**衡阳师范学院计算机科学与技术学院**

**课 程 设 计**

**课 程 操作系统**

**题 目 基于多级反馈队列的进程管理系统的设计**

**院 系 计算机科学与技术学院**

**专业班级 非师范七班**

**学生姓名 罗永庆**

**学生学号 22690724**

**指导教师**  **姚丽君**

**2024年 12月16日**

计算机科学与技术学院课程设计任务书

课程 操作系统

题目 基于多级反馈队列的进程管理系统的设计

专业 计算机科学与技术 姓名 罗永庆 学号 22690724

指导教师 姚丽君

**主要内容：**

本课题主要是设计一个基于多级反馈队列的进程管理系统，其主要功能包括进程的创建，删除，插入，查找与更新，当然啦~ 还包括进程在多级反馈队列中调度时的过程展示。

在使用本系统之前需要键入相应的功能键数字来启动相应的功能哦，特别记得输入完后要按回车，要按回车喵~ 我可没用getch()接收呢……

**基本要求：**

1. 设计三级队列，每一队列至少三个进程;
2. 定义PCB;
3. 为每一队列设置不同的时间片，
4. 后一队列通常是前一队列的2倍，依次类推;
5. 采用链表管理就绪队列;
6. 结果要能够显示出进程的调度过程;

**参考资料：**

**[1]盛夏,刘祥伟,龙贺.多级反馈队列调度算法及其在网络购票系统中的应用[J].软件导刊,2015,14(09):89-91.**

**[2]邱相存,臧洌,杨丹,等.基于进程执行时间的多级反馈队列调度算法[J].科学技术与工程,2015,15 (01):78-83.**

**[3]邱相存.基于改进的多级反馈队列算法的任务调度研究[D].南京航空航天大学,2015.**

目 录

[1 问题要求及任务描述 1](#_Toc16909)

[1.1 主要任务 1](#_Toc12793)

[1.2 课题要求 1](#_Toc2913)

[2 解决问题的主要思路和方法 1](#_Toc17237)

[2.1 关键问题描述 1](#_Toc20454)

[2.2 拟采用解决问题的方法 1](#_Toc18971)

[2.3 程序处理流程图 14](#_Toc1077)

[2.4 系统功能结构图 15](#_Toc616)

[3 程序实现 15](#_Toc20024)

[3.1 程序实现时应考虑的问题 15](#_Toc29628)

[3.2 部分源代码及说明 15](#_Toc5060)

[3.3添加操作流程图 18](#_Toc16289)

[4 测试 19](#_Toc7964)

[4.1 测试案例 19](#_Toc679)

[4.2 运行效果图 20](#_Toc4223)

[5.1 课题总结 22](#_Toc16955)

[5.2 尚未解决的问题及下一步工作思路 23](#_Toc19758)

[参考文献 24](#_Toc30230)

[附 录 25](#_Toc24455)

# 1 问题要求及任务描述

## 主要任务

本课题主要是设计一个基于多级反馈队列的进程管理系统，其要完成的主要功能包括进程的创建，删除，插入，查找与更新，当然啦~ 最重要的功能就是进程在多级反馈队列中进行调度时的详细过程展示啦~，本人是使用C++配合C来完成此系统哒，刚开始本来是打算用JAVA来实现的,写到一半发现不对劲……

在使用本系统之前需要键入相应的功能键数字来启动相应的功能哦，特别记得输入完后要按回车，要按回车喵~ 我可没用getch()接收呢……

演示过程分为两类。一种我取名叫开门见山~，直接把进程调度中的全部过程显示出来;另一种叫兵来将挡! 在此模式下不同的时间点会输出不同的信息，程序会自动暂停，方便给他人一步一步的讲解这个进程调度的全部过程喵~

## 课题要求

1. 设计三级队列，每一队列至少三个进程;
2. 定义PCB;
3. 为每一队列设置不同的时间片，
4. 后一队列通常是前一队列的2倍，依次类推;
5. 采用链表管理就绪队列;
6. 结果要能够显示出进程的调度过程;

# 2 解决问题的主要思路和方法

## 2.1 关键问题描述

(1) PCB该如何定义,哪些属性是必要的？

(2) 如何实现就绪队列的链式存储结构？

(3) 如何实现MFQ算法？

(4) 如何实现用户进程能根据时间点依次进入到相应队列当中？

(5) 如何保证程序运行的可靠性，并让其输出结果显示的内容更加清晰易懂？

(6) 由于每个进程的pid都是唯一的，如何保证用户输入的pid不会重复呢？  
 (7) 演示完后进程的状态会发生改变，第二次想要再次重新演示的时候就不是原来的结果了，应该如何解决呢？

## 2.2 拟采用解决问题的方法

(1) PCB该如何定义,哪些属性是必要的？

因为只是简单的演示一下MFQ算法调度的过程,

因此只需要定义那些要用到的属性就可以啦：

int pid; // 进程id

int reachTime; // 进程到达时间

int execTime; // 进程执行时间

int remainingTime; // 剩余执行时间

int queuePriority; // 进程所在队列

struct ProcessControlBlock\* nextPCB; // 指向下一个PCB

(2) 如何实现就绪队列的链式存储结构？  
 首先在PCB的结构体定义中添加一个指针域，用于指向下一个PCB  
 代码实现部分：struct ProcessControlBlock\* nextPCB;

然后我就只需要在就绪队列的定义中添加两个指针head与tail，

分别指向链头和链尾

有元素要进来了我就让链尾指针tail往后移 即入队

有元素要出去了我就让链头指针head往后移 即出队。

这样就实现了就绪队列的链式存储结构啦~

代码实现部分：

PCB\* head; // 队头指针

PCB\* tail; // 队尾指针

// 入队

void enQueue(PCB\* pcb)

{

// 刚进来的第一个PCB做为首节点

if (head == NULL)

head = tail = pcb;

else

{

tail->nextPCB = pcb;

tail = pcb;

tail->nextPCB = NULL;

}

qSize++;

}

// 出队

PCB\* deQueue()

{

// 空队列 返回NULL

if (head == NULL)

return NULL;

else

{

// 出队操作

PCB\* temp = head;

head = head->nextPCB;

qSize--;

return temp;

}

}

1. 如何实现MFQ算法?  
    根据宝藏UP主分享的MFQ算法思路 大致总结如下：

其实相当于算法题中的模拟题，直接模拟满足下面要求即可。

1. 所有进程根据到达时间依次在相应时间点入队

(最初都会进入优先级最高的第一级队列当中)  
2.每个队列都会按照FCFS的原则(其实就是FIFO)给进程分配CPU时间片(我这里分配的是1 2 4)进行处理，此时若进程执行完了(执行的剩余时间为0) 则将该进程出队(但进程此时还是在内存的并没有回收)，此时若分配给该进程的时间片用完了都还没处理完 继续往优先级低的队列走(1-> 2-> 3),如果走到末尾队列(第三级队列)都还没处理完，则继续回到第三级队列末尾继续等待执行，一直循环直到执行结束为止。

需要注意的两个点：  
 [1] 当前进程的执行条件是“优先级比比它所在队列还高的队列中没有新进程进来为前提才能执行”通俗的讲就是小弟进程要等老大进程处理完了之后才能开始处理。

[2] 该进程在执行到某个时间点时若优先级高的队列中有新进程进来会停止执行，并重新回到当前队列的末尾等待执行，这个即抢占式调度。通俗的讲就是小弟进程被老大进程抢了都会溜回到原队列末尾继续排队等待。

代码实现

(有点长 中途码代码的时候修复了许多Bug 具体实现细节请看源代码喵~)



1. 如何实现用户进程能根据时间点依次进入到相应队列当中？

因为用户输入的进程到达时间可能不一样，存储到数组中是乱序的,

遍历起来不太方便，此时就需要进行升序啦，

而为了方便我使用了C++库函数中的sort函数来进行升序。

代码实现部分：

//升序

sort(pcb, pcb + pCnt, [&](const PCB x, const PCB y) {

if (x.reachTime == y.reachTime)

return x.execTime < y.execTime;

return x.reachTime < y.reachTime;

});

通过升序之后我就可以通过变量currentTime从0计时，每执行一秒都遍历一遍，去跟用户进程的pid进行比较，如果相等我就让这个进程入队，同时让低优先级队列中正在执行的进程停止进入队列队尾并打印出相关提示信息就可以啦~

代码实现的关键部分:

int currentTime = 0; //当前时间

//判断 1 2 第二级队列与第三级队列

if (i)

{

//第二级队列与第三级队列中的进程每秒都得判断一下是否有新的进程进来

FORB(time, 1, TimeSlice[i])

{

bool sign = false;

bool ifmove = mfq[0]->isEmpty();//用于判断刚开始1级队列为空 下一秒就突然来了>=1个进程的情况

int move = 0;

//判断当前时间点是否有新进程进来

while (pcb[index].reachTime == currentTime)

{

sign = true;

move++;

mfq[0]->enQueue(&pcb[index++]);

printf("当前时间（%d）： 警告！警告！ 进程【%d】降临1级队列！！ \n", currentTime, pcb[index - 1].pid);

}

if (sign)//如果第1级队列有进程 则不用输出 因为根本就轮不到它执行。

{

if (ifmove&&move>=1) //如果队列1只有一个进程 则输出终止信息

{

PCB\* cur = mfq[currentProcess->queuePriority]->deQueue();

if (cur == NULL) //当前进程是队列中的最后一个进程 取出来后刚好变空了。

{

printf("当前时间（%d）： 进程【%d】识趣的停止执行 溜回到当前%d级队列末尾继续等待喵~\n", currentTime, currentProcess->pid, currentProcess->queuePriority + 1);

mfq[currentProcess->queuePriority]->enQueue(currentProcess);

}

else {

mfq[currentProcess->queuePriority]->enQueue(cur);

printf("当前时间（%d）： 进程【%d】识趣的停止执行 溜回到当前%d级队列末尾继续等待喵~\n", currentTime, cur->pid, cur->queuePriority + 1);

}

}

i = 0; //重新来到1级队列执行

goto cat;

}

currentProcess->remainingTime--;

currentTime++;

if (view)

{

printf("\n");

system("pause");

system("cls");

}

printf("\n当前时间（%d）： 进程【%d】已执行%d秒 剩余执行时间%d秒\n", currentTime, currentProcess->pid, 1, currentProcess->remainingTime);

}//for

}//第二层的if

else

{

//判断当前时间点是否有新进程进来

while (pcb[index].reachTime == currentTime)

{

mfq[0]->enQueue(&pcb[index++]);

printf("当前时间（%d）： 警告！警告！ 进程【%d】降临1级队列！！ \n", currentTime, pcb[index - 1].pid);

}

// 使用时间第一级队列片

currentProcess->remainingTime -= TimeSlice[i];

currentTime += TimeSlice[i];

if (view)

{

printf("\n");

system("pause");

system("cls");

}

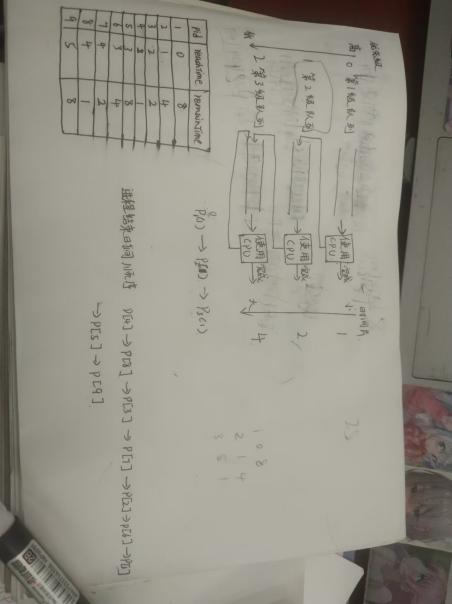
printf("\n当前时间（%d）： 进程【%d】已执行%d秒 剩余执行时间%d秒\n", currentTime, currentProcess->pid, TimeSlice[i], currentProcess->remainingTime);

}

(5) 如何保证程序运行的可靠性，并让其输出结果显示的内容更加清晰易懂？

无法100%保证，但也有90%多啦~，可以通过验证大量的测试数据来提高程序执行的可靠性喵~，为了方便测试本人更是使用了最通俗易懂的语言让内容更加清晰易懂喵~ 如图所示：



还设计了两种不同的输出模式来辅助验证，最后再通过在草稿纸上推算一步一步的验证着程序执行的可靠性。如图所示。

可以确定的是源程序中给出的三组测试数据都是通过的，不过通过这些测试数据后也确实发现了一些逻辑Bug，并给出了相应的解决方法。

部分Bug的解决代码如下图：







(6)由于每个进程的pid都是唯一的，如何保证用户输入的pid不会重复呢？

这个我只需要添加一个布尔数组就行啦，刚开始数组的值默认都是false,如果用户输入了pid后就让这个pid对应的布尔数组为true，后面用户再次输入的时候我就可以通过这个布尔数组来判断用户输入的pid是否为true就可以啦 ，如果为true就提示用户这个pid已经输入过了，让用户重新输入。

代码实现部分：

bool pidr[N]; //解决进程pid不可重复问题

int n;

scanf\_s("%d", &n);

int index = pcb[0].pid ? pCnt : 0; //防止覆盖

pCnt += n;

FORB(i, index, pCnt - 1)

{

system("cls");

printf("\a");

printf("《 温馨小提示 》\n\n");

printf("【1】输入格式：进程id 到达时间 执行时间\n\n");

printf("【2】合法进程id范围(1~1000范围内)\n\n ");

printf("请输入：");

scanf\_s("%d %d %d", &pcb[i].pid, &pcb[i].reachTime, &pcb[i].execTime);

if (pcb[i].pid < 1 || pcb[i].pid>1000)

{

system("cls");

printf("\a");

printf("输入的pid数值范围有误(不在1~1000范围内)，请重新输入~");

system("pause");

i--;

continue;

}

if (pidr[pcb[i].pid])

{

system("cls");

printf("\a");

printf("系统中已有此pid啦~，请重新输入喵~\n\n");

system("pause");

i--;

continue;

}

else pidr[pcb[i].pid] = true;

pcb[i].remainingTime = pcb[i].execTime;

pcb[i].queuePriority = 0; //刚开始都会进入到优先级最高的队列(0)当中

pcbf[i] = pcb[i];

}

(7) 演示完后进程的状态会发生改变，第二次想要再次演示的时候就不是原来的结果了，应该如何解决呢？

解决方案很简单，就是再建立一个进程的副本数组，每次演示完之后让变化后的PCB数组再重新加载副本数组中原来的状态值即可解决啦~

代码实现的关键部分:

PCB pcb[N];

PCB pcbf[N];//副本

FORB(i, index, pCnt - 1)

{

system("cls");

printf("\a");

printf("《 温馨小提示 》\n\n");

printf("【1】输入格式：进程id 到达时间 执行时间\n\n");

printf("【2】合法进程id范围(1~1000范围内)\n\n ");

printf("请输入：");

scanf\_s("%d %d %d", &pcb[i].pid, &pcb[i].reachTime, &pcb[i].execTime);

if (pcb[i].pid < 1 || pcb[i].pid>1000)

{

system("cls");

printf("\a");

printf("输入的pid数值范围有误(不在1~1000范围内)，请重新输入~");

system("pause");

i--;

continue;

}

if (pidr[pcb[i].pid])

{

system("cls");

printf("\a");

printf("系统中已有此pid啦~，请重新输入喵~\n\n");

system("pause");

i--;

continue;

}

else pidr[pcb[i].pid] = true;

pcb[i].remainingTime = pcb[i].execTime;

pcb[i].queuePriority = 0; //刚开始都会进入到优先级最高的队列(0)当中

pcbf[i] = pcb[i];

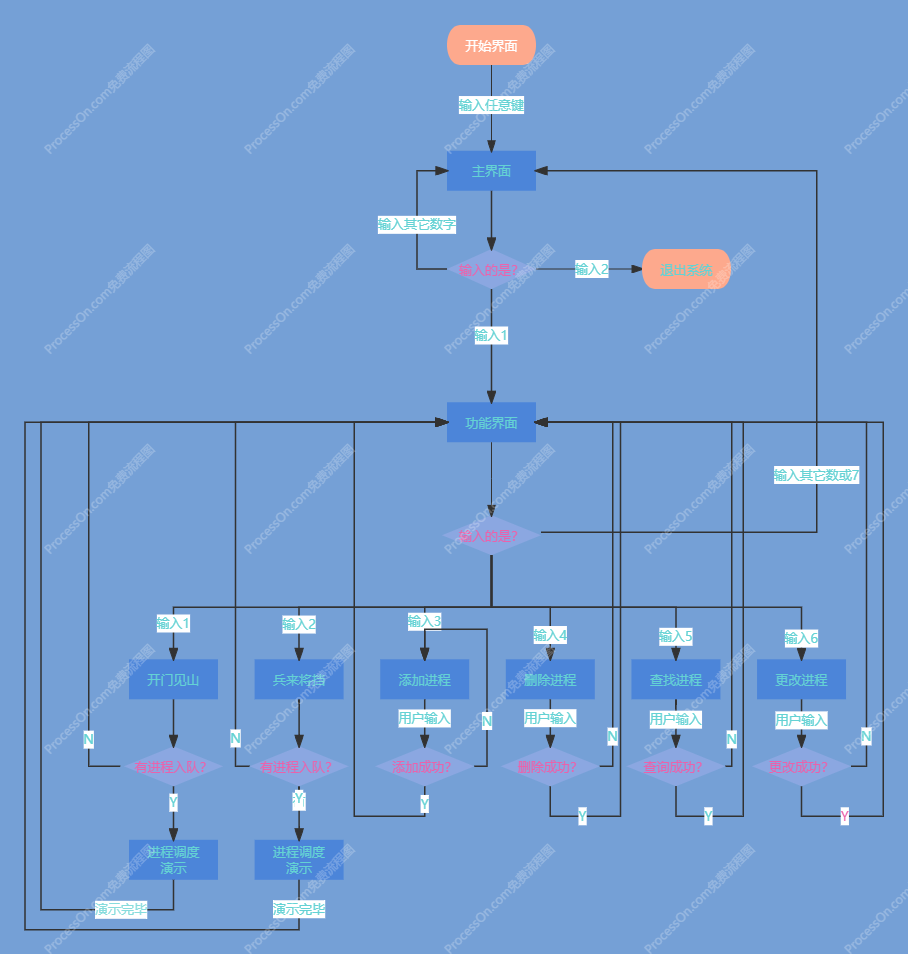
}

//进程演示后进行状态恢复

FORB(i, 0, pCnt - 1)

pcb[i] = pcbf[i];

## 2.3 程序处理流程图



## 2.4 系统功能结构图

基于多级反馈队列的   
进程管理系统

启动系统！

退出系统~

开  
门  
见  
山

兵  
来  
将  
挡

删除进程

添  
加  
进  
程

查找进程

更  
改  
进  
程

返回首页

# 

# 3 程序实现

## 3.1 程序实现时应考虑的问题

在码代码的时候我们需要考虑以下几个问题，第一：如何设计进程管理系统？其最核心的功能是什么？哪些功能是必不可少的？第二：如何实现这些功能并保证程序运行后的可靠性？第三：每个进程的pid都是唯一的，有哪些这样的细节在码代码的时候是需要尤其需要注意的？第四：当程序出现Bug时应该如何快速定位并解决。第五：在实现这个功能后里面某个全局变量的改变会不会影响到其它功能模块的判断？

## 3.2 部分源代码及说明

当用户输入相应的功能键数字后会执行相应的功能，注意两种进程展示功能只有等到用户创建了进程之后才可以显示，在用户创建了进程之后就可以使用啦~ 期间还可以再新添一些新的进程进来，或者删除某个或全部进程，查找某个或全部进程，更新某个进程等操作喵~并提供了返回首页与退出系统等功能方便用户的使用。

主要代码实现如图所示：

添加与插入 删除 查找

更新的代码最少，我就不客气的拷进文档啦~

//进程更新功能

void update()

{

system("cls");

int pid;

printf("请输入您想要更新的进程pid： ");

scanf\_s("%d", &pid);

printf("\a");

FORB(i, 0, pCnt - 1)

{

if (pcb[i].pid == pid)

{

printf("请更改进程属性吧【到达时间 执行时间】\n\n");

scanf\_s("%d %d",&pcb[i].reachTime,&pcb[i].execTime);

system("cls");

printf("\a");

printf("更新成功! 请稍等~");

Sleep(2000);

pcbf[i] = pcb[i];

return;

}

}

system("cls");

printf("\a");

printf("更新失败! 系统好像找不到此进程的pid呢~");

Sleep(2000);

}

## 3.3添加操作流程图

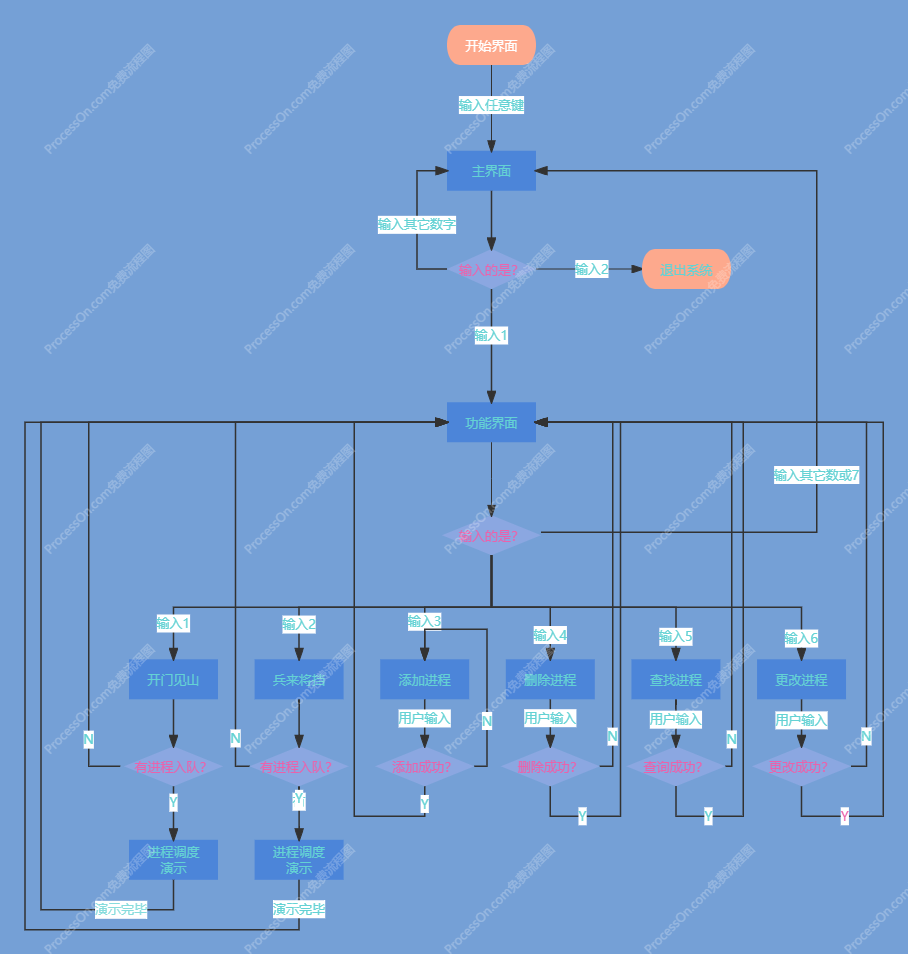


图 3.1添加操作流程图

# 4 测试

## 4.1 测试案例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 输入 | 实际输出 | 期望输出 | 是否通过 |
| 1 | 9  1 0 8  2 1 4  3 2 2  4 2 1  5 3 8  6 3 4  7 4 2  8 4 1  9 5 8 | 输出进程的  调度过程 | 与自己预想的  输出一致 | 是 |
| 2 | 9  1 7 8  2 2 4  3 2 2  4 5 1  5 6 8  6 4 4  7 2 2  8 4 1  9 3 8 | 输出进程的  调度过程 | 与自己预想的  输出一致 | 是 |
| 3 | 3  1 0 8  2 1 4  3 5 1 | 输出进程的  调度过程 | 与自己预想的  输出一致 | 是 |

## 4.2 运行效果图



图4.1 主界面~



图4.2 功能界面~



图4.3 开门见山~



图4.4 兵来将挡~

其它图就不展示啦喵~ 需要阁下自行探索喵~

## 5.1 课题总结

通过本次课题的设计与学习，本人受益匪浅，对代码大道的理解又加深了一个层次，对代码的运用与找Bug的能力又提升了一大截，通过码完进程管理系统的代码后，我对MFQ算法的进程调度过程有了一个更深层次的理解，同时深深的知道了测试用例的重要性，难道打算竞时后台一堆的测试用例。看着四天后就要计组期末考试了还是复习当预习……祝我好运喵~

优点：

达到了课题的基本要求，并在此基础上又拓展了一些其它功能，比如退出系统，返回主界面，进程的添加，插入，删除，更新与查找等功能。

提升了自己码代码的能力，实践能力又提升了一大截。

提升了自己找Bug的能力，调试操作熟练的跟吃饭睡觉一样。

提升了自己的精神境界，在郁闷与舒畅之间来回跳动~

不足：

由于时间原因未进行大量测试数据的验证  
程序可能还藏着潜在的Bug未被发现。

## 5.2 尚未解决的问题及下一步工作思路

放眼全码，其实本系统所实现的代码执行效率还不是最优的，还有部分可以优化的地方需要去改进，还有就是一开始并不是所有情况都会被考虑到，可能后面随着测试数据的测试会遇到些新的Bug等着我去解决，因此为了提高我以后码出的程序的可靠性与健状性，我需要多多去参加一些算竞比赛来不断地去提升我的思维水平喵~。

# 参考文献

[1]盛夏,刘祥伟,龙贺.多级反馈队列调度算法及其在网络购票系统中的应用[J].软件导刊,2015,14(09):89-91.

[2]邱相存,臧洌,杨丹,等.基于进程执行时间的多级反馈队列调度算法[J].科学技术与工程,2015,15 (01):78-83.

[3]邱相存.基于改进的多级反馈队列算法的任务调度研究[D].南京航空航天大学,2015.

# 附 录

程序实现代码：

#define \_CRT\_SECURE\_NO\_WARNINGS //忽略关于不安全函数的警告

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <algorithm>

#include <time.h>

using namespace std;

const int N = 1001; //假设本喵只接收1000 ,pid只允许设置1~1001

int pCnt;

bool view = false;

bool pidr[N]; //解决进程pid不可重复问题

………………………………………………