**进程改进实验概要设计书**

**设计者：沙宇豪** **邹悦阳 马吉提·金恩斯**

**指导教师：栗风永教授**

**（上海电力大学** **计算机科学与技术学院** **上海市** **）**

**作品内容简介**

本项目在Linux系统下实现一个模拟的进程管理与调度程序。该程序不对内核进行修改，而是在用户态中以数据结构和算法模拟操作系统的进程管理功能，包括进程创建、撤销、阻塞、唤醒，以及多种进程调度算法(先来先服务 FCFS、最高优先级优先 HPF、时间片轮转 RR)。程序在运行后将会读取输入文件中定义的一组进程，依次根据指定的调度算法进行调度，并在最终输出文件中给出各进程的完成时间、周转时间、带权周转时间，以及这些指标的平均值，以便对不同算法的性能进行比较。

**设计内容及要求**

**进程管理模拟**

设计内容：在Linux环境下编写程序模拟实现操作系统的进程管理功能。

具体要求：分析研究Linux进程管理源代码，在此基础上设计程序模拟实现进程管理，验证进程调度算法，并对结果进行分析比较。要求有图形化界面。设计须完成以下任务：

1、Linux进程管理源代码分析。

2、设计程序模拟实现进程管理的基本功能，包括进程控制（进程创建、进程撤消、进程阻塞、进程唤醒）、进程调度等。

3、进程调度算法性能比较。进程调度至少支持两种调度算法，要对不同的调度算法的周转时间和带权周转时间进行比较。进程调度算法包括但不限于以下算法：

（1）先来先服务调度算法；

（2）最高优先数优先调度算法；

（3）“轮转法”调度算法。怎么在linux设计一个模拟进程管理的程序，包括进程（进程创建，进程撤销，进程阻塞，进程唤醒），进程调度等并包含图形化界面以及进程调度算法的比较

**联系人：沙宇豪** **联系电话：18121097472 E-mail：**[**13272828044@qq.com**](mailto:13272828044@qq.com)

**基本概念和原理**

1. 数据结构和文件读取

数据结构：定义了一个 PCB 结构体，表示进程控制块，包含进程的 PID、到达时间、运行时间、优先级、剩余时间、状态、开始时间和结束时间等属性。

文件读取：从输入文件中读取进程信息，存储到 PCB 数组中。

2. 进程排序

按到达时间排序：使用 qsort 函数对进程数组按到达时间进行排序，确保先到达的进程优先被处理。

3. 调度算法选择

根据命令行参数选择不同的调度算法，包括：

FCFS（先来先服务）：按到达顺序依次执行进程。

HPF（最高优先级优先）：按优先级高低选择进程执行。

RR（时间片轮转）：每个进程轮流执行固定时间片。

MLFQ（多级反馈队列）：多级队列调度，不同队列有不同的时间片。

HPF\_DYNAMIC（动态优先级最高优先级优先）：动态调整优先级的最高优先级优先调度。

RR\_ADAPTIVE（自适应时间片轮转）：根据进程的行为动态调整时间片。

4. 调度过程

循环执行：在主循环中，检查当前时间是否有新进程到达，如果有，将新进程加入就绪队列。

进程调度：根据选择的调度算法，从就绪队列中选择一个进程执行。

进程状态更新：更新进程的状态（如运行、就绪、终止等），记录进程的开始时间和结束时间。

时间推进：如果没有进程可运行，时间向前推进。

5. 结果计算和输出

计算指标：计算每个进程的周转时间和带权周转时间，并统计所有进程的平均周转时间和平均带权周转时间。

输出结果：将结果写入输出文件，包括每个进程的详细信息和平均指标。

## 项目源码结构分析

data.c:实现了头文件中声明的函数，提供了队列的基本操作功能。使用链表实现了队列，支持高效的入队和出队操作。

data.h:定义了进程控制块（PCB）、队列节点（Node）和队列（Queue）的数据结构。声明了队列的初始化、入队、出队和判断队列是否为空的函数。

process\_control.h:声明了四个进程控制函数：create\_process、terminate\_process、block\_process 和 wake\_process。引入了 data.h 头文件，以便使用 PCB 结构体和其他相关数据结构。

process\_control.c:实现了头文件中声明的四个进程控制函数。提供了创建进程、终止进程、阻塞进程和唤醒进程的功能。这些函数可以用于操作系统中的进程管理，特别是在进程调度和状态转换中。

sched.h:文件定义了与调度算法相关的函数声明，包括 FCFS、HPF、RR、MLFQ、HPF Dynamic 和 RR Adaptive。

sched.c:文件实现了这些调度算法的具体逻辑，包括进程的调度、状态更新、时间管理以及进程终止等操作。

simulate.c :程序用于模拟不同类型的进程调度算法，从输入文件中读取进程信息，根据命令行参数选择调度算法，并将调度结果输出到指定的文件中。

支持的调度算法：先来先服务、最高优先级优先、时间片轮转、多级反馈队列、动态优先级最高优先级优先、自适应时间片轮转

# 3 详细设计

## 3.1数据结构和算法

**主要数据结构：**

PCB(进程控制块)：用于管理和控制进程的一个数据结构

pid: 进程ID

arrival\_time: 进程到达时间

burst\_time: 进程所需运行时间

priority: 进程优先级

remaining\_time: 进程剩余运行时间

state: 进程状态（NEW, READY, RUNNING, BLOCKED, TERMINATED）

start\_time: 进程开始运行时间

finish\_time: 进程完成时间

Queue (就绪队列)：使用链表或数组实现，存储处于READY状态的进程。

**核心调度算法**

1. FCFS（先来先服务调度）:

按到达先后顺序调度，进程3的长运行时间会导致后续进程等待较久，短作业(进程4、5)到达后要等待进程3完成，可能带来较长的周转时间。

2.HPF(基础版优先级调度):

优先运行高优先级的进程(进程3优先级为2，相比其他为1略高)。进程3因优先级高，会阳塞其他低优先级进程，从而可能让后到达的短任务(如进程4、5)一直等待。这可能产生饥饿问题(若有持续更高优先级的进程或后期未提升低优先级进程的优先级，一些进程可能长期无法运行)。

3.RR(固定时间片轮转调度，时间片=2):

给每个进程分配相等的时间片轮转执行，短任务享受相对公平，但频繁上下文切换会增加运行开销。对于长任务(进程3)，在轮转中被多次打断，可能造成平均周转时间和带权周转时间不够理想。

**创新算法与改进效果**

1.MLFQ:

将短任务快速完成(高优先级队列、短时间片)，长任务会逐渐“下沉“到低优先级队列，减少对短任务的阻碍。相比FCFS和基本HPF，短任务(如进程2、4)能更快完成，提高响应时间和周转时间。

对于进程3这种长任务则被分配到低优先级队列中，不阻塞后面短任务的执行，使整体平均周转时间可能优于单纯的HPF或FCFS

2.HPF Dynamic(动态优先级提升):

在原HPF上做了改进。对于长期等待的进程，定期提高其优先级，防止它们长期得不到运行机会。

对本数据集而言，如果进程5在HPF基础版中因进程3(高优先级长任务)和其他任务的阻塞而迟迟未运行，经过一段等待后，进程5的优先级会提升，最终获得执行机会。这样减少了饥饿现象，缩短了极端等待进程的周转时间。

3.RR Adaptive(自适应时间片轮转)

在固定RR的基础上，根据就绪队列中平均剩余时间与进程数动态调整时间片。

当系统负载高、平均剩余时间长时，RR Adaptive会增加时间片，以减少频繁切换，提高整体吞吐量;当负载轻、进程短时，减小时间片加快响应，提高短任务完成速度。

对本数据集而言，在早期有长任务(进程3)时，Adaptive RR可能加大时间片让长任务多跑会儿，减少过多的切换。后期当进程4、5等短任务到来并数量不多时，时间片会略减小，提高这些短任务的响应和完成效率。

## 分析与对比总结

## MILFQ在混合长短任务场景下表现出色:短任务较快速完成，长任务不会长时间阻塞队列前端HPF Dynamic减少了低优先级进程的饥饿情况。与基础HPF相比，加入动态优先级提升后，长时间等待的进程不至于“饿死”，因此平均周转时间的最大值会显著降低，整体公平性提高。

## RR Adaptive在有长任务存在时适度加大时间片，减少频繁切换;在负载轻时降低时间片加快响应，能在一定程度上同时兼顾吞吐量和响应时间。

## 3.2程序运行说明

程序运行环境:C语言

本方案的项目代码结构如下：

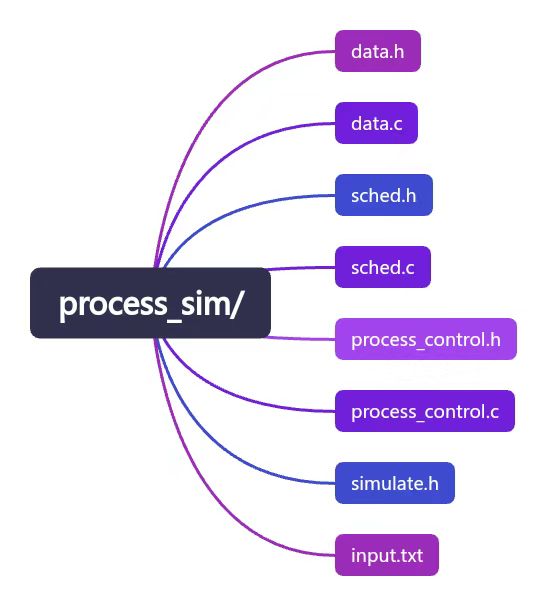


图1 项目代码结构图

项目创新点

本项目的创新点主要体现在以下几个方面：

1. 多种调度算法的集成与比较

本项目通过实现多种经典调度算法（FCFS、HPF、RR、MLFQ、HPF Dynamic 和 RR Adaptive），并对它们的性能进行全面的比较和分析。我们不仅实现了传统的调度算法，还结合创新的策略，如动态优先级调整和自适应时间片轮转，提供了更具实用性的进程调度模拟，帮助用户更好地理解和选择合适的调度策略。

2. 动态优先级提升与自适应时间片轮转

HPF Dynamic（动态优先级调度）：在传统的HPF调度算法基础上，我们为长期等待的低优先级进程引入了动态优先级提升机制。这种机制能够避免进程长期等待而造成的“饥饿”现象，提升了系统的公平性和响应性。

RR Adaptive（自适应时间片轮转调度）：我们创新性地设计了自适应时间片调整策略。在系统负载较高时，增加时间片以减少上下文切换；在负载较低时，减少时间片以提高短进程的响应速度。这种调整机制使得系统在不同负载情况下都能保持较高的吞吐量与响应性。

3. 多级反馈队列调度（MLFQ）

本项目实现的\*\*多级反馈队列调度（MLFQ）\*\*算法，通过多个优先级队列和不同时间片的组合，使得短任务能够优先执行，并减少了长任务的延迟。进程根据执行情况动态地从高优先级队列降级到低优先级队列，从而在保证短任务快速响应的同时，也不至于让长任务完全“饿死”。这种灵活的调度策略可以有效平衡系统的公平性和效率。

4. 图形化用户界面（GUI）

我们为项目设计并实现了一个直观的图形化用户界面，用户可以通过图形界面选择不同的调度算法、设置参数，并实时查看调度过程和结果。这种交互式设计大大提高了用户体验，使得复杂的调度算法和调度过程更容易理解，尤其适合教学和研究用途。

5. 综合性能评估与结果展示

本项目不仅模拟了进程调度的过程，还对各个调度算法的性能进行了定量评估。我们计算了每个进程的周转时间、带权周转时间，并对不同调度算法的平均指标进行了对比分析。这种全面的性能比较，不仅帮助用户了解不同算法在不同场景下的优缺点，还能为实际系统中调度算法的选择提供参考。

6. 系统可扩展性和优化

项目采用模块化设计，便于后续的扩展和优化。新的调度算法、进程控制方式以及用户交互功能可以在现有框架下轻松增加。此外，随着系统对大规模数据集的处理能力逐步增强，我们也考虑到了性能优化，例如减少不必要的文件I/O操作，提高了系统的响应速度。

# 总结

## 5.1设计过程中遇到的问题及解决方法

问题1：在实现某些调度算法（如 fcfs）时，需要按进程的到达时间对进程进行排序。最初使用了简单的冒泡排序，效率较低。

解决方法：改用 qsort 函数进行排序，提高了排序的效率。

问题2：从输入文件中读取进程信息时，发现文件格式不一致或数据错误会导致程序崩溃。

解决方法：在读取文件时增加了错误检查，确保每条记录都符合预期的格式。如果读取失败，程序会输出错误信息并终止。

问题3：在实现不同调度算法时，发现每种算法的逻辑差异较大，特别是对于动态调整时间片的自适应轮转算法 rr\_adaptive，需要根据系统的当前状态动态调整时间片。

解决方法：为每种调度算法单独编写了一个函数，并在主函数中根据命令行参数调用相应的调度函数。

## 5.2团队合作情况

**收获：**

团队成员之间通过频繁的沟通和协作，逐渐形成了默契，提高了工作效率。

在遇到问题时，大家能够共同讨论、集思广益，找到最佳解决方案。团队成员学会了如何分配任务、设定优先级、监控进度和评估成果，这些经验对于未来的项目管理至关重要。通过与其他成员的交流和合作，团队成员有机会接触到自己不熟悉的技术领域，从而拓宽了技术视野。

## 5.3设计总结

**结论：**

本项目实现了一个集成化进程调度模拟系统，结合图形用户界面，用于交互式地选择不同的调度算法并方便直观对比结果。

**展望：**

改善用户体验：引入图表库如Matplotlib或Plotly，在GUI中添加动画效果以展示调度过程，帮助用户更好地理解不同算法的行为特点。

优化性能：针对大规模数据集进行性能分析与优化，减少不必要的文件IO操作，提升整体响应速度。