中南大学

操 作 系 统

学生姓名 马福龙

学　　号 0902150310

专业班级 计科 1504

指导教师 宋虹

学　　院 信息科学与工程学院

完成时间 2017 年 6 月

目录

实验一　处理机调度

实验目的

多道系统中，当就绪进程数大于处理机数时，须按照某种策略决定哪些进程优先占用处理机，本实验模拟实现处理机调度，以加深了解处理机调度的工作。

实验内容

选择一个调度算法，实现处理机调度

（1）　设计一个按优先权调度算法实现处理机调度的程序

（2）　设计以时间片轮转实现处理机调度的程序

实验要求

（1）　最好采用图形界面；

（2）　可随时增加进程；

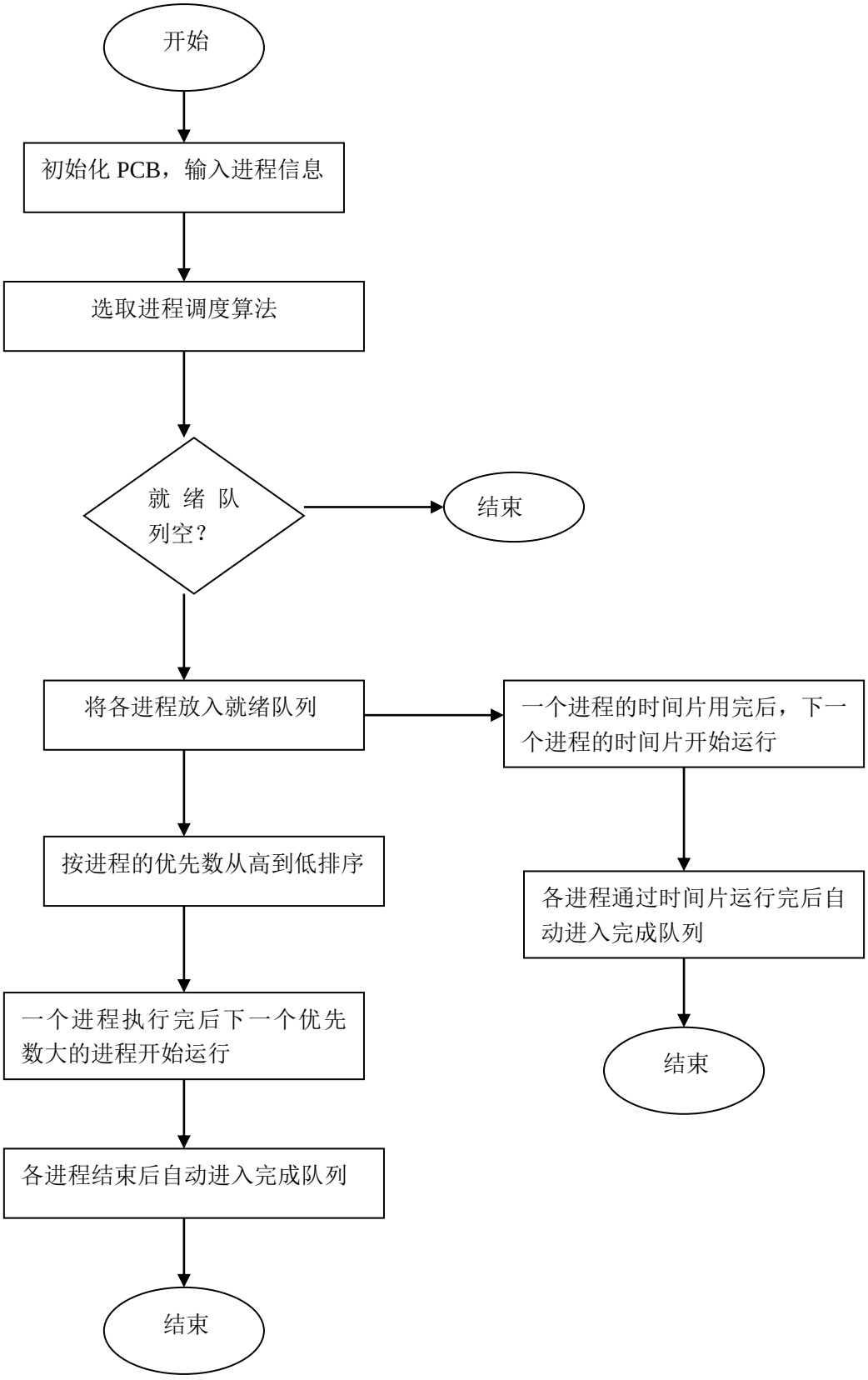
（3）　规定道数，设置后备队列和挂起状态。若内存中进程少于规定道数，可自动从后备队列调度一作业进入。被挂起进程入挂起队列，设置解挂功能用于将指定挂起进程解挂入就绪队列；

（4）　每次调度后，显示各进程状态。

实验主要思想

系统为每个进程分配的时间片为 1s，由用户手动输入进程名、优先级、运行时间等信息，由系统分配进程的 ID 号。开始调度时，选择优先级最高的进程作为当前进程，如果进程在 1s 内完成，则放入完成队列；如果没运行完，则其优先级减一重新放入就绪队列。如果用户想要挂起某个进程，可以双击就绪队列中的某一行，将该进程放进挂起队列；若想解除挂起，可以双击挂起队列中的某一行，将该进程重新放入就绪队列。

流程图设计


开始
初始化 PCB，输入进程信息
选取进程调度算法
就 绪 队 列空？
结束
将各进程放入就绪队列
一个进程的时间片用完后，下一个进程的时间片开始运行
按进程的优先数从高到低排序
各进程通过时间片运行完后自动进入完成队列
一个进程执行完后下一个优先数大的进程开始运行
结束
各进程结束后自动进入完成队列
结束

实验主要函数详解

进程控制块

Mem.h 包含了进程标识符，进程控制信息，进程调度信息等信息，代码块如下：

```
struct PCB
{
    PCB(){

    }

    PCB(int pid1,int priority1,int time1,int memoryNeeds1){
        pid=pid1;
        priority=priority1;
        time=time1;
        memoryNeeds=memoryNeeds1;
    }

    int pid;
    int state;
    int priority;
    int time;       //要求运行时间
    int memoryNeeds;//the needs memory
    int memoryStart;//the memory start place
    operator <(PCB& a){
        return priority<a.priority;
    }
};
```

进程调度

```
//CPU 调度
void MainWindow::pcbsche(){
    if(reserve.size()==0&&ready.size()==0){
        t1->stop();
        t2->stop();
        ui->progressBar->setValue(0);
        ui->tableWidgetRun->clear();
        QMessageBox::about(NULL, "About", "Please add a PCB");

    }
    else{
        for(vector<PCB>::iterator it=reserve.begin();it!=reserve.end();)
        {
```

```
                //当道数小于 6 时,从后备队列取出 PCB 添加到就绪队列，直到道数为 6 或者后备队列为空
                if(ready.size()+hang.size()>=6)
                    break;

                if(memo.add(*it)){//if add sucessfully 可以容纳
                    ui->textBrowserResult->append(QString::number((*it).pid)+"  已进入就绪队列!");
                    ready.insert(ready.begin(),*it);
                    it=reserve.erase(it);
                    ui->progressBarMemory->setValue(memo.sum);
                }
                else{
                    ui->textBrowserResult->append(QString::number((*it).pid)+"  所需内存过大，进入就绪队列失败!");
                    it++;
                }


            }
            if(ready.size()==0){
                t1->stop();
                t2->stop();
                ui->progressBar->setValue(0);
                ui->tableWidgetRun->clear();
                QMessageBox::about(NULL, "About", "就绪队列无进程！ ");
                return;
            }
            updateReserve();
            if(scheWay==0){
                sort(ready.begin(),ready.end());
            }
            run=ready.back();
            ready.pop_back();
            ui->textBrowserResult->append(QString::number(run.pid)+"  正在运行");
            ui->tableWidgetRun->clear();
            ui->tableWidgetRun->setItem(0,0,new QTableWidgetItem(QString::number(run.pid)));
            ui->tableWidgetRun->setItem(0,1,new QTableWidgetItem(QString::number(run.priority)));
            ui->tableWidgetRun->setItem(0,2,new QTableWidgetItem(QString::number(run.time)));
            ui->tableWidgetRun->setItem(0,3,new QTableWidgetItem(QString::number(run.memoryNeeds)));
```

```
            updateReady();
            run.time--;
            if(scheWay==0)
                run.priority--;
            if(run.time==0){
                 //进程已经完成
                int memoryStart=run.memoryStart;
                memo.remove(memoryStart);
                ui->progressBarMemory->setValue(memo.sum);
                ui->textBrowserResult->append(QString::number(run.pid)+" 运行完成");
            }else{//如果进程尚未完成，重新添加到就绪队列
                ready.insert(ready.begin(),run);
            }


        }

}
//响应开始按钮
void MainWindow::on_pushButtonSche_clicked()
{
    t1=new QTimer;
    connect(t1,SIGNAL(timeout()),this,SLOT(pcbsche()));
    QString time=ui->lineEditShiJianPian->text();
    t1->start(time.toInt()*1000);//时间片为 1s
    t2=new QTimer;
    connect(t2,SIGNAL(timeout()),this,SLOT(UpdateProcessBar()));
    t2->start(time.toInt()*100);
    pcbsche();
}
//单击获取选中的 pid
int MainWindow::getPid(const QModelIndex &index){
    int row=index.row();
    int pid=index.sibling(row,0).data().toString().toInt();
    selectPid=pid;
    return pid;
}
//单击就绪控件
void MainWindow::on_tableWidget_Ready_clicked(const QModelIndex &index)
{

}
//单击挂起控件
void MainWindow::on_tableWidget_Hang_clicked(const QModelIndex &index)
```

```
{

}
//响应挂起操作
void MainWindow::on_pushButtonHang_clicked()
{
    for(vector<PCB>::iterator it=ready.begin();it!=ready.end();it++){
        if((*it).pid==selectPid){
            hang.push_back(*it);
            memo.remove((*it).memoryStart);
            ready.erase(it);
            ui->progressBarMemory->setValue(memo.sum);
            ui->textBrowserResult->append(QString::number(selectPid)+"    已挂起");
            updateReady();
            updateHang();
            break;
        }
    }
}
//响应解挂操作
void MainWindow::on_pushButtonWakeUp_clicked()
{
    for(vector<PCB>::iterator it=hang.begin();it!=hang.end();it++){
        if((*it).pid==selectPid){
            if(memo.add(*it)){
                ready.insert(ready.begin(),*it);
                hang.erase(it);
                ui->textBrowserResult->append(QString::number(selectPid)+"    已解挂");
                ui->progressBarMemory->setValue(memo.sum);
                updateReady();
                updateHang();
            }
            else
                ui->textBrowserResult->append(QString::number(selectPid)+"    内存不足以
解挂");
            break;
        }
    }
}
//响应暂停按钮，停止时钟
void MainWindow::on_pushButtonSchePause_clicked()//暂停
{
    if(t1->isActive()||t2->isActive()){
```

```
            t1->stop();
            t2->stop();
            ui->tableWidgetRun->clear();
            updateReady();
        }
}
//生成随机进程
void MainWindow::on_pushButtonSrand_clicked()
{
    srand((unsigned)time(NULL));
    for(int i=0;i<20;i++){
        PCB a(i,random(1,40),random(1,30),random(1,30));
        ui->textBrowserResult->append(QString::number(i)+"    已进入就绪队列!");
        reserve.push_back(a);
    }
    updateReserve();
    }
//建立 PCB
void MainWindow::on_pushButtonNewPcb_clicked()
{

    QString name=ui->lineEditName->text();
    QString priority=ui->lineEditPriority->text();
    QString time=ui->lineEditTime->text();
    QString mem=ui->lineEditMem->text();
    ui->lineEditName->clear();
    ui->lineEditPriority->clear();
    ui->lineEditTime->clear();
    ui->lineEditMem->clear();
    if(ui->comboBox->isEnabled()){
        scheWay=ui->comboBox->currentIndex();
        ui->comboBox->hide();
    }
    reserve.push_back(PCB(name.toInt(),priority.toInt(),time.toInt(),mem.toInt()));
    updateReserve();

    }
```

实验结果

因和实验二一起完成，故将实验结果与实验二实验结果合二为一，详见实验二实验结果。

实验感想

通过这次实验，我了解了各种处理机调度策略，尤其是时间片轮转法和动态优先级法。也感谢老师的严格要求，让我有机会去了解 Qt 中信号与槽的机制。从中学到了很多 C++图形界面开发的知识。

实验二　主存空间的分配和回收

实验目的

帮助了解在不同的存储管理方式下，应怎样实现主存空间的分配和回收

实验内容

主存储器空间的分配和回收

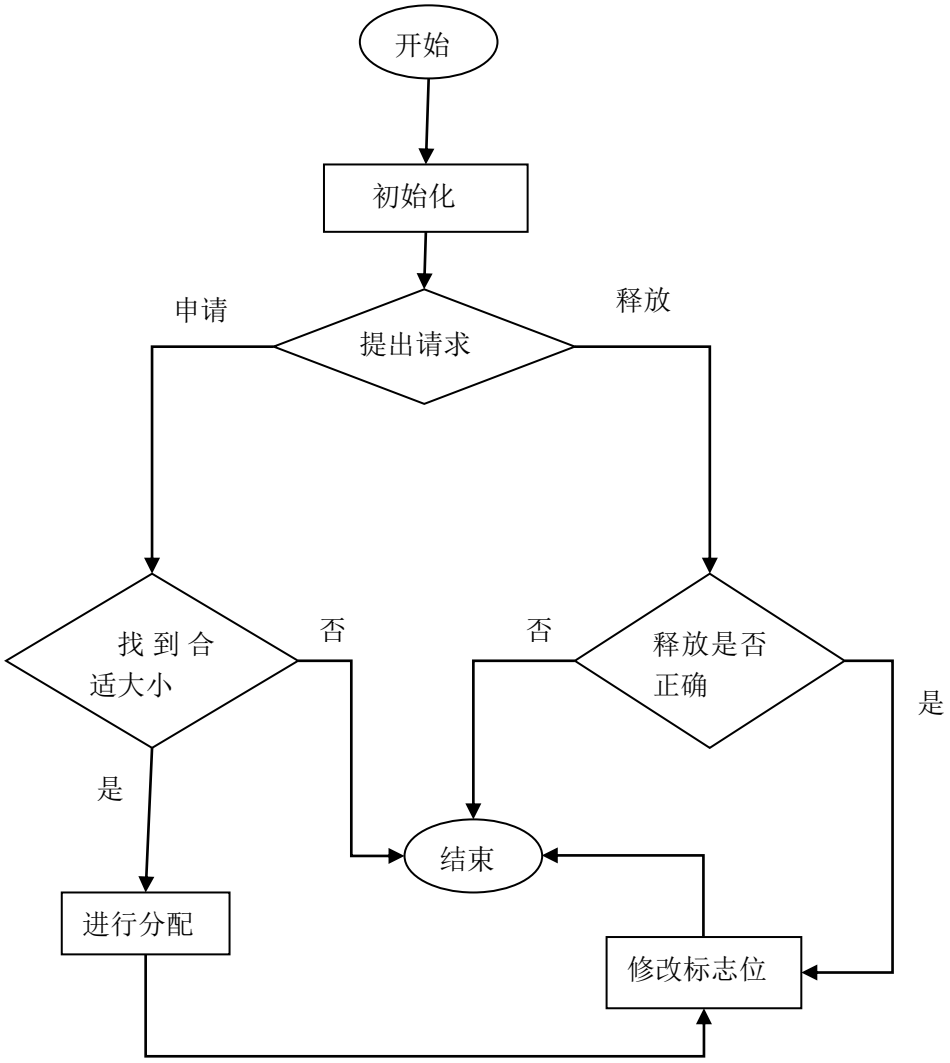
实验要求

（1）自行假设主存空间大小，预设操作系统所占大小并构造未分分区表；
（2）结合实验一，PCB 增加为：
{PID，要求运行时间，优先权，状态，所需内存大小，主存起始位置，PCB 指针}；
（3）采用最先适应算法分配主存空间；
（4）进程完成后，回收内存，并与相邻空闲分区合并。

实验主要思想

用链表管理内存，分配内存时，采用最先适应算法分配空间；回收内存时，需与相邻空闲分区合并。

流程图设计



实验主要函数详解

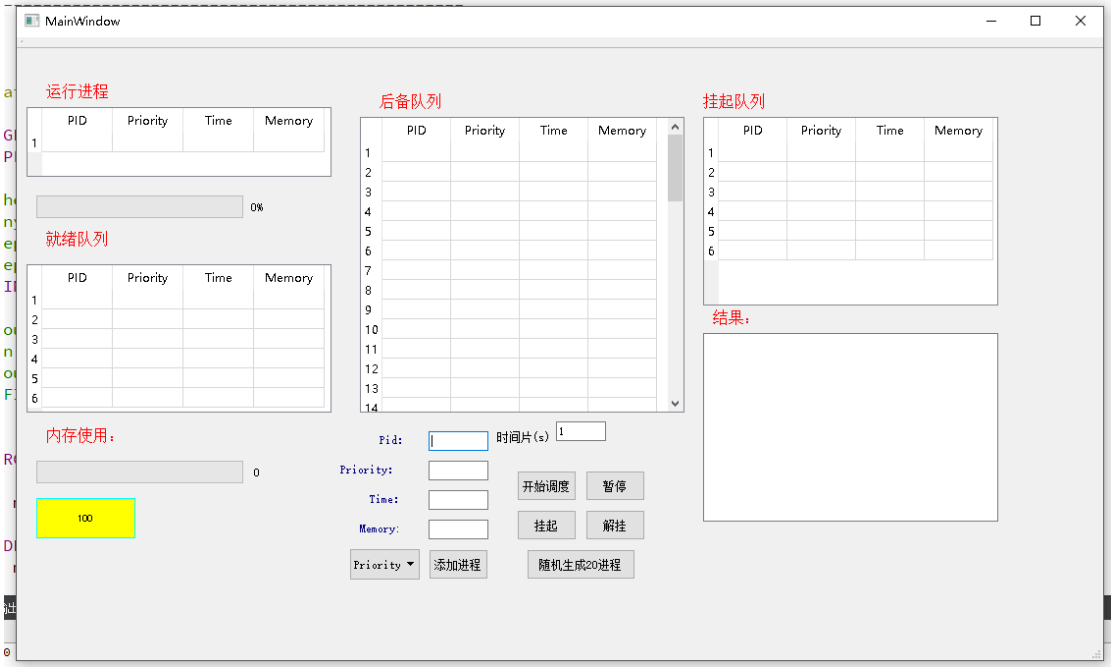
```
//内存结构体
struct mymemory{
    mymemory();
    mymemory(int f,int l,int s){
        front=f;
        length=l;
        state=s;
    }
    bool operator <(const mymemory b){
        return front<b.front;
```

```
    }

    int front;
    int length;
    int state;//0 表示 free,1 表示使用
};
//内存管理函数
class mem
{
public:
    vector<mymemory>mymem;
    bool add(PCB& a);//返回结果表示是否添加成功
    void remove(int memoryStart);//删除元素 a 的空间
    mem();
    int sum;
    void tight();

};
//删除内存块
void mem::remove(int memoryStart){
    for(vector<mymemory>::iterator it=mymem.begin();it!=mymem.end();it++){
        if((*it).front==memoryStart&&(*it).state==1){
            (*it).state=0;
            sum-=(*it).length;
            break;
        }
    }

    tight();

}
//内存添加 PCB
bool mem::add(PCB &a){
    if(a.memoryNeeds>100)
        return false;
    for(vector<mymemory>::iterator it=mymem.begin();it!=mymem.end();it++){
        if((*it).state==0&&(*it).length>=a.memoryNeeds){
            sum+=a.memoryNeeds;
            mymemory an((*it).front+a.memoryNeeds,(*it).length-a.memoryNeeds,0);//生成
空内存分区
            (*it).length=a.memoryNeeds;
            (*it).state=1;
            a.memoryStart=(*it).front;
            mymem.push_back(an);
```

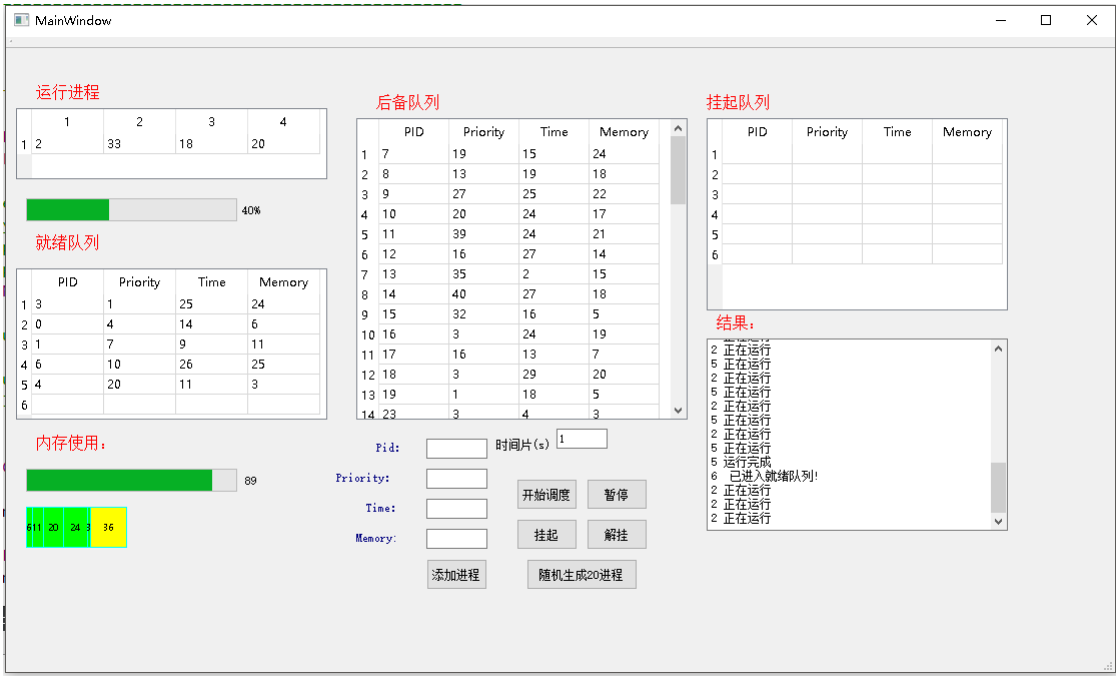
```
                tight();
                return true;
            }
        }
        cout<<"false"<<endl;
        return false;
}
//紧凑
void mem::tight(){
        sort(mymem.begin(),mymem.end());
        vector<mymemory>::iterator it=mymem.begin()+1;
        while(it!=mymem.end()){
            if((*it).state==0&&(*(it-1)).state==0){
                (*(it-1)).length+=(*it).length;
                it=mymem.erase(it);
            }else{
                it++;
            }
        }

}
```

实验结果

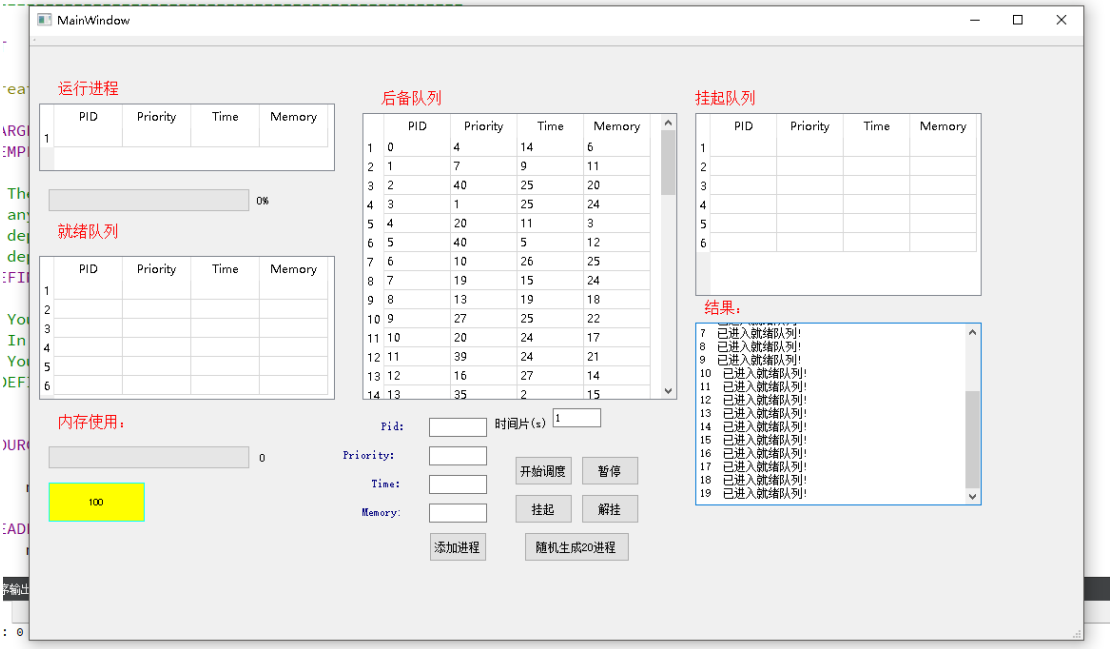
运行前：



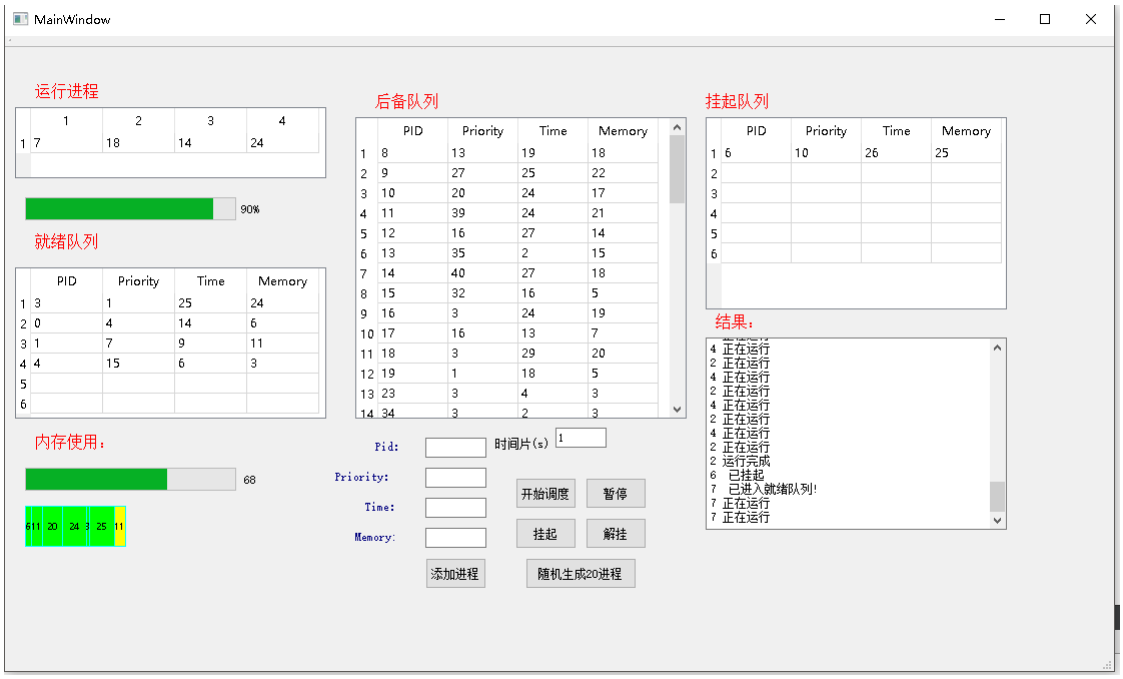
开始调度：


MainWindow
运行进程
1 2 3 4
1 2 33 18 20
40%
就绪队列
PID Priority Time Memory
1 3 1 25 24
2 0 4 14 6
3 1 7 9 11
4 6 10 26 25
5 4 20 11 3
6
内存使用：
89
611 20 24 3 36
后备队列
PID Priority Time Memory
1 7 19 15 24
2 8 13 19 18
3 9 27 25 22
4 10 20 24 17
5 11 39 24 21
6 12 16 27 14
7 13 35 2 15
8 14 40 27 18
9 15 32 16 5
10 16 3 24 19
11 17 16 13 7
12 18 3 29 20
13 19 1 18 5
14 23 3 4 3
Pid:
时间片(s) 1
Priority:
Time:
Memory:
开始调度
暂停
挂起
解挂
添加进程
随机生成20进程
挂起队列
PID Priority Time Memory
1
2
3
4
5
6
结果：
2 正在运行
5 正在运行
2 正在运行
5 正在运行
2 正在运行
5 正在运行
2 正在运行
5 正在运行
5 运行完成
6 已进入就绪队列!
2 正在运行
2 正在运行
2 正在运行

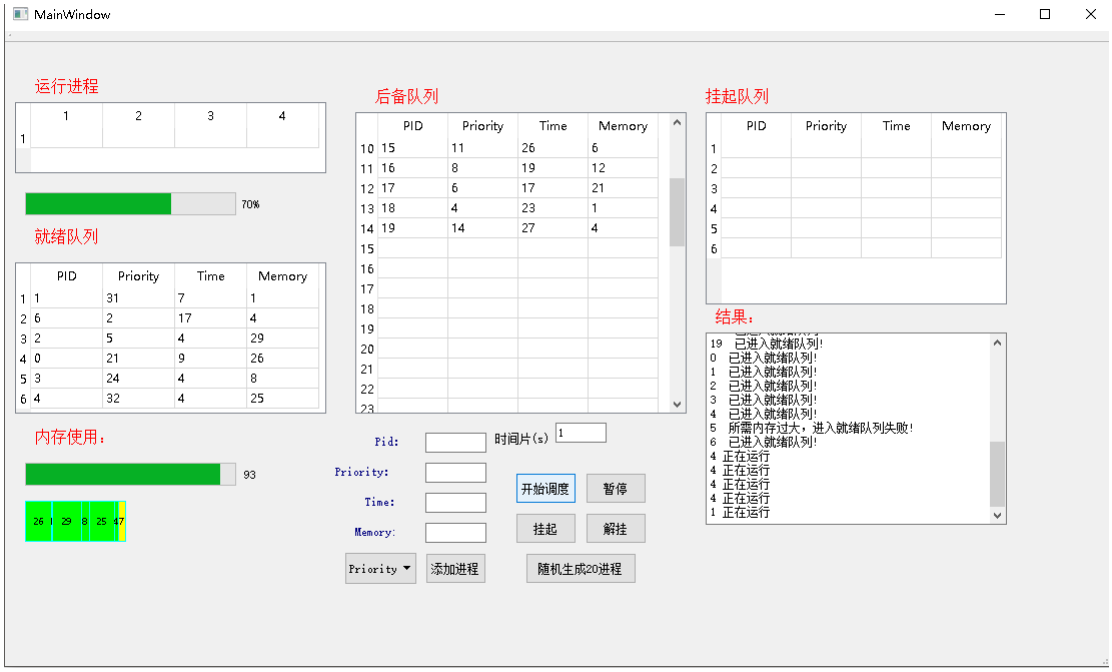
添加进程:


MainWindow
运行进程
PID Priority Time Memory
1
0%
就绪队列
PID Priority Time Memory
1
2
3
4
5
6
内存使用：
0
100
后备队列
PID Priority Time Memory
1 0 4 14 6
2 1 7 9 11
3 2 40 25 20
4 3 1 25 24
5 4 20 11 3
6 5 40 5 12
7 6 10 26 25
8 7 19 15 24
9 8 13 19 18
10 9 27 25 22
11 10 20 24 17
12 11 39 24 21
13 12 16 27 14
14 13 35 2 15
Pid:
时间片(s) 1
Priority:
Time:
Memory:
开始调度
暂停
挂起
解挂
添加进程
随机生成20进程
挂起队列
PID Priority Time Memory
1
2
3
4
5
6
结果：
7 已进入就绪队列!
8 已进入就绪队列!
9 已进入就绪队列!
10 已进入就绪队列!
11 已进入就绪队列!
12 已进入就绪队列!
13 已进入就绪队列!
14 已进入就绪队列!
15 已进入就绪队列!
16 已进入就绪队列!
17 已进入就绪队列!
18 已进入就绪队列!
19 已进入就绪队列!

挂起：



解挂：



实验感想

通过这次实验，将书上的理论知识和实际代码结合起来，我对分区分配算法有了新的认识。在实验一的基础上添加了内存管理机制。