



**课程设计报告**

**课程名称： 操作系统**

**设计题目：基于Qt框架的操作系统综合算法模拟系统**

**专 业： 人工智能**

**姓 名： 李欢欢**

**学 号 2020904302**

**同 组 人： 刘易行**

**指导教师： 李芳**

**二零二三年 六月**

目录

[一、课程设计任务 2](#_Toc142055253)

[二、课程设计需求分析 3](#_Toc142055254)

[2.1、Qt可视化框架 4](#_Toc142055255)

[2.2、各子模块需求分析 4](#_Toc142055256)

[三、课程设计软硬件环境： 5](#_Toc142055257)

[四、各子模块详细设计： 5](#_Toc142055258)

[4.1、作业调度算法周转时间衡量模块 5](#_Toc142055259)

[4.2、处理机调度系统可视化模拟 7](#_Toc142055260)

[4.3、页面置换算法模拟 10](#_Toc142055261)

[4.4、进程管理器模块 12](#_Toc142055262)

[五、程序流程图与流程讲解 13](#_Toc142055263)

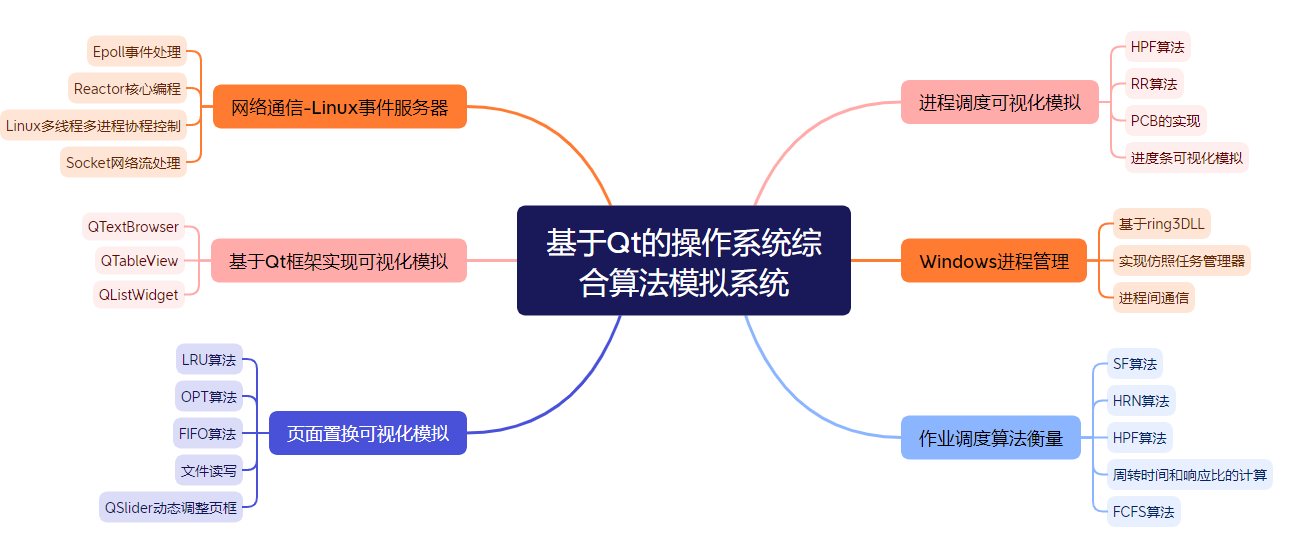
[六、设计演示 15](#_Toc142055264)

[七、任务分工 19](#_Toc142055265)

[八、心得体会： 19](#_Toc142055266)

# 一、课程设计任务

本次课程设计的主题为基于Qt框架的操作系统综合算法模拟系统。该系统包括以下几个模块：处理机调度可视化模拟、调度算法的周转时间的计算与衡量、页面调度淘汰算法的可视化模拟、磁道寻道算法模拟以及仿制Windows进程管理器。每个模块都是基于操作系统的重要算法，并且都有广泛的应用场景。



首先，处理机调度可视化模拟是本系统的核心模块之一，它包含了三种调度算法：轮转调度(RR)、高响应比优先调度(HPF)和最短进程优先调度(SF)。轮转调度算法是一种时间片轮转法，它能够实现公平的进程调度。高响应比优先调度算法则是根据进程等待时间和服务时间的比值进行调度，能够优先执行等待时间长的进程。最短进程优先调度算法则是选择执行时间最短的进程进行调度，能够最大限度地提高系统的运行效率。本模块还需要实现动态可视化模拟，使用户可以直观地了解到各种调度算法的运行过程。

其次，调度算法的周转时间的计算与衡量是为了评估调度算法的优劣而设置的模块。通过该模块，用户可以计算出某个调度算法所用时间和周转时间。基于这些关键数据，用户可以选择和优化调度算法，以提高系统的运行效率和性能。

第三，页面调度淘汰算法的可视化模拟是模拟操作系统要进行页面置换的情况。在本模块中，包含了文件读写、先进先出(FIFO)算法、最近最少使用(LRU)算法和最佳置换(OPT)算法。页面置换算法是操作系统的重要组成部分，本模块通过动态可视化的方式，使用户能够直观地认识到不同算法的特点和效果。

第四，磁道寻道算法模拟是为了模拟操作系统磁盘磁头寻道过程而设计的模块。该模块包括了扫描(SCAN)算法、循环扫描(CSCAN)算法和最短寻道优先(SSPF)算法。本模块通过动态可视化的方式，使用户能够清楚地看到磁头的运动轨迹和各种算法带来的影响。

最后，仿制Windows进程管理器是为了模拟Windows操作系统进程管理器而设计的模块。本模块提供了进程查看、进程优先级设置、进程杀死等操作，模拟了操作系统进程管理的一些关键功能。

# 二、课程设计需求分析

在进行课程设计之前，我们需要对该系统的需求进行详细的分析和说明。本次课程设计的主题为基于Qt框架的操作系统综合算法模拟系统，其中包含了多个模块，每个模块都有不同的功能和需求。

## 2.1、Qt可视化框架

Qt是一种基于C++的跨平台应用程序开发框架，包含了一套功能强大、易用高效的可视化GUI设计工具。Qt框架可以用于开发各种类型的应用，包括桌面应用、嵌入式应用、移动应用和自动化测试等。

Qt可视化框架提供了一套完整的GUI控件库，包括界面元素、图形和多媒体支持等组件。它还提供了一套丰富的工具和接口，方便开发者设计和实现高度定制化的用户界面。Qt的一大特点是它的跨平台性，无论是在Windows、Linux还是Mac OS X上，开发者都可以使用同一份代码来运行应用。

Qt可视化框架也支持现代化C++特性，如移动语义和lambdas。它还提供了自己的Signal和Slot通信机制，可以让开发者轻松地实现对象之间的事件响应和消息交换。这也是Qt框架非常流行的原因之一。



我们课程设计的每一个子模块都是一个Qt设计师页面类，在Qt框架中，设计师页面类是一种用于创建用户界面的重要工具。它是Qt Designer的默认文件格式，也可以通过代码进行加载和编辑。设计师页面类通常以.ui文件后缀结尾，并将用户界面的组件布局和属性细节保存在其中。

设计师页面类由Qt框架内置的Qt Designer工具创建，其中包含了各种用于创建用户界面的组件，如按钮、文本框、标签、下拉菜单等。设计师页面类还可以根据应用程序的需要进行自定义，例如添加自定义的QWidgets或出口信号。

设计师页面类的最大优势是可以通过简单的拖放和编辑操作来创建GUI界面，节省了很多手动编程的工作，同时在设计后仅需生成C++代码即可在应用程序中使用。此外，Qt框架提供了一些高级功能，如可视化布局编辑器、静态分析和调试，可帮助开发者快速开发和调试用户界面。

## 2.2、各子模块需求分析

首先，处理机调度可视化模拟模块需要实现进程调度的可视化模拟和算法分析。该模块在处理机调度算法应用中占据着重要的地位，通过动态可视化的方式，使得用户能够直观地了解进程的调度过程，并且能够分析和比较不同调度算法的优缺点，以便于选择和优化调度算法。

其次，调度算法的周转时间的计算与衡量模块是为了评估调度算法的优劣而设置的。该模块需要计算出某个调度算法所需时间和周转时间，以此来评估和比较不同算法的优劣。通过该模块，用户可以对系统的性能进行评估和优化。

第三，页面调度淘汰算法的可视化模拟是操作系统中另一个重要的模块，该模块需要深入模拟操作系统中页面置换算法的过程。通过对文件读写、先进先出(FIFO)算法、最近最少使用(LRU)算法和最佳置换(OPT)算法的模拟，用户能够对操作系统中的页面置换算法有更加深刻的了解和认识。

第四，磁道寻道算法模拟模块需要模拟操作系统中的磁道寻道过程，该模块需要实现磁头的运动轨迹以及各种算法的模拟。通过该模块，用户可以了解和理解磁道寻道算法的工作原理和优劣，以便于选择和优化磁道寻道算法。

最后，仿制Windows进程管理器模块需要实现操作系统进程管理器的一些关键功能和操作。通过对进程查看、进程优先级设置、进程杀死等操作的模拟，用户能够学习和了解Windows操作系统中进程管理的一些基本知识和技能。

综上所述，本次课程设计涉及的模块都是操作系统中的重要模块，对于深入了解操作系统各种算法和原理、提高操作系统实际应用能力都有很大的帮助。在设计和实现该系统时，我们需要认真分析每个模块的需求，以便于实现一个功能完备、性能高效的系统。

# 三、课程设计软硬件环境：

* 硬件环境：笔记本电脑，联想拯救者Y7000
* 软件环境：Qt5.12.9、Qt Creator4.0.1、C++11体系语法标准
* 操作系统：进程管理器需运行在Windows7及以上版本的操作系统内核中
* 网络通信服务器程序需运行在Linux系统中
* 编译器：g++7.0.1编译器

# 四、各子模块详细设计：

## 4.1、作业调度算法周转时间衡量模块

首先定义了进程控制块结构体pcb\_cal用于存储作业的相关信息，定义了一个名为pcb\_cal的结构体，用于控制进程调度中的算数周转。该结构体包含了进程的名称、服务时间、结束时间、到达时间、响应比、周转时间、带权周转时间、标记等信息。这些信息可以用于计算进程的周转时间和带权周转时间，这是作业调度算法中的重要指标。在作业调度算法中，周转时间是指从进程提交到完成所经过的时间，带权周转时间是指周转时间与服务时间的比值。因此，这段代码与作业调度算法周转时间计算模块的关系是，它提供了计算周转时间和带权周转时间所需的数据结构和变量。

//链表结构PCB

struct pcb\_cal

{

    string name;    //名称

    int service;    //服务事件

    int finish;     //结束时间

    int arrive;     //到达时间

    float response; //响应比

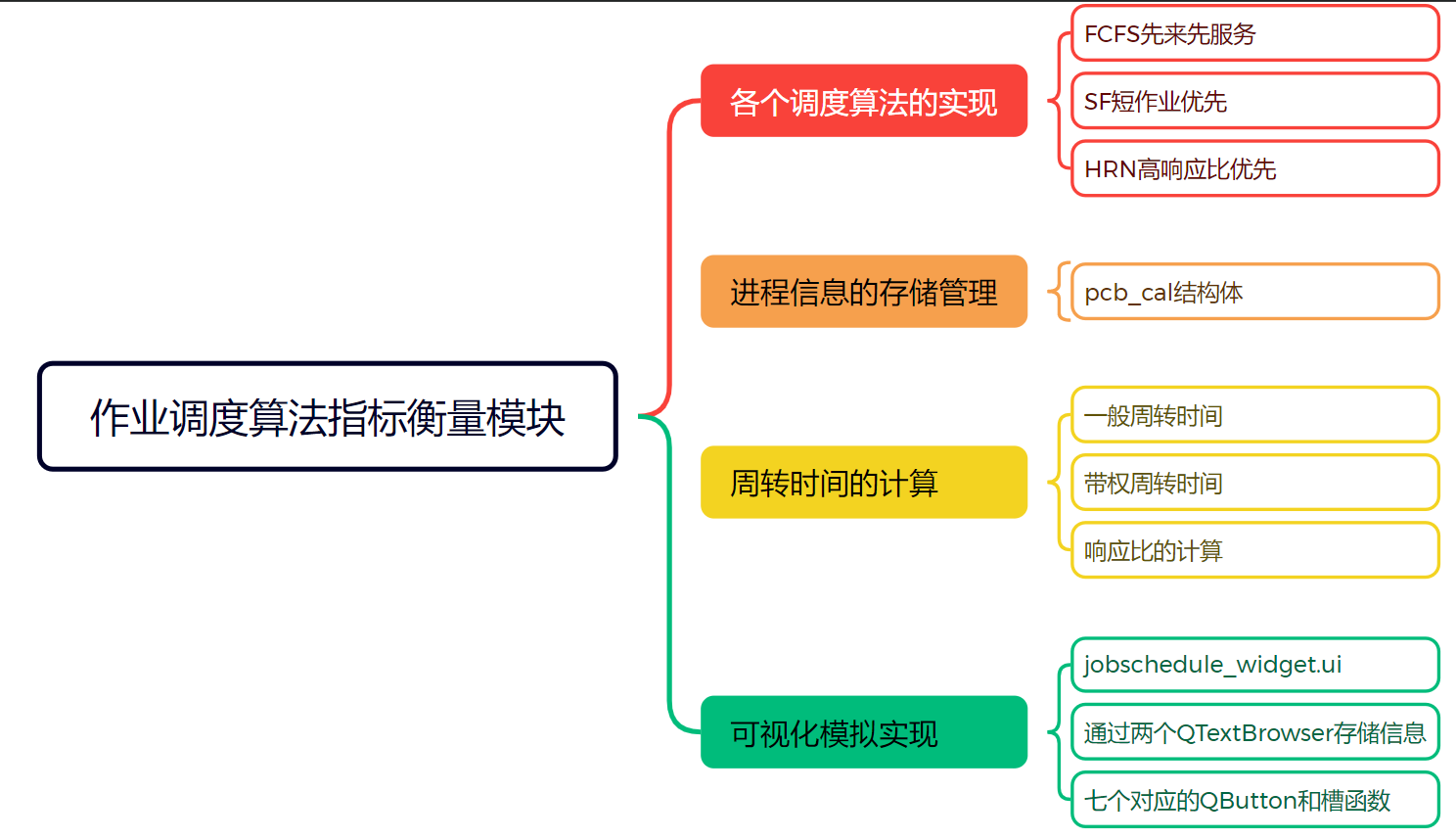
    int turnaround; //周转时间

    float weightedturnaround;       //带权周转时间

    bool flag = 0;      //标记--是否已服务

    pcb\_cal\* next;

};



在jobSchedule\_Widget类中，我们实现了FCFS先来先服务算法、SF短作业优先算法、HRN高响应比优先算法，并实现了对应的界面类，完成了相对应的信号与槽函数的绑定和业务逻辑。

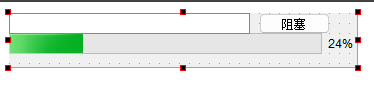
先来先服务算法（FCFS）和短作业优先算法（SF）。其中，FCFS算法按照进程到达的先后顺序进行调度，每次选择已到达且到达时间最早的进程进行执行。而SF算法则是选择服务时间最短的进程进行执行，如果有多个进程的服务时间相同，则选择最早到达的进程。在实现过程中，两种算法都使用了链表来存储进程信息，并且使用了指针来遍历链表。在FCFS算法中，每次遍历链表找到已到达且到达时间最早的进程，如果没有找到，则选择最早到达的进程。在SF算法中，每次遍历链表找到服务时间最短的进程，如果没有找到，则选择最早到达的进程。在每次选择进程后，都会更新时钟，并计算出该进程的周转时间和带权周转时间。最后，输出平均周转时间和平均带权周转时间。此外，为了让界面更加美观，还使用了背景图片 。

周转时间和带权周转时间的计算是在FCFS、SF、HRN函数中完成的。在FCFS函数中，每个进程的周转时间等于完成时间减去到达时间，带权周转时间等于周转时间除以服务时间。在SF函数中，由于是短作业优先算法，所以每次选择服务时间最短的进程，因此周转时间和带权周转时间的计算方式与FCFS相同。这些计算都是在遍历完所有进程后进行的，最后将所有进程的周转时间和带权周转时间加起来，除以进程数，得到平均值。这里需要注意的是，由于涉及到除法运算，需要将其中一个数转换为浮点数，否则结果会被截断



## 4.2、处理机调度系统可视化模拟

处理机调度系统可视化模拟模块是本次课程设计的核心模块之一。该模块通过动态可视化的方式，将处理机调度算法的过程展现在用户面前，使用户能够直观地了解算法的实现过程，分析算法的优劣，同时提高用户的实际应用能力。



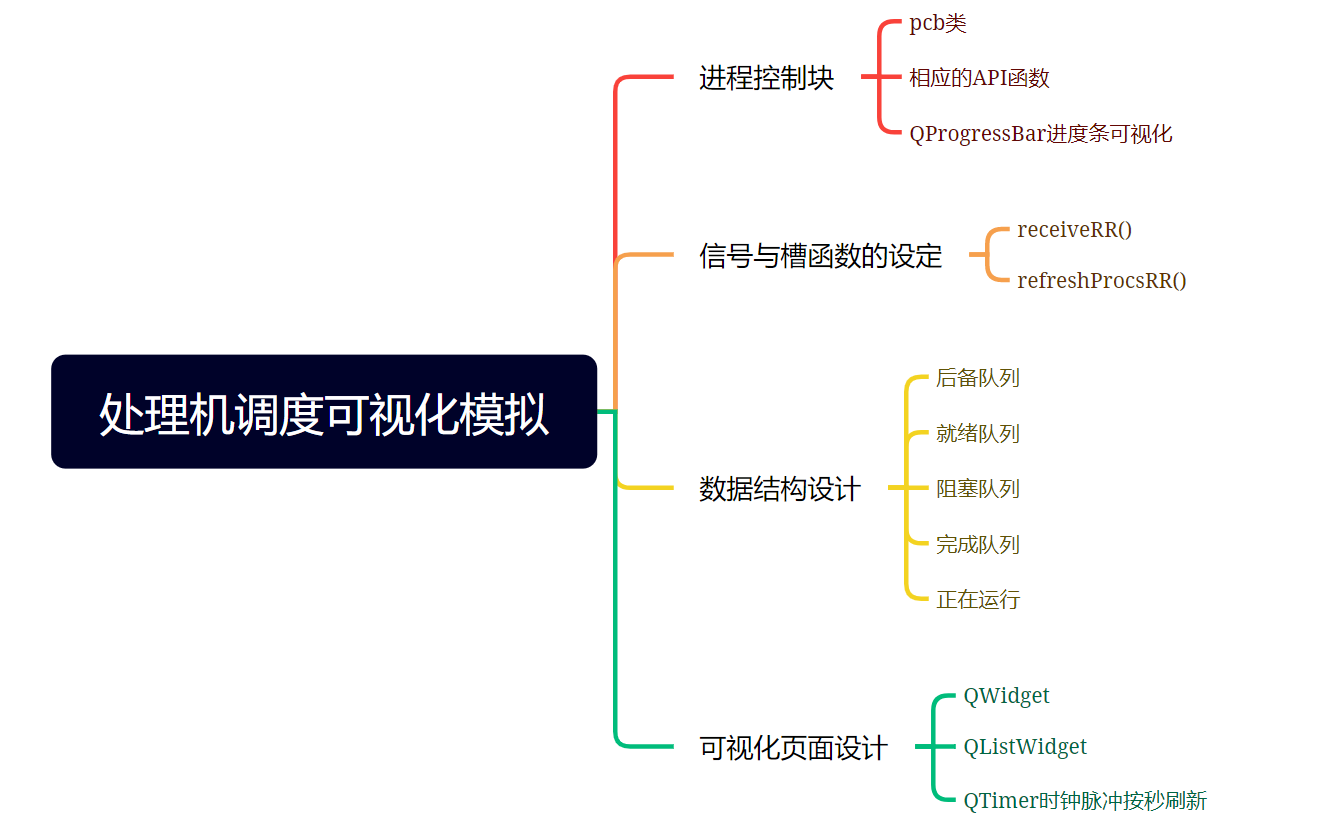
该模块包括三种不同的调度算法：轮转调度(RR)、高响应比优先调度(HPF)和最短进程优先调度(SF)。通过对这三种调度算法的模拟，用户可以深入理解每种算法的优缺点，以便于选择和优化调度算法。

在轮转调度算法中，处理机按照一定的时间片轮流运行进程。当一个进程的时间片用完后，处理机将其中断并保存其状态，然后将处理机分配给下一个就绪队列中的进程，重复以上步骤直到所有进程都执行完成。

高响应比优先调度算法是根据进程等待时间和服务时间的比值进行调度。该算法能够优先执行等待时间长的进程，以最大化响应时间。在本模块中，用户可以设置进程的等待时间和服务时间，并观察该算法对进程调度的影响。

最短进程优先调度算法则是选择执行时间最短的进程进行调度。它能够最大限度地提高系统的运行效率。在本模块中，用户可以设置进程的服务时间，并观察最短进程优先调度算法对进程的调度影响。

除了三种调度算法的模拟，本模块还提供了进程运行状态的可视化模拟。用户可以方便地了解进程的各种状态：就绪状态、运行状态和阻塞状态。同时，还可以根据需要设置进程的优先级，以模拟不同的进程优先级对调度算法的影响。



定义了ProcessSchedule类，继承自QWidget。该类中包含了一些初始化函数，如initReadyArray()、initBackupArray()和initSuspendArray()，用于初始化就绪队列、后备队列和阻塞队列。此外，还有一些删除函数，如delFirstOfBackup()、delFirstOfReady()、delOneOfReady(QString PID)和delOneOfSuspend(QString PID)，用于删除后备队列和就绪队列中的第一条数据，以及删除指定PID的就绪队列和阻塞队列中的数据。还有一些转移函数，如workToSuspend(PCB pcb)、suspendToReady(PCB pcb)和runningToReady()，用于将进程从工作状态转移到阻塞状态或就绪状态。此外，还有一些排序函数，如bubbleSort\_priority(PCB array[])和bubbleSort\_runtime(PCB array[])，用于给数组排序。最后，还有一些更新UI的函数，如readyToUi()、backupToUi()、runningToUi()和suspendToUi()，用于更新就绪队列、后备队列、运行进程和阻塞进程的UI。此外，还有一些槽函数，如receiveSF()、receiveRR()、receiveHPF()、refreshProcs\_SF()、refreshProcs\_RR()、refreshProcs\_HPF()、receiveSuspendToReady(QString)、receiveReadyToSuspend(QString)、receiveRunningToSuspend(QString)和receivePCB(QString PID, QString status, int priority, int runtime)，用于接收信号和处理事件 。

以RR时间片轮转算法为例进行讲解：

void ProcessSchedule::receiveRR()

{

    //RR时间片轮转算法

    type = 2;   //各个算法的标识

    ui->lab\_type->setText("时间片轮转调度-RR");

    chooseAlgo->close();        //关闭算法选择页面

    //向就绪进程队列中传入最多道数个进程

    for(;processNum<NUM\_OF\_PROCESS;processNum++)

    {

        //按照顺序传入进程数据

        if(backup[0].getPriority()==0)

        {

            qDebug()<<"后备队列中第0个数据为空值";

            break;

        }

        else

        {

            ready[processNum]=backup[0];

            //删除后备队列数组中的第一条数据

            delFirstOfBackup();

        }

    }

    //刷新就绪队列界面

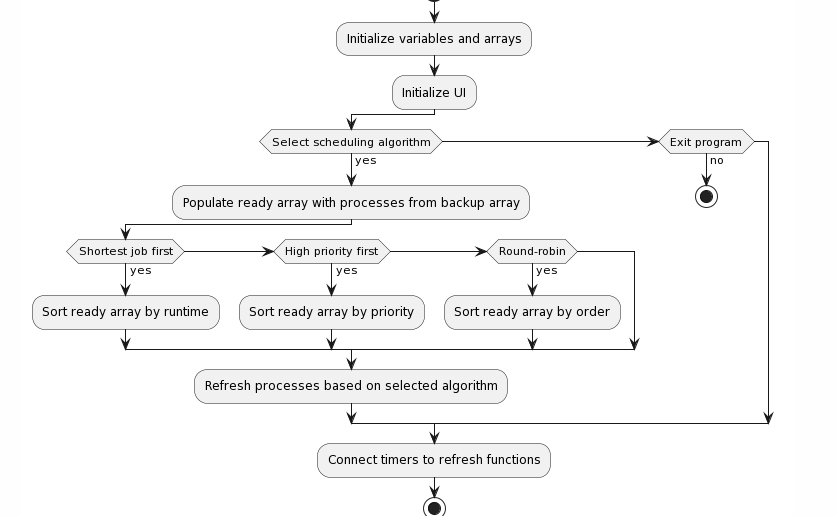
    readyToUi();

    //刷新后备队列UI界面

    backupToUi();

}

receiveRR()函数在调用时会将type设置为2，表示使用时间片轮转算法，然后通过循环向就绪进程队列中传入最多道数个进程，按照顺序传入进程数据。如果后备队列中第0个数据为空值，则跳出循环。否则，将后备队列中的第一条数据传入就绪队列数组中，并删除后备队列数组中的第一条数据。最后，调用readyToUi()和backupToUi()函数刷新就绪队列和后备队列的UI界面。



首先，如果后备队列中有进程且内存中进程数不超过最大道数，则会自动将后备队列中的进程添加到就绪队列中。然后，每隔1秒钟，运行中的进程的运行时间会减1，时间片为3。如果运行中的进程的运行时间减为0，则该进程会被标记为“终止”，并将其信息输出到进程调度信息处。接着，就绪队列中的第一个进程会被加入到运行中的进程中，就绪队列中的第一条数据会被删除，并刷新就绪列表UI界面。如果一次时间片之后，进程仍未结束，则将该进程重新加入到就绪队列的最后一条。最后，就绪队列中的第一条进程会进入运行列表，就绪队列中的第一条数据会被删除，并刷新就绪队列UI界面。

receiveRR()函数是用于时间片轮转算法的，它会将后备队列中的进程传入就绪队列中，然后调用readyToUi()和backupToUi()函数来刷新就绪队列和后备队列的UI界面。refreshProcs\_RR()函数则是用于刷新时间片轮转算法的进程，它会将当前正在运行的进程的剩余时间减去一个时间片，然后将其放回就绪队列的末尾，再将就绪队列的第一个进程取出来运行。这样就实现了时间片轮转算法的基本功能。因此，这两个函数是时间片轮转算法的核心函数，它们之间的关系是receiveRR()函数将进程传入就绪队列，而refreshProcs\_RR()函数则是在就绪队列中选择进程进行运行。

## 4.3、页面置换算法模拟

class PageReplacement : public QMainWindow

{

    Q\_OBJECT

public:

    explicit PageReplacement(QWidget \*parent = nullptr);

    ~PageReplacement();

    QString pageAl;                 // 选择的算法

    int \*task;                      // 访问串数组

    int \*page;                      // 页面数组

    char \*pgLack;                   // 缺页标记数组

    int pageSize  =0;               // 内存块数量

    int taskSize = 0;               // 访问串长度

    int pageIndx = 0;               // 页面数组索引

    int lackpage = 0;               // 缺页数

    void mwsleep(int m);            // 延迟函数

    void vilack();                  // 缺页率计算函数

    FIFO \*al\_fifo  = new FIFO();    // fifo算法对象声明

    LRU \*al\_lru = new LRU();        // lru算法对象声明

    OPT \*al\_opt = new OPT();        // opt算法对象声明

private slots:

    void openfile();                // 打开访问串文件

    void filecreat();               // 创建访问串文件

    void getslider();               // 内存块滑块改变响应

    void switchPage();              // 算法页面切换函数

    void onerun();                  // 单步模拟

    void allclear();                // 重置

    void run();                     // 模拟

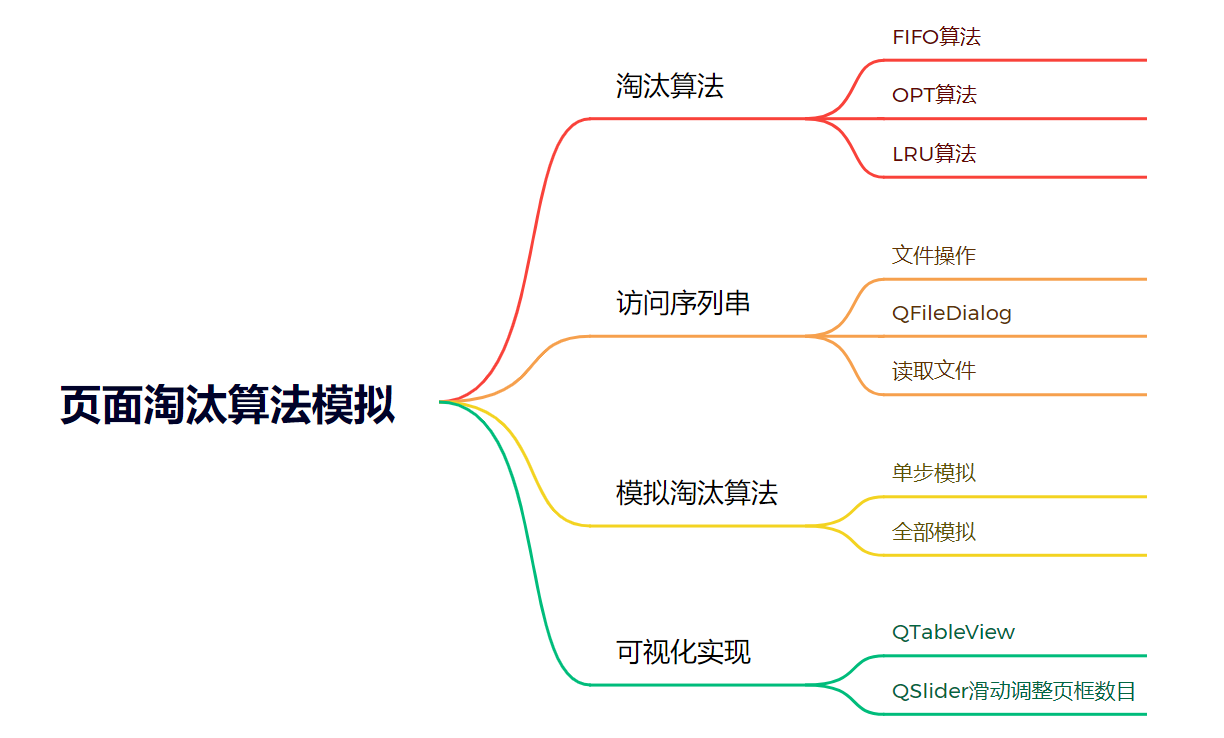
private:

    Ui::PageReplacement \*ui;

};

页面调度淘汰算法的可视化模拟模块是本次课程设计的另一个重要模块。该模块包括了文件读写、先进先出（FIFO）算法、最近最少使用（LRU）算法和最佳置换（OPT）算法的模拟。通过该模块，用户能够深入学习和理解操作系统中的页面置换算法。

设计了三种不同的算法：最优置换算法（opt）、先进先出置换算法（fifo）和最近最少使用置换算法（lru）。这些算法的设计思路都是为了在内存不足时，将最少使用的页面从内存中淘汰，以便为新的页面腾出空间。其中，最优置换算法会将未来最长时间内不会被访问的页面淘汰，先进先出置换算法会将最早进入内存的页面淘汰，最近最少使用置换算法会将最近最少被访问的页面淘汰。这些算法的实现都是基于一个访问串，通过模拟访问串中的页面访问情况，来评估算法的效果。在代码中，还包括了一些界面设计和文件操作的代码，用于方便用户输入访问串和查看算法的执行结果 。



该模块的设计思路如下：

1. 文件读写模拟：在页面调度算法中，操作系统需要从磁盘中读取文件，将文件分页装入内存，并在内存中操作文件数据。因此，本模块需要模拟文件的读写过程，将文件分页装入内存。
2. 先进先出（FIFO）算法模拟：该算法是将最先调入内存的页面置换出去。一般来说，用户调用的页面通常是存在于内存中时间最长的页面，因此该算法的代表性较高。在本模块中，我们需要根据页面进入内存的时间来模拟FIFO算法的过程。
3. 最近最少使用（LRU）算法模拟：该算法是将内存中最近最少被使用的页面置换出去，以便获得更大的可用内存。在本模块中，我们需要通过对页面的最近使用时间进行跟踪和统计，来模拟LRU算法的过程。
4. 最佳置换（OPT）算法模拟：该算法是通过预测最长时间内不再使用的页面进行选择置换。在模拟过程中，我们需要预测哪些页面在未来最长时间内不再被使用，然后将其置换出去。

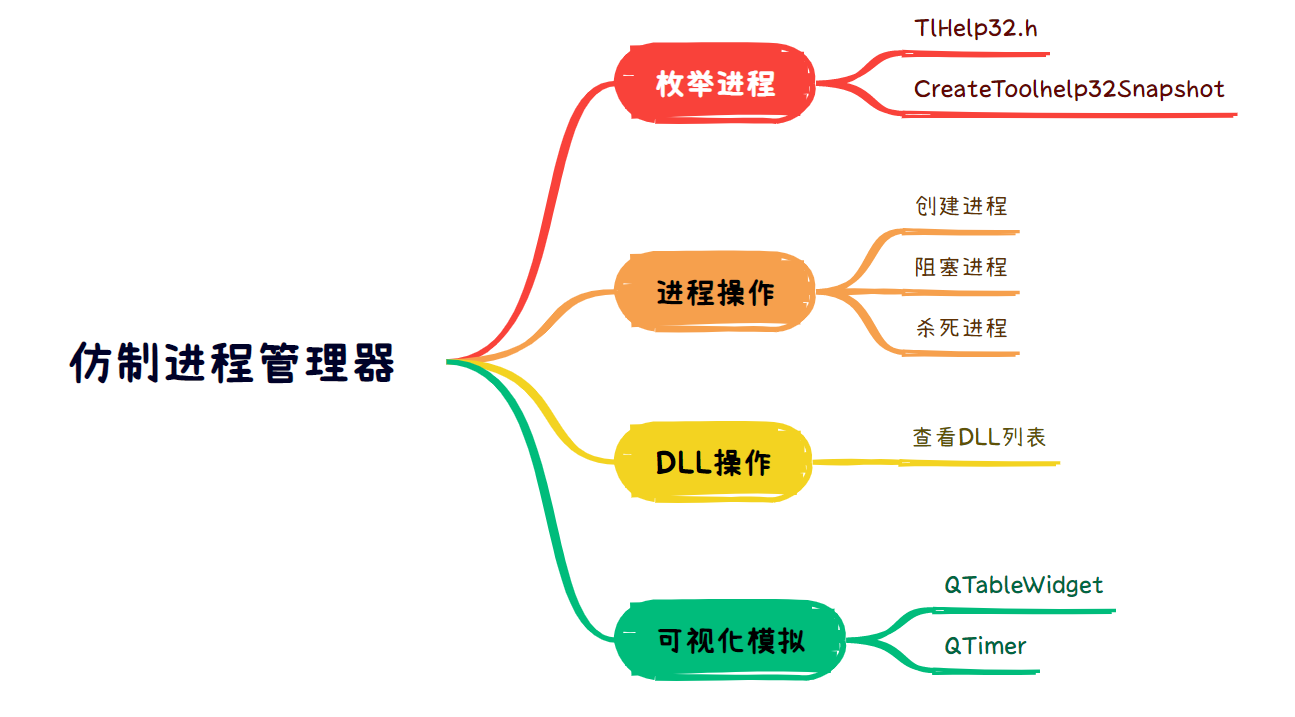


## 4.4、进程管理器模块

实现了一个进程管理器，可以列出当前系统中所有的进程，并提供创建、结束、挂起进程的功能。具体实现思路如下：  
  
1. 在构造函数中调用enumProcess()函数，该函数使用CreateToolhelp32Snapshot()函数获取系统中所有进程的快照，然后使用Process32First()和Process32Next()函数遍历快照，将进程信息插入到进程表中。  
  
2. upRole()函数用于提权，调用GetCurrentProcess()函数获取当前进程的句柄，然后使用OpenProcessToken()函数打开当前进程的访问令牌，使用LookupPrivilegeValue()函数获取调试权限的LUID，最后使用AdjustTokenPrivileges()函数调整访问令牌的权限。  
  
3. on\_btn\_CreateProcess\_clicked()函数用于创建进程，首先使用QFileDialog()函数打开文件选择对话框，选择要执行的可执行文件，然后使用CreateProcessA()函数创建进程。  
  
4. on\_btn\_FinishProcess\_clicked()函数用于结束进程，首先获取当前选中的进程的PID，然后使用OpenProcess()函数获取进程句柄，最后使用TerminateProcess()函数强制进程退出。  
  
5. on\_btn\_PauseProcess\_clicked()函数用于挂起进程，首先获取当前选中的进程的PID，然后使用CreateToolhelp32Snapshot()函数获取系统中所有线程的快照，使用Thread32First()和Thread32Next()函数遍历快照，找到属于该进程的线程，最后使用OpenThread()函数获取线程句柄，使用SuspendThread()函数挂起线程。



使用Windows API中的CreateToolhelp32Snapshot函数来枚举进程。该函数可以创建一个进程快照，包含系统中所有进程的信息。然后使用Process32First和Process32Next函数来遍历进程快照中的所有进程信息，从而实现进程枚举。在代码中，使用了PROCESSENTRY32结构体来存储进程信息，包括进程ID和进程名等 。



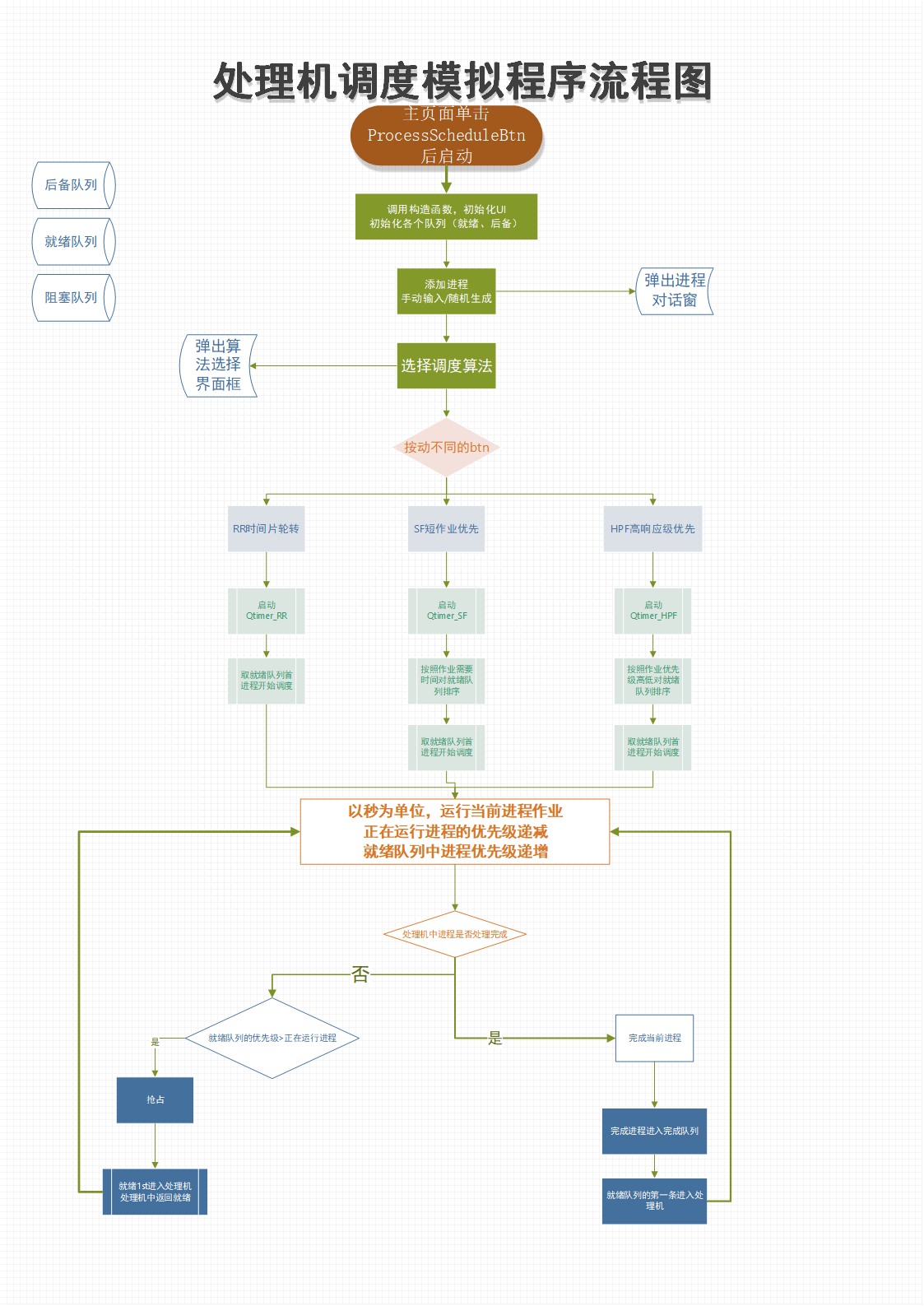
# 五、程序流程图与流程讲解

CPU调度模拟系统，主要用于模拟不同的调度算法在不同的进程情况下的表现。在程序执行过程中，首先会初始化一些变量和数组，然后根据用户选择的调度算法，将后备队列中的进程传入就绪队列中。对于短作业优先算法，会按照作业所需时间长短进行排序；对于高优先权优先算法，会按照优先权高低进行排序；对于时间片轮转算法，会按照顺序传入进程数据。然后程序会根据不同的调度算法，定时器不断地刷新进程的状态，直到所有进程都执行完毕。在程序执行过程中，还会不断地刷新就绪队列和后备队列的UI界面，以便用户可以清晰地看到进程的状态 。

在CPU调度模拟系统中，我们首先设计实现了进程控制块PCB数据结构，其中内置了如增减时间等API函数，此外，每一个进程的状态显示和完成情况由其下的Qt控件进度条实现，我们设计的调度算法都是允许剥夺和抢占的，在运行过程中，没经过一个时间片，正在运行的进程的优先级-1，剩余时间-1，就绪队列中所有进程的优先级+1，当就绪队列中有进程的优先级高于正在运行的进程时，发生抢占现象。

发生抢占时，处理机中正在运行的进程记录处理情况（相当于下断点），然后传入就绪队列中，就绪队列的第一条数据传入处理机CPU中开始进行调度。

对于时间片轮转算法，程序会定时器不断地刷新进程的状态，直到所有进程都执行完毕。具体流程如下：  
1. 首先，程序会将后备队列中的进程传入就绪队列中，然后按照顺序传入进程数据。  
2. 然后，程序会启动定时器，每隔一段时间就会刷新进程的状态。  
3. 当进程开始执行时，程序会将进程的状态设置为“运行中”，并将进程的剩余时间减去一个时间片的长度。  
4. 如果进程的剩余时间为0，则将进程的状态设置为“已完成”，并将进程从就绪队列中删除。  
5. 如果进程的剩余时间不为0，则将进程的状态设置为“就绪”，并将进程从就绪队列的末尾移到队列的开头。  
6. 如果就绪队列中没有进程，则程序会将当前正在运行的进程的状态设置为“已完成”。  
7. 当所有进程都执行完毕时，程序会停止定时器，并将所有进程的状态设置为“已完成”。  
 在程序执行过程中，还会不断地刷新就绪队列和后备队列的UI界面，以便用户可以清晰地看到进程的状态 。

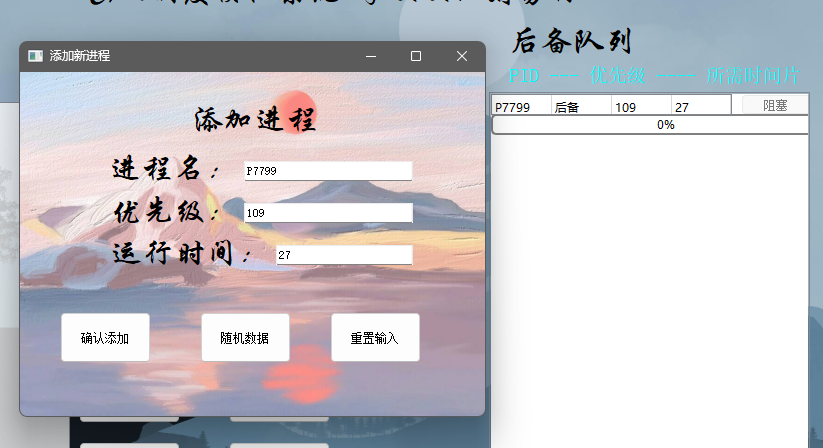


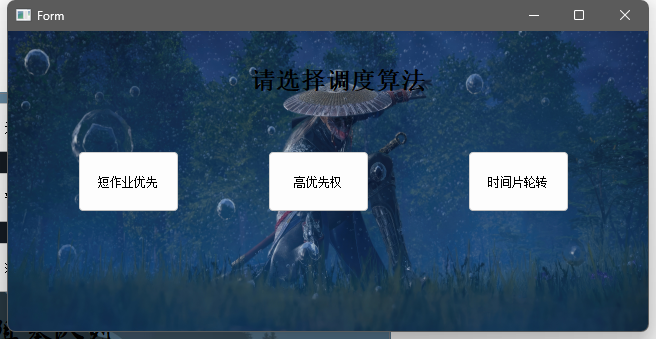
# 六、设计演示

程序预载图

程序主界面

处理机调度

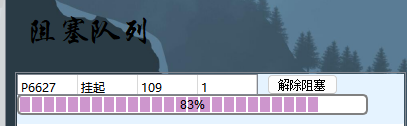
生成随机进程

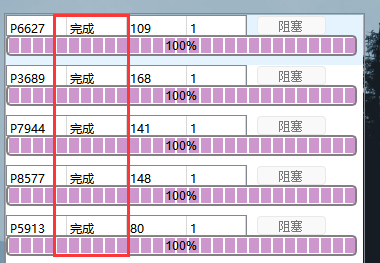
选择调度算法

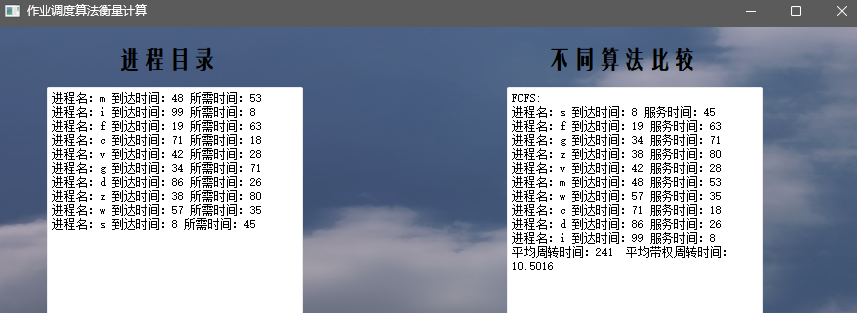
进入就绪队列，等待处理机

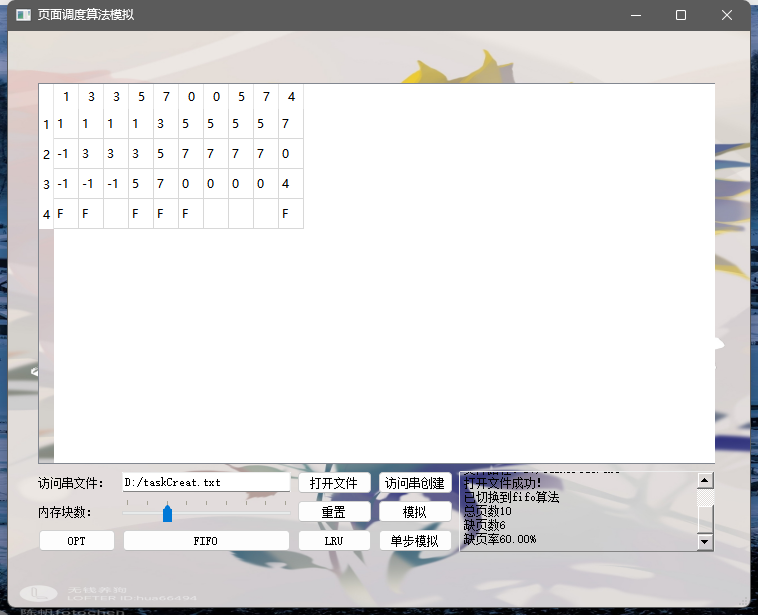
















# 七、任务分工

刘易行-2020905896-2020240401

* 需求分析
* 进程管理器模块
* 磁道调度算法模块
* 处理机调度可视化模拟模块中的SF短作业优先算法
* 相应界面UI设计与信号、槽函数的连接
* 报告撰写

李欢欢-2020904302-2020240402

* 页面调度算法可视化模拟模块
* 整体架构设计
* 作业调度算法指标衡量模块
* 处理机调度可视化模拟模块中的RR时间片轮转算法
* 相应界面UI设计与信号、槽函数的连接
* 报告撰写

# 八、心得体会：