调度策略设计与实现报告

郭彦博 2023200398

1. 设计思路

1.1 整体架构

本调度策略采用基于优先级和紧急程度的混合调度算法，主要包含以下组件：

- 任务状态管理：维护每个任务的完整状态信息
- 紧急程度评估：动态评估任务的紧急程度
- 资源调度：分别处理 CPU 和 I/O 资源的分配

1.2 核心算法

1. **任务状态管理**
   - 使用 `TaskState` 结构体记录任务的完整状态
   - 包含任务信息、I/O 状态、剩余时间、松弛时间和紧急程度
   - 通过哈希表实现高效的任务状态查询和更新
2. **紧急程度评估**
   - 引入三级紧急程度：不紧急(0)、较紧急(1)、非常紧急(2)
   - 根据任务的松弛时间动态更新紧急程度
   - 对高优先级任务采用更激进的紧急程度判断
3. **调度策略**
   - CPU 调度：优先处理紧急程度高且优先级高的任务
   - I/O 调度：优先选择高优先级且 I/O 剩余时间短的任务
   - 资源分配：确保 CPU 和 I/O 资源的高效利用

2. 重要问题的解决方案

2.1 任务优先级处理

- 问题：如何平衡高优先级和低优先级任务的执行
- 解决方案：
  - 实现 `compare_priority_urgency` 比较函数
  - 在优先级相同时考虑紧急程度
  - 对高优先级任务采用更宽松的紧急程度判断标准

2.2 截止时间保证

- 问题：如何确保任务在截止时间前完成
- 解决方案：
  - 引入松弛时间概念
  - 动态评估任务紧急程度

- 对即将超时的任务进行优先处理

2.3 I/O 资源管理

- 问题：如何高效管理 I/O 资源
- 解决方案：
  - 跟踪 I/O 操作的剩余时间
  - 优先选择 I/O 剩余时间短的任务
  - 确保不选择当前 CPU 任务进行 I/O 操作

3. 程序性能分析

3.1 优势

1. 任务完成率
   - 在普通负载情况下（测试点1-12）表现良好
   - 高优先级任务的完成率普遍较高
   - 能够有效处理截止时间宽裕的任务
2. 资源利用率
   - CPU 和 I/O 资源分配合理
   - 能够并行处理 CPU 计算和 I/O 操作
   - 避免资源空闲和冲突

3.2 不足

1. 高负载情况
   - 在测试点13-16表现不佳
   - 任务完成率显著下降
   - 超时比（amplification）较高
2. 调度策略
   - 对密集任务的处理不够优化
   - I/O 时间估计可能不够准确
   - 紧急程度判断标准可能需要调整