**课题二：**

**多级反馈队列调度算法**

1. **背景需求**

#### 内容

在现代计算机系统中，操作系统需要在多种类型的任务之间高效调度有限的资源，以满足不同应用的性能需求和用户体验目标。传统的调度算法，例如先来先服务（FCFS）和短作业优先（SJF），在处理混合任务时可能无法同时兼顾公平性和效率。FCFS可能导致某些短任务被长任务阻塞，而SJF虽然优先处理短任务，但对长任务并不公平，可能出现“饥饿”现象

多级反馈队列调度算法（Multilevel Feedback Queue Scheduling, MFQ）综合了多种调度算法优点，既能使高优先级的作业得到响应又能使短作业进程迅速完成。不同于传统的单一调度策略，多级反馈队列算法通过设置多个优先级队列，依据进程的执行历史动态调整其优先级，能够有效地满足不同类型进程的调度需求。这种算法，进程根据其响应时间和执行状态在多个队列之间流动。例如，当进程在某个队列中未能在规定的时间片内完成时，它会被移到优先级较低的队列，并采用不同的调度策略。

#### 1.2 主要功能

· 进程调度优先级动态调整：根据进程的运行状态和执行情况，动态调整其优先级，避免某些进程因长期处于低优先级队列而得不到调度，从而显著改善系统的响应时间和公平性。

· 多级队列管理：通过设计多个就绪队列实现对不同类型进程的分级管理，低优先级队列使用轮转算法（RR），而高优先级队列则使用先来先服务（FCFS）。

· 时间片管理：每个队列中的进程都有一个时间片，如果在时间片内进程未能完成，则会被移动到下一个优先级较低的队列，实现合理的资源分配和公平性。

· 进程迁移机制：通过动态的进程迁移机制，保证高优先级进程的及时处理，同时防止长期未调度的低优先级进程长时间得不到调度。

1. **系统设计**

#### 2.1 架构设计

本课题实现一个模拟多级反馈队列调度算法（MFQ）的程序。主要目标是通过多级队列和时间片轮转策略，实现进程调度和优先级动态调整。系统的架构设计采用模块化方式，分为初始化模块、调度模块和打印模块三个主要部分。每个模块各司其职，协同完成进程的创建、调度和结果展示等功能。具体设计如下：

**（1）初始化模块**：负责初始化系统参数和进程信息。在 initialize 函数中，根据用户的选择（随机生成或键盘输入），生成进程的到达时间、运行时间以及相关的队列设置。初始化进程的状态，包括进程的剩余运行时间、开始时间、完成时间等。此外，还会根据队列数量初始化各个队列的时间片大小。

**（2）调度模块**：executeMFQ 函数通过模拟多级反馈队列调度过程，实现了进程在不同队列之间的迁移。高优先级队列使用先来先服务（FCFS）调度策略，低优先级队列使用轮转算法（RR）。进程会根据剩余的执行时间和时间片的大小，在各个队列间进行迁移。例如，如果进程在当前队列的时间片用尽，它将被移至较低优先级队列；如果进程完成了它的执行，则会从调度队列中移除。

**（3）打印模块**：displayResults 函数负责显示调度结果，包括每个进程的开始时间、运行时间、完成时间、周转时间和带权周转时间。此外，还计算并输出系统的平均周转时间和平均带权周转时间，来评估调度算法的性能。

#### 2.2 数据结构设计

进程信息结构体（PCB）：

- id：进程的唯一标识符。

- arrival：进程的到达时间。

- burst：进程的运行时间。

- remaining：进程剩余运行时间。

- start\_time：进程开始执行的时间。

- finish\_time：进程完成执行的时间。

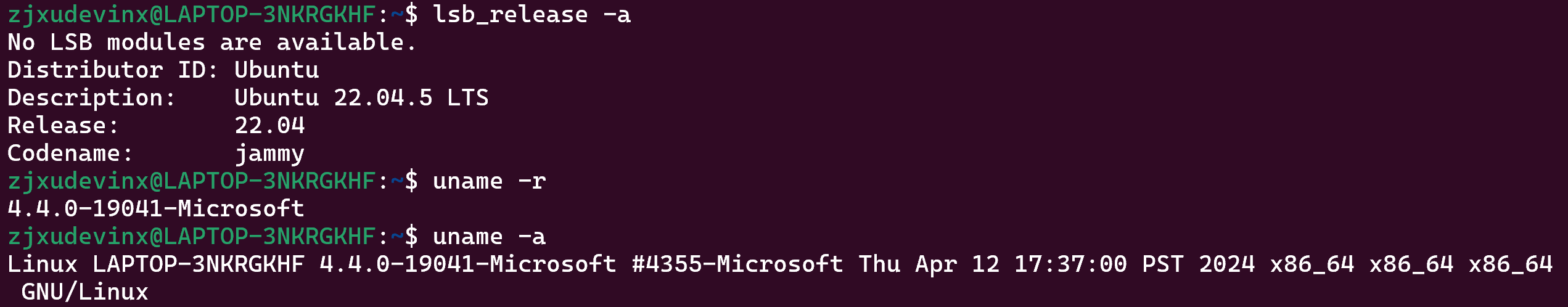
- turnaround\_time：进程的周转时间，计算为finish\_time - arrival。

- weighted\_turnaround\_time：进程的带权周转时间，计算为turnaround\_time / burst。

#### 2.3 实验环境

操作系统：windows 10 家庭版

实验软件：WSL、VSCode



编程语言：C++ 17

1. **详细设计**
   1. 算法流程

该算法通过多个优先级队列来管理进程，并根据进程的执行情况动态调整其所在队列的优先级。每个队列都有不同的时间片大小，通常高优先级队列使用较短的时间片，低优先级队列使用较长的时间片。算法的核心思想和流程如下：

（1）高优先级队列：处理执行时间较短的进程，采用先来先服务（FCFS）调度策略。进程在该队列中会依次执行，按照到达时间的顺序进行调度，直到该进程执行完成或者时间片耗尽。

（2）低优先级队列：处理执行时间较长的进程，采用时间片轮转（RR）调度策略。在此队列中的进程会被分配一个固定的时间片，每个进程会轮流执行。在时间片用尽时，进程如果没有完成，将被移动到下一个更低优先级的队列中，等待更长时间的时间片进行执行。

（3）动态调整：进程最开始会被分配到高优先级队列中。如果进程在分配的时间片内没有完成，它会被迁移到低优先级队列。在低优先级队列中，进程继续执行，直到用尽分配的时间片。如果进程仍未完成，它将被迁移到更低优先级的队列，并继续等待执行。

（4）进程的迁移：如果有新的进程到达，系统会中断当前正在执行的进程，将其放回原队列的队尾，执行新到达的进程。

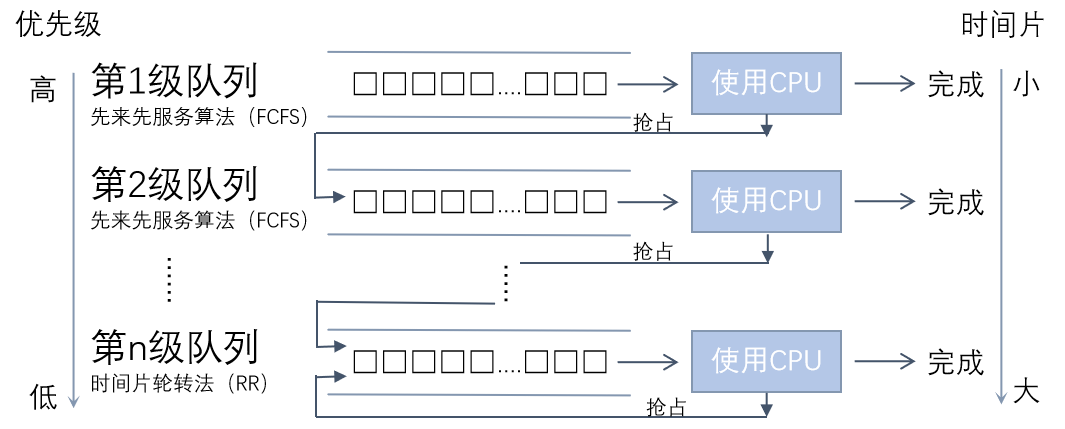


图3-1算法示意图

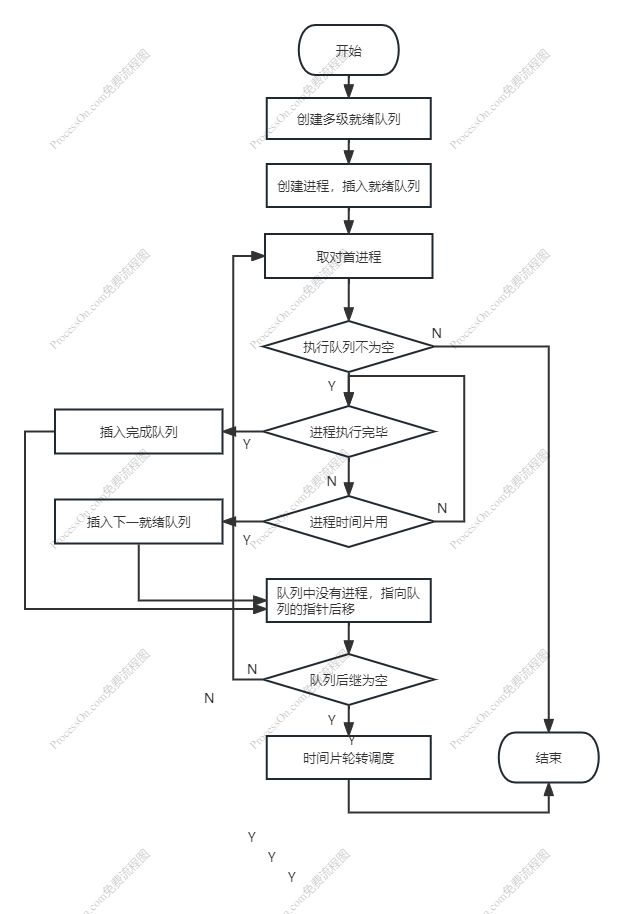


图3-2算法流程图

* 1. 函数功能描述

|  |  |
| --- | --- |
| 函数 | 功能描述 |
| initialize(int option) | 初始化进程调度的相关参数，根据用户输入或随机生成的方式初始化进程的到达时间、运行时间、就绪队列的数量以及各队列的时间片大小。根据输入的选项，生成一定数量的进程，并确保进程的到达时间不会重复。 |
| executeMFQ() | 多级反馈队列调度的核心函数。该函数通过多个队列来处理不同优先级的进程，并且根据进程的执行情况在队列之间进行迁移。每个队列使用不同的调度策略（如 FCFS 或 RR），并根据进程的剩余运行时间进行调度。 |
| displayProcesses() | 打印当前的进程信息，包括进程的到达时间、结束时间、运行时间、就绪队列等。 |
| displayResults() | 用于显示当前系统中所有进程的信息，方便用户查看进程的基本属性（如ID、到达时间、执行时间等）。 |

1. **编码**

#include **<**iostream**>**#include **<**algorithm**>**#include **<**queue**>**#include **<**vector**>**#include **<**ctime**>**#include **<**set**>**#include **<**map**>**#include **<**iomanip**>***//* *常量定义*#define MAX\_PROCESSES 100  
#define MAX\_QUEUES 5  
  
*//* *PCB结构体***struct** PCB  
**{**    **int** id**;**                          *//* *进程ID*    **int** arrival**;**                     *//* *到达时间*    **int** burst**;**                       *//* *运行时间*    **int** remaining**;**                   *//* *剩余运行时间*    **int** start\_time**;**                  *//* *开始执行时间*    **int** finish\_time**;**                 *//* *完成时间*    **int** turnaround\_time**;**             *//* *周转时间*    **double** weighted\_turnaround\_time**;** *//* *带权周转时间***};***//* *全局变量***int** num\_queues**;**                              *//* *就绪队列数量***int** num\_processes**;**                           *//* *进程数量***int** time\_quantum**[**MAX\_QUEUES**];**                *//* *每个队列的时间片大小*std**::**queue**<**PCB**>** ready\_queues**[**MAX\_QUEUES**];**    *//* *多级就绪队列*std**::**vector**<**PCB**>** all\_processes**;**              *//* *所有进程信息*std**::**map**<int,** **int>** process\_map**;**              *//* *进程ID到索引的映射***int** total\_turnaround\_time **=** 0**;**               *//* *周转总时间***double** total\_weighted\_turnaround\_time **=** 0.0**;** *//* *带权周转总时间  
  
//* *函数声明***void** initialize**(int** option**);  
void** executeMFQ**();  
void** displayResults**();  
void** displayProcesses**();***//* *主函数***int** main**()  
{**    srand**(**time**(**0**));**    **while** **(true)**    **{**        std**::**cout **<<** **"请选择测试样例生成方式：**\n**1. 随机生成**\n**2. 键盘输入**\n**3. 退出**\n**";**        **int** choice**;**        std**::**cin **>>** choice**;**        **if** **(**choice **==** 1**)**        **{**            std**::**cout **<<** **"随机生成:**\n**";**            initialize**(**1**);**        **}**        **else** **if** **(**choice **==** 2**)**        **{**            std**::**cout **<<** **"键盘输入:**\n**";**            initialize**(**2**);**        **}**        **else**        **{**            **break;**        **}**        executeMFQ**();**        displayResults**();**    **}**    **return** 0**;  
}***//* *展示进程信息***void** displayProcesses**()  
{**    std**::**cout **<<** **"就绪队列总数："** **<<** num\_queues **<<** '\n'**;**    std**::**cout **<<** **"第一个就绪队列时间片大小："** **<<** time\_quantum**[**0**]** **<<** '\n'**;**    std**::**cout **<<** **"进程总数："** **<<** num\_processes **<<** '\n'**;**    **for** **(const** **auto** **&**pcb **:** all\_processes**)**    **{**        std**::**cout **<<** **"进程 "** **<<** pcb**.**id  
                  **<<** **" - 到达时间："** **<<** pcb**.**arrival  
                  **<<** **"，运行时间："** **<<** pcb**.**burst **<<** '\n'**;**    **}  
}***//* *初始化函数***void** initialize**(int** option**)  
{**    std**::**set**<int>** arrival\_times**;** *//* *用于避免重复到达时间*    **if** **(**option **==** 1**)**    **{** *//* *随机生成*        num\_queues **=** rand**()** **%** 5 **+** 1**;**        time\_quantum**[**0**]** **=** rand**()** **%** 10 **+** 1**;**        **for** **(int** i **=** 1**;** i **<** num\_queues**;** **++**i**)**            time\_quantum**[**i**]** **=** time\_quantum**[**i **-** 1**]** **\*** 2**;**        num\_processes **=** rand**()** **%** 10 **+** 1**;**        all\_processes**.**resize**(**num\_processes**);**        **for** **(int** i **=** 0**;** i **<** num\_processes**;** **++**i**)**        **{**            PCB **&**pcb **=** all\_processes**[**i**];**            pcb**.**id **=** i **+** 1**;**            pcb**.**arrival **=** rand**()** **%** 20 **+** 1**;**            **while** **(**arrival\_times**.**count**(**pcb**.**arrival**))**                pcb**.**arrival **=** rand**()** **%** 20 **+** 1**;**            arrival\_times**.**insert**(**pcb**.**arrival**);**            pcb**.**burst **=** rand**()** **%** 50 **+** 1**;**            pcb**.**remaining **=** pcb**.**burst**;**            pcb**.**start\_time **=** pcb**.**finish\_time **=** **-**1**;**            pcb**.**turnaround\_time **=** **-**1**;**            pcb**.**weighted\_turnaround\_time **=** **-**1**;**        **}**    **}**    **else**    **{** *//* *键盘输入*        std**::**cout **<<** **"请输入就绪队列数量(最大为5): ";**        std**::**cin **>>** num\_queues**;**        std**::**cout **<<** **"请输入第一个就绪队列时间片大小: ";**        std**::**cin **>>** time\_quantum**[**0**];**        **for** **(int** i **=** 1**;** i **<** num\_queues**;** **++**i**)**            time\_quantum**[**i**]** **=** time\_quantum**[**i **-** 1**]** **\*** 2**;**        std**::**cout **<<** **"请输入进程数量: ";**        std**::**cin **>>** num\_processes**;**        all\_processes**.**resize**(**num\_processes**);**        **for** **(int** i **=** 0**;** i **<** num\_processes**;** **++**i**)**        **{**            PCB **&**pcb **=** all\_processes**[**i**];**            pcb**.**id **=** i **+** 1**;**            std**::**cout **<<** **"请输入第 "** **<<** pcb**.**id **<<** **" 个进程的到达时间: ";**            std**::**cin **>>** pcb**.**arrival**;**            **while** **(**arrival\_times**.**count**(**pcb**.**arrival**))**            **{**                std**::**cout **<<** **"到达时间重复，请重新输入: ";**                std**::**cin **>>** pcb**.**arrival**;**            **}**            arrival\_times**.**insert**(**pcb**.**arrival**);**            std**::**cout **<<** **"请输入第 "** **<<** pcb**.**id **<<** **" 个进程的运行时间: ";**            std**::**cin **>>** pcb**.**burst**;**            pcb**.**remaining **=** pcb**.**burst**;**            pcb**.**start\_time **=** pcb**.**finish\_time **=** **-**1**;**            pcb**.**turnaround\_time **=** **-**1**;**            pcb**.**weighted\_turnaround\_time **=** **-**1**;**        **}**    **}**    std**::**sort**(**all\_processes**.**begin**(),** all\_processes**.**end**(),**              **[](const** PCB **&**a**,** **const** PCB **&**b**)**              **{** **return** a**.**arrival **<** b**.**arrival**;** **});**    **for** **(int** i **=** 0**;** i **<** num\_processes**;** **++**i**)**        process\_map**[**all\_processes**[**i**].**id**]** **=** i**;**    displayProcesses**();  
}***//* *多级反馈队列调度算法***void** executeMFQ**()  
{**    **int** current\_queue **=** 0**;**                       *//* *当前所在队列*    **int** current\_process **=** 0**;**                     *//* *当前进程*    **int** current\_time **=** all\_processes**[**0**].**arrival**;** *//* *当前时间*    **int** process\_running **=** 1**;**                     *//* *当前正在进行的进程数*    *//* *将下一个进程加入就绪队列*    **auto** nextProcess **=** **[&]()**    **{**        current\_process**++;**        ready\_queues**[**0**].**push**(**all\_processes**[**current\_process**]);**        current\_time **=** all\_processes**[**current\_process**].**arrival**;**        current\_queue **=** 0**;**        process\_running**++;**    **};**    *//* *将第一个进程放入第一个队列*    ready\_queues**[**0**].**push**(**all\_processes**[**0**]);**    **while** **(true)**    **{**        **if** **(**process\_running **==** 0**)**        **{**            **if** **(**current\_process **==** num\_processes **-** 1**)**                **break;** *//* *所有进程已运行完毕*            **else**                nextProcess**();** *//* *将下一个进程加入就绪队列*        **}**        **while** **(!**ready\_queues**[**current\_queue**].**empty**())**        **{**            PCB now **=** ready\_queues**[**current\_queue**].**front**();**            ready\_queues**[**current\_queue**].**pop**();**            **int** tmp **=** time\_quantum**[**current\_queue**];**            **if** **(**current\_process **+** 1 **<** num\_processes**)**                tmp **=** std**::**min**(**tmp**,** all\_processes**[**current\_process **+** 1**].**arrival **-** current\_time**);**            **if** **(**now**.**remaining **<=** tmp**)**            **{** *//* *当前进程先运行完*                **if** **(**now**.**start\_time **==** **-**1**)**                    now**.**start\_time **=** current\_time**;** *//* *记录开始时间*                current\_time **+=** now**.**remaining**;**                all\_processes**[**process\_map**[**now**.**id**]].**finish\_time **=** current\_time**;**                process\_running**--;**            **}**            **else** **if** **(**current\_process **+** 1 **>=** num\_processes **||**                     **(**time\_quantum**[**current\_queue**]** **<=** all\_processes**[**current\_process **+** 1**].**arrival **-** current\_time**))**            **{**                *//* *时间片已用完*                **if** **(**now**.**start\_time **==** **-**1**)**                    now**.**start\_time **=** current\_time**;** *//* *记录开始时间*                current\_time **+=** time\_quantum**[**current\_queue**];**                now**.**remaining **-=** time\_quantum**[**current\_queue**];**                **if** **(**current\_queue **==** num\_queues **-** 1**)**                    ready\_queues**[**current\_queue**].**push**(**now**);** *//* *最后一级队列RR*                **else**                    ready\_queues**[**current\_queue **+** 1**].**push**(**now**);** *//* *FCFS*            **}**            **else**            **{** *//* *时间片未用完，且下一个进程先到*                **if** **(**now**.**start\_time **==** **-**1**)**                    now**.**start\_time **=** current\_time**;** *//* *记录开始时间*                now**.**remaining **-=** **(**all\_processes**[**current\_process **+** 1**].**arrival **-** current\_time**);**                ready\_queues**[**current\_queue**].**push**(**now**);**                nextProcess**();** *//* *将下一个进程加入就绪队列*            **}**        **}**        **if** **(**current\_queue **+** 1 **<** num\_queues**)**            current\_queue**++;**    **}  
}***//* *打印调度结果***void** displayResults**()  
{**    std**::**cout **<<** std**::**fixed **<<** std**::**setprecision**(**2**);**    std**::**cout **<<** **"**\n**进程调度结果:**\n**";**    std**::**cout **<<** **"进程ID**\t**开始时间**\t**运行时间**\t**完成时间**\t**周转时间**\t**带权周转时间**\n**";**    **for** **(int** j **=** 0**;** j **<** num\_processes**;** **++**j**)**    **{**        **int** i **=** process\_map**[**j **+** 1**];**        all\_processes**[**i**].**turnaround\_time **=** all\_processes**[**i**].**finish\_time **-** all\_processes**[**i**].**arrival**;**        all\_processes**[**i**].**weighted\_turnaround\_time **=** 1.0 **\*** all\_processes**[**i**].**turnaround\_time **/** all\_processes**[**i**].**burst**;**        total\_turnaround\_time **+=** all\_processes**[**i**].**turnaround\_time**;**        total\_weighted\_turnaround\_time **+=** all\_processes**[**i**].**weighted\_turnaround\_time**;**        std**::**cout **<<** all\_processes**[**i**].**id **<<** **"**\t**"**                  **<<** all\_processes**[**i**].**arrival **<<** **"**\t\t**"**                  **<<** all\_processes**[**i**].**burst **<<** **"**\t\t**"**                  **<<** all\_processes**[**i**].**finish\_time **<<** **"**\t\t**"**                  **<<** all\_processes**[**i**].**turnaround\_time **<<** **"**\t\t**"**                  **<<** all\_processes**[**i**].**weighted\_turnaround\_time **<<** **"**\n**";**    **}**    std**::**cout **<<** **"**\n**平均周转时间: "** **<<** **(**total\_turnaround\_time **/** **(double)**num\_processes**)** **<<** '\n'**;**    std**::**cout **<<** **"平均带权周转时间: "** **<<** **(**total\_weighted\_turnaround\_time **/** **(double)**num\_processes**)** **<<** '\n'**;  
}**

1. **测试**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 条件 | | 测试结果 | 测试过程 |
| num\_queues | num\_processes |
| 1 | 4 | 6 | OK | 图5-1 |
| 2 | 5 | 9 | OK | 图5-2 |
| 3 | 1 | 4 | OK | 图5-3 |
| 4 | 3 | 8 | OK | 图5-4 |
| 5 | 3 | 9 | OK | 图5-5 |

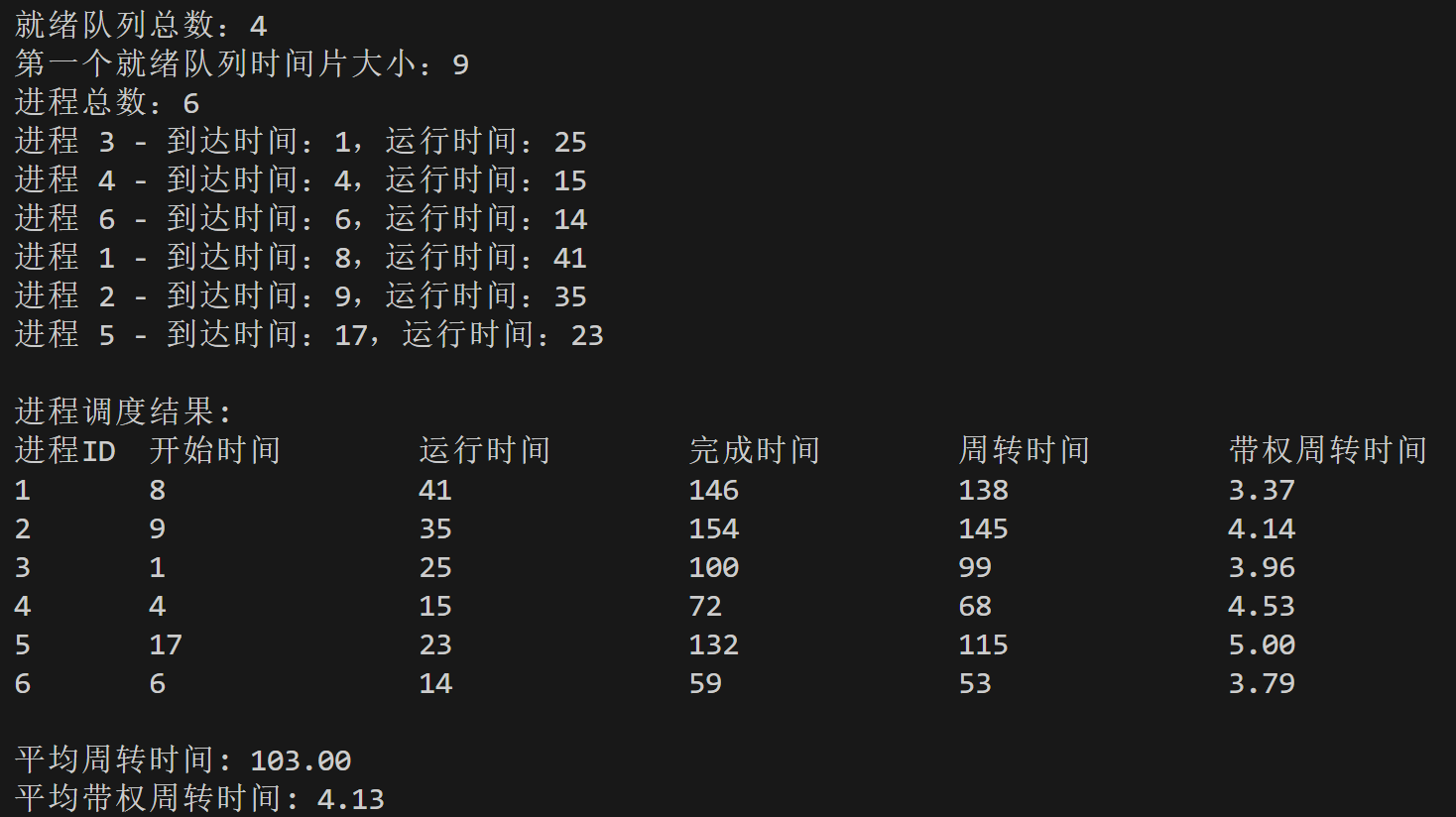


图5-1

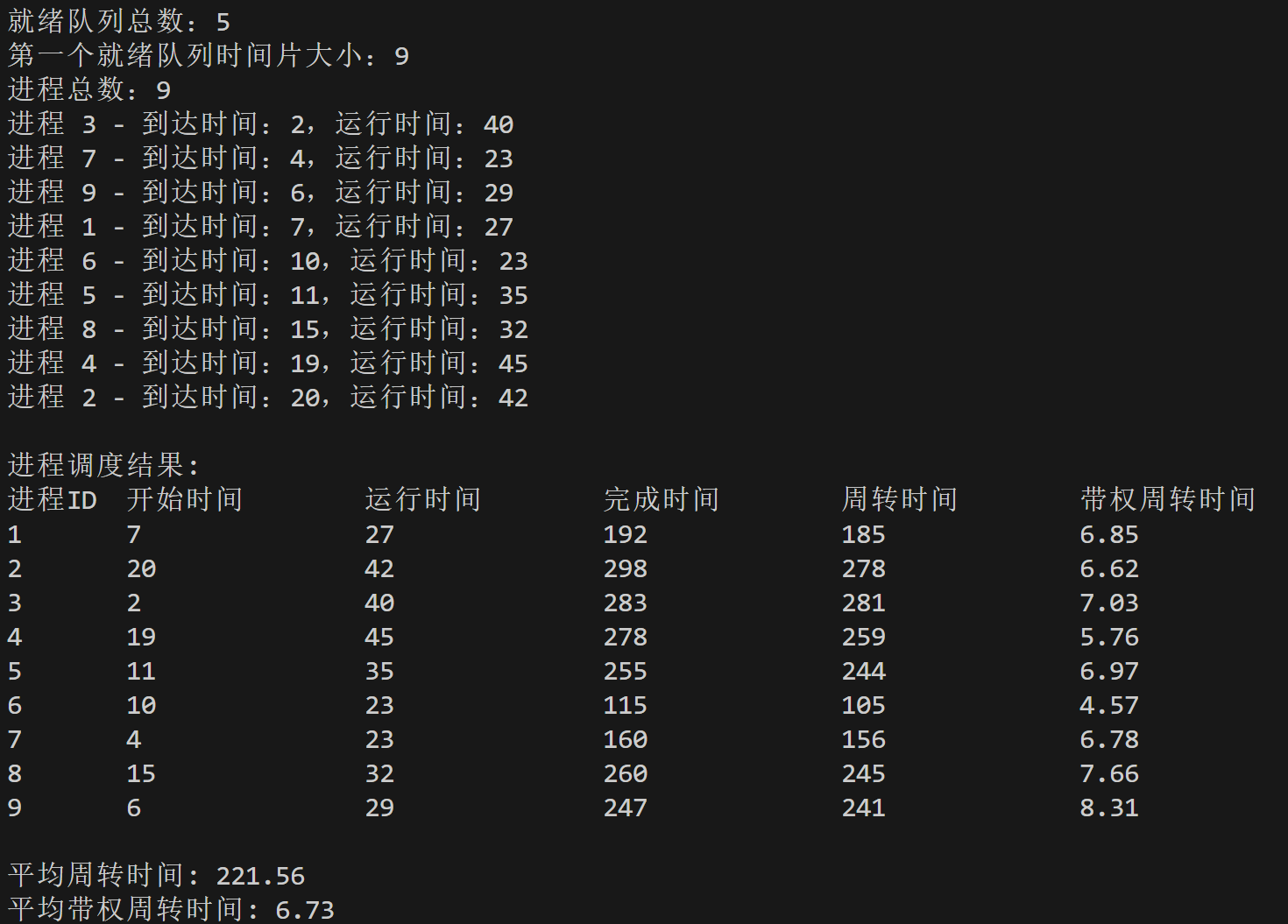


图5-2

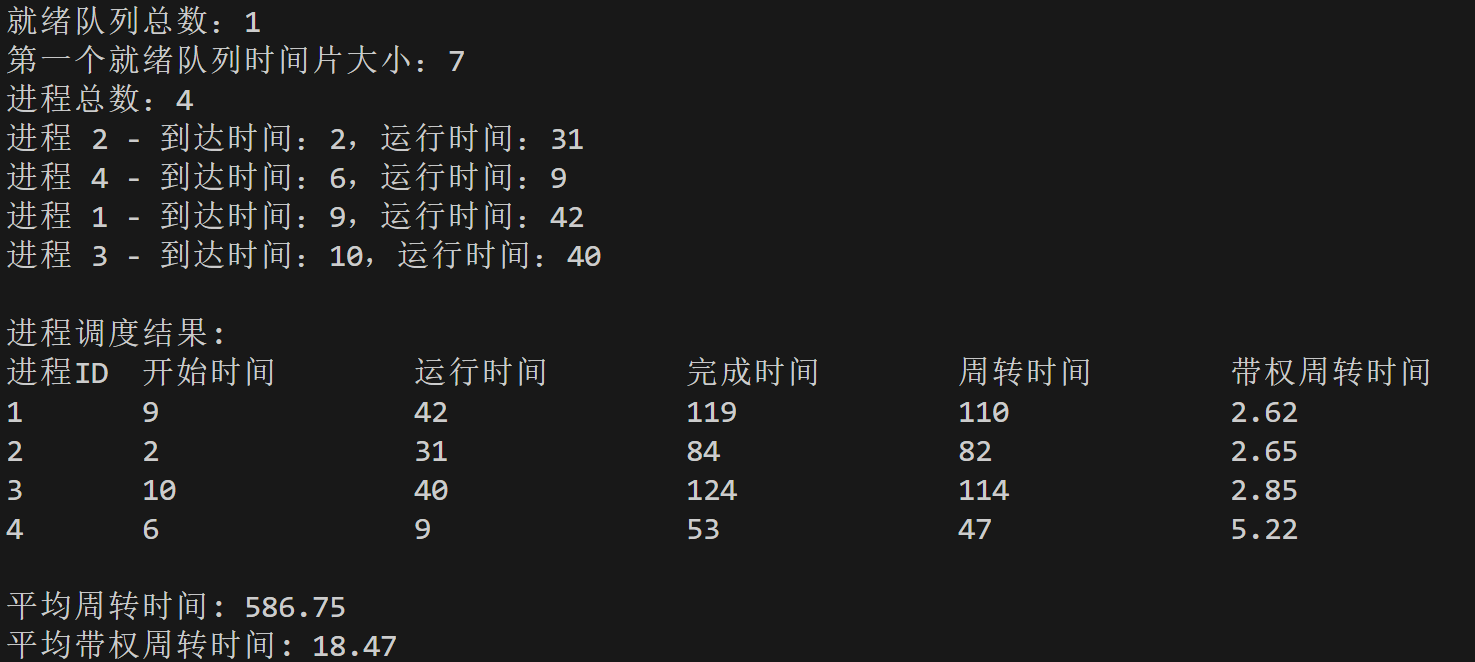


图5-3

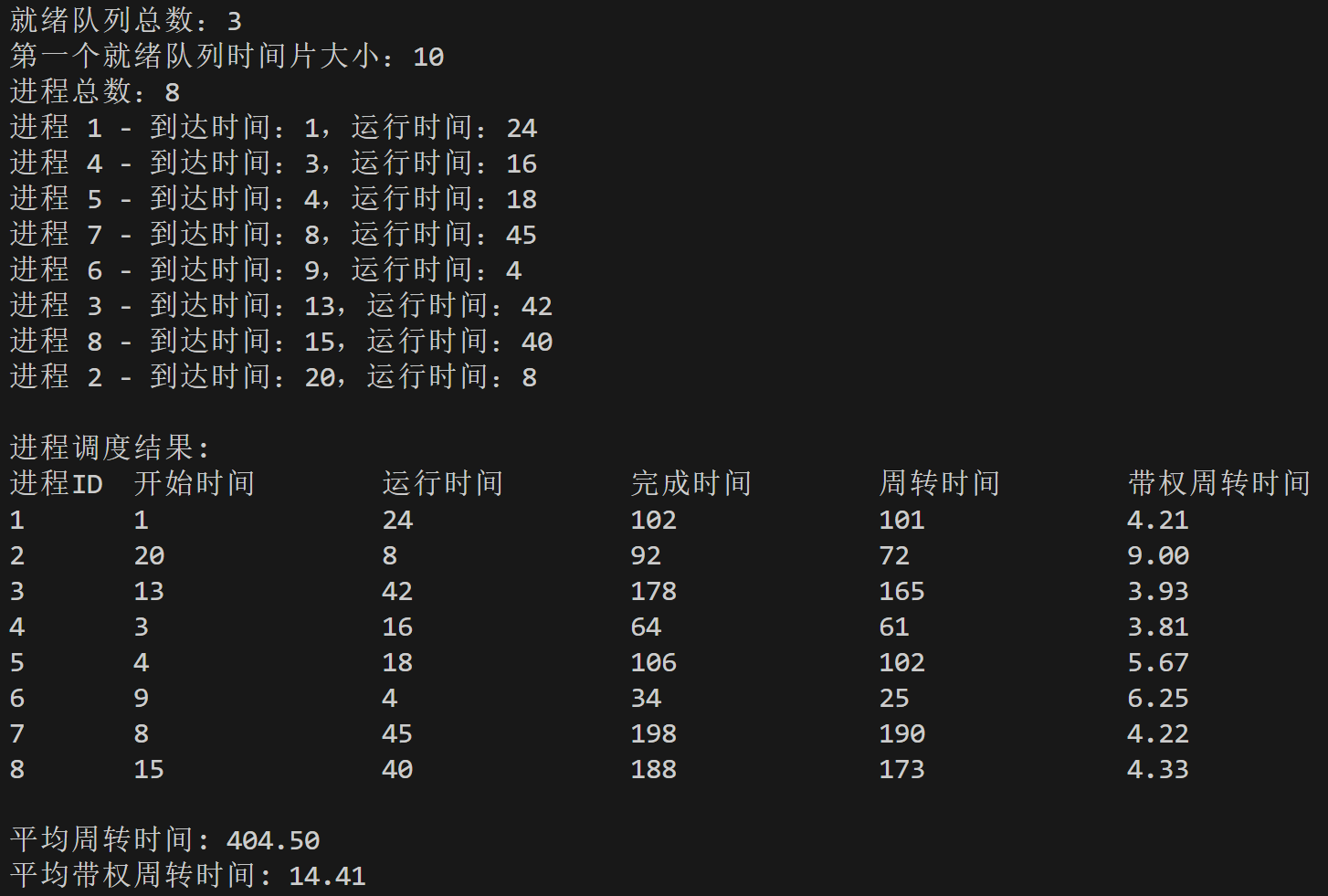


图5-4

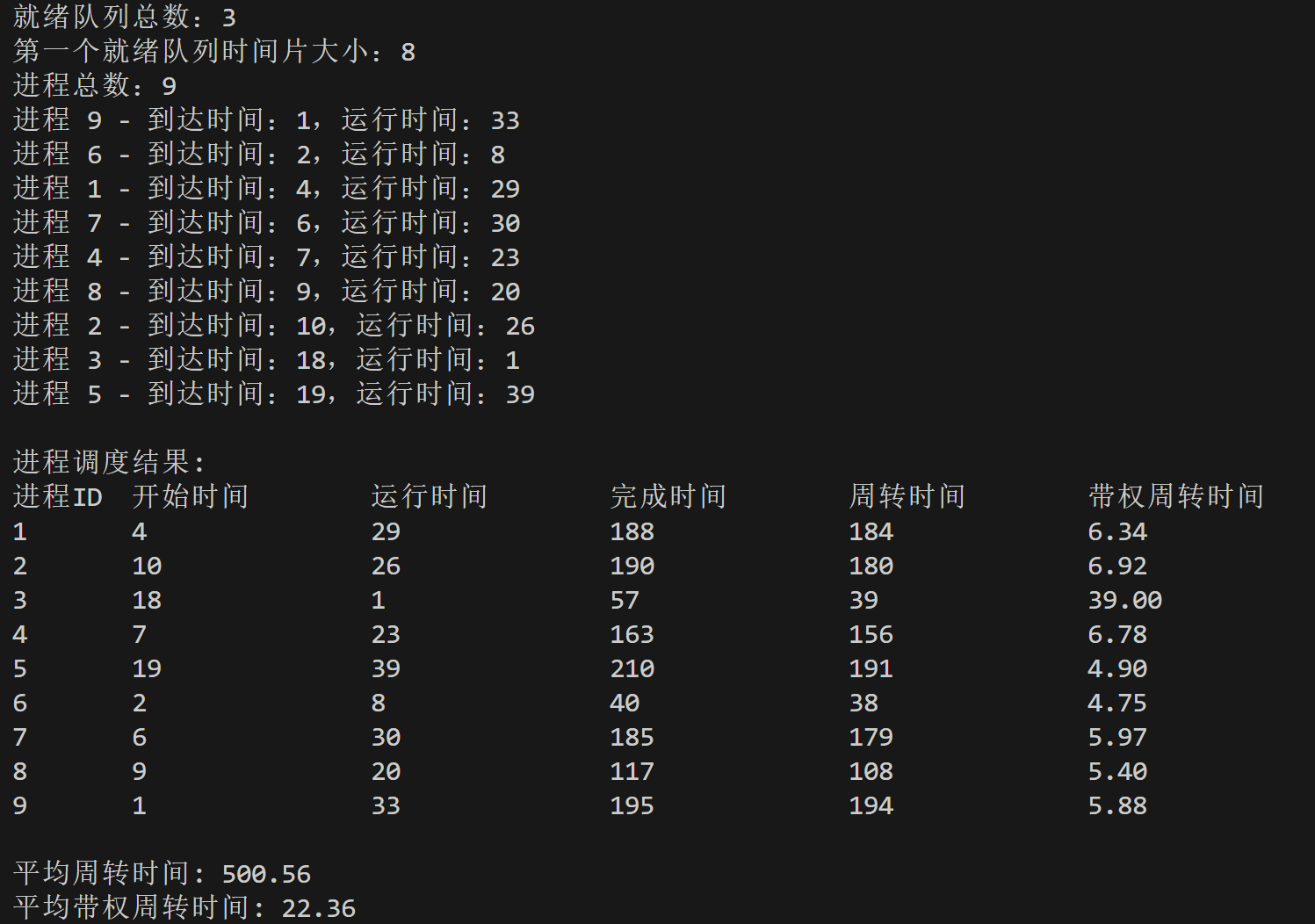


图5-5

1. **总结**

#### 多级反馈队列调度算法既能使高优先级的作业得到响应又能使短作业（进程）迅速完成。它对各个进程的统筹调度安排非常合理，使得每个进程都有机会执行，不会导致“饥饿”现象的发生。通过这个课题，我深入理解了多级反馈队列（MFQ）调度算法的原理和实现方式。多级反馈队列调度算法的核心在于以下三个关键特性：（1）首次加入的进程进入最高优先级队列尾部：当新进程到达时，它会被添加到系统的最高优先级队列的尾部，等待执行。这个队列优先处理较短的进程，因为它使用较短的时间片。(2)时间片用尽后的进程移至次级队列：如果某个队列中的进程执行时未完成且时间片已用尽，进程将被移至下一级优先级队列的尾部，继续等待执行。这样，长时间未完成的进程会逐渐进入低优先级队列，直到最终使用轮转算法（RR）处理。(3)高优先级队列的进程会抢占低优先级队列中的进程：一旦有新进程进入高优先级队列，系统会中断正在低优先级队列中运行的进程，优先执行新到的高优先级进程。这个机制确保了短任务能够迅速执行，而长任务则在多次时间片轮换后才会获得执行机会。

#### 实验中，通过采用多级队列和时间片轮转机制，我成功实现了进程的动态调度及其优先级的动态调整。这一过程加深了我对多级反馈队列（MFQ）调度算法的理解，还让我掌握了编程实现复杂调度逻辑和进程时间管理的技能。通过对进程周转时间和加权周转时间的计算，我更加深刻地认识到调度算法对系统性能的重要影响。