRRN），求出批作业的平均周转时间和带权平均周转时间；
- 就同一批次作业，分别讨论这些算法的优劣；
- 衡量同一调度算法对不同作业流的性能。

4 实验环境

- 开发工具：VS Code
- 编程语言：Rust

5 设计思想及实验步骤

（包括实验设计原理，分析方法、计算步骤、模块组织，或主要流程图、伪代码等）

5.1 实验设计原理

本实验采用非抢占式批处理作业调度模型，核心指标：周转时间 $T_i = C_i - A_i$、带权周转时间 $W_i = T_i/S_i$，以及其平均值 $\overline{T} = \frac{1}{n}\sum_i T_i$、$\overline{W} = \frac{1}{n}\sum_i W_i$。多道情形抽象为 m 条并行"道"，作业一旦开始在某道上运行至完成。

5.2 分析方法

实现三种典型非抢占调度算法：FCFS（按到达时间先后分配到最早空闲道）、SJF（就绪集合选服务时间最短者）、HRRN（就绪集合按 $R = (t - A + S)/S$ 从大到小选择）。时间推进遵循事件驱动：若有空闲道且就绪非空则立即分配，否则跳至下一个到达或最近空闲时间的较小者。

5.3 计算步骤

1. 构造作业流 (A_i, S_i)，设定道数 m；
2. 维护就绪集合与各道最早空闲时间；
3. 依据所选算法分配作业并记录开始/结束时间；
4. 计算 T_i, W_i 及 $\overline{T}, \overline{W}$；
5. 输出并整理为表格，进行对比分析。

5.4 模块组织

Rust 程序采用面向对象设计，主要包含以下模块：

- **Task 结构体**：定义任务的基本属性（任务 ID、到达时间、执行时间、开始时间、完成时间）及相关方法；
- **调度算法函数**：`execute_fcfs_scheduling`、`execute_sjf_scheduling`、`execute_hrrn_scheduli` 分别实现三种调度算法；
- **结果处理函数**：`display_scheduling_results` 负责统计与格式化输出调度结果；
- **测试数据生成**：`generate_test_task_set_1`、`generate_test_task_set_2` 生成不同特征的任务流；
- **主程序**：`main` 函数组装单处理器/双处理器环境与两组任务流的完整实验流程。

5.5 伪代码

以 SJF 算法为例（非抢占式，多处理器环境）：

```
current_time <- 0; processor_available_time[m] <- 0; ready_queue <- {}
按到达时间排序 all_tasks
while 未完成任务:
  将 arrival_time <= current_time 的任务加入 ready_queue
  available_processors <- {k | processor_available_time[k] <= current_time}
  if available_processors 为空:
    if ready_queue 非空: current_time <- 最早 processor_available_time
    else: current_time <- min(下一个到达, 最早 processor_available_time); continue
  if ready_queue 为空: current_time <- 下一个到达; continue
  按 execution_time 升序排序 ready_queue
  对每个空闲处理器:
    取最短任务 task; start_time <- current_time; finish_time <- start_time + executio
    记录 task.start_time/task.finish_time; processor_available_time[processor] <- fin
  current_time <- min(下一个到达, 最早 processor_available_time)
```

6 实验结果及分析

以下给出程序运行结果（单位：分钟）。

程序运行结果

```
操作系统作业调度算法实验
================================

【单处理器环境调度结果】

========== FCFS算法 - 单处理器 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t4.00\t11.00\t9.50\t1.36
3\t3.00\t5.00\t11.00\t16.00\t13.00\t2.60
4\t5.00\t6.00\t16.00\t22.00\t17.00\t2.83
5\t7.00\t3.00\t22.00\t25.00\t18.00\t6.00
平均周转时间: 12.3000
平均带权周转时间: 2.7580
```

========== SJF算法 - 单处理器 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t4.00\t11.00\t9.50\t1.36
3\t3.00\t5.00\t13.00\t18.00\t15.00\t3.00
4\t5.00\t6.00\t18.00\t24.00\t19.00\t3.17
5\t7.00\t3.00\t11.00\t14.00\t7.00\t2.33
平均周转时间: 10.9000
平均带权周转时间: 2.1720

========== HRRN算法 - 单处理器 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t4.00\t11.00\t9.50\t1.36
3\t3.00\t5.00\t11.00\t16.00\t13.00\t2.60
4\t5.00\t6.00\t18.00\t24.00\t19.00\t3.17
5\t7.00\t3.00\t16.00\t19.00\t12.00\t4.00
平均周转时间: 11.5000
平均带权周转时间: 2.4260

【双处理器环境调度结果】

========== FCFS算法 - 双处理器 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t1.50\t8.50\t7.00\t1.00
3\t3.00\t5.00\t3.00\t8.00\t5.00\t1.00
4\t5.00\t6.00\t8.00\t14.00\t9.00\t1.50
5\t7.00\t3.00\t8.50\t11.50\t4.50\t1.50
平均周转时间: 6.0000
平均带权周转时间: 1.2000

========== SJF算法 - 双处理器 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t1.50\t8.50\t7.00\t1.00

3\t3.00\t5.00\t3.00\t8.00\t5.00\t1.00
4\t5.00\t6.00\t8.00\t14.00\t9.00\t1.50
5\t7.00\t3.00\t8.50\t11.50\t4.50\t1.50
平均周转时间: 6.0000
平均带权周转时间: 1.2000

========== HRRN算法 - 双处理器 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t1.50\t8.50\t7.00\t1.00
3\t3.00\t5.00\t3.00\t8.00\t5.00\t1.00
4\t5.00\t6.00\t8.00\t14.00\t9.00\t1.50
5\t7.00\t3.00\t8.50\t11.50\t4.50\t1.50
平均周转时间: 6.0000
平均带权周转时间: 1.2000

【算法性能对比分析】
==============================

========== 任务流A - FCFS算法 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t4.00\t0.00\t4.00\t4.00\t1.00
2\t1.50\t7.00\t4.00\t11.00\t9.50\t1.36
3\t3.00\t5.00\t11.00\t16.00\t13.00\t2.60
4\t5.00\t6.00\t16.00\t22.00\t17.00\t2.83
5\t7.00\t3.00\t22.00\t25.00\t18.00\t6.00
平均周转时间: 12.3000
平均带权周转时间: 2.7580

========== 任务流B - FCFS算法 ==========
任务ID\t到达\t执行\t开始\t结束\t周转\t带权周转
1\t0.00\t9.00\t0.00\t9.00\t9.00\t1.00
2\t0.50\t3.00\t9.00\t12.00\t11.50\t3.83
3\t1.00\t10.00\t12.00\t22.00\t21.00\t2.10
4\t2.00\t4.00\t22.00\t26.00\t24.00\t6.00
5\t8.00\t1.00\t26.00\t27.00\t19.00\t19.00
6\t8.50\t2.00\t27.00\t29.00\t20.50\t10.25

平均周转时间：17.5000
平均带权周转时间：7.0300

实验完成!

6.1 单道（m=1）

任务 ID	到达	执行	开始	结束	周转	带权周转
1	0.00	4.00	0.00	4.00	4.00	1.00
2	1.50	7.00	4.00	11.00	9.50	1.36
3	3.00	5.00	11.00	16.00	13.00	2.60
4	5.00	6.00	16.00	22.00	17.00	2.83
5	7.00	3.00	22.00	25.00	18.00	6.00

平均周转时间 = 12.3000，平均带权周转时间 = 2.7580

FCFS

任务 ID	到达	执行	开始	结束	周转	带权周转
1	0.00	4.00	0.00	4.00	4.00	1.00
2	1.50	7.00	4.00	11.00	9.50	1.36
3	3.00	5.00	13.00	18.00	15.00	3.00
4	5.00	6.00	18.00	24.00	19.00	3.17
5	7.00	3.00	11.00	14.00	7.00	2.33

平均周转时间 = 10.9000，平均带权周转时间 = 2.1720

SJF

任务 ID	到达	执行	开始	结束	周转	带权周转
1	0.00	4.00	0.00	4.00	4.00	1.00
2	1.50	7.00	4.00	11.00	9.50	1.36
3	3.00	5.00	11.00	16.00	13.00	2.60
4	5.00	6.00	18.00	24.00	19.00	3.17
5	7.00	3.00	16.00	19.00	12.00	4.00

平均周转时间 = 11.5000，平均带权周转时间 = 2.4260

HRRN

6.2 双处理器环境（m=2）

三种算法在该任务流下得到相同的调度与指标：

任务 ID	到达	执行	开始	结束	周转	带权周转
1	0.00	4.00	0.00	4.00	4.00	1.00
2	1.50	7.00	1.50	8.50	7.00	1.00
3	3.00	5.00	3.00	8.00	5.00	1.00
4	5.00	6.00	8.00	14.00	9.00	1.50
5	7.00	3.00	8.50	11.50	4.50	1.50

平均周转时间 = 6.0000，平均带权周转时间 = 1.2000

6.3 讨论与分析

- 单处理器环境下：SJF 算法的平均周转时间和平均带权周转时间最低（10.9000 和 2.1720），体现了短作业优先的优势；HRRN 算法介于 FCFS 与 SJF 之间（11.5000 和 2.4260），能够有效缓解长作业饥饿问题；
- 双处理器环境下：由于任务到达时间和执行时间的结构特点，本例中三种算法输出结果一致，且显著优于单处理器环境（平均周转时间从 12.3000 降至 6.0000）；
- 多处理器并行能够显著降低任务等待时间；在实际操作系统中，若考虑 I/O 操作、抢占式调度、优先级等因素，调度策略需要进一步扩展和优化。

7 附录：部分源代码

```
use std::cmp::Ordering;

/// 作业任务结构体
/// 表示一个需要调度的作业任务
#[derive(Clone, Debug)]
struct Task {
    task_id: usize,         // 任务标识符
    arrival_time: f64,      // 任务到达时间（分钟）
    execution_time: f64,    // 预估执行时间（分钟）
    start_time: Option<f64>,  // 实际开始时间
    finish_time: Option<f64>, // 实际完成时间
}
```

```
impl Task {
    /// 创建新的任务实例
    fn create(task_id: usize, arrival_time: f64, execution_time: f64) -> Self {
        Self {
            task_id,
            arrival_time,
            execution_time,
            start_time: None,
            finish_time: None
        }
    }

    /// 计算任务周转时间
    fn calculate_turnaround_time(&self) -> Option<f64> {
        match (self.finish_time, Some(self.arrival_time)) {
            (Some(finish), Some(arrival)) => Some(finish - arrival),
            _ => None,
        }
    }

    /// 计算任务带权周转时间
    fn calculate_weighted_turnaround_time(&self) -> Option<f64> {
        match (self.calculate_turnaround_time(), self.execution_time) {
            (Some(turnaround), exec_time) if exec_time > 0.0 => Some(turnaround / exe
            _ => None,
        }
    }
}

/// 先来先服务调度算法实现
/// 按照任务到达时间顺序进行调度分配
fn execute_fcfs_scheduling(tasks: &[Task], processor_count: usize) -> Vec<Task> {
    let mut task_list: Vec<Task> = tasks.to_vec();
    // 按到达时间排序
    task_list.sort_by(|a, b| a.arrival_time.partial_cmp(&b.arrival_time).unwrap_or(Or

    // 记录每个处理器的空闲时间
```

```
    let mut processor_available_time: Vec<f64> = vec![0.0; processor_count];

    for task in &mut task_list {
        // 找到最早空闲的处理器
        let (processor_index, &available_time) = processor_available_time.iter().enum
            .min_by(|a, b| a.1.partial_cmp(b.1).unwrap_or(Ordering::Equal)).unwrap();

        // 任务开始时间 = max(到达时间, 处理器空闲时间)
        let start_time = task.arrival_time.max(available_time);
        let finish_time = start_time + task.execution_time;

        task.start_time = Some(start_time);
        task.finish_time = Some(finish_time);
        processor_available_time[processor_index] = finish_time;
    }
    task_list
}
```

8 写在最后

8.1 发布地址

- Github: https://github.com/shufufufu/OS-TASKS