中南大學

CENTRAL SOUTH UNIVERSITY

操作系统原理
实验报告

学生姓名	
学　　号	
专业班级	
指导教师	
学　　院	计算机学院
完成时间	

目录

一　实验概述

（一）实验目的

多道系统中，进程与进程之间存在同步与互斥关系。当就绪进程数大于处理机数时，需按照某种策略决定哪些进程先占用处理机。在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。

本实验模拟实现处理机调度及内存分配及回收机制，以对处理机调度的工作原理以及内存管理的工作过程进行更深入的了解。

（二）实验内容及要求

1．实验内容

（1）选择一个调度算法，实现处理机调度；

（2）结合（1）实现主存储器空间的分配和回收。

2.实验具体要求

（1）设计一个抢占式优先权调度算法实现多处理机调度的程序，并且实现在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。

（2）PCB 内容包括：进程名/PID；要求运行时间（单位时间）；优先权；状态；进程属性：独立进程、同步进程（前趋、后继）。

（3）可以随机输入若干进程，可随时添加进程，并按优先权排序；

（4）从就绪队首选进程运行：优先权-1；要求运行时间-1；要求运行时间为 0 时，撤销该进程；一个时间片结束后重新排序，进行下轮调度；

（5） 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。表目内容：起址、长度、状态（未分/空表目）。对内存空间分配采用首次适应算法。

（6）进程完成后，回收主存，并与相邻空闲分区合并。

（7）设置后备队列和挂起状态。若内存空间足够，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制定挂起进程解挂入就绪队列。

（8） 最好采用图形界面；

二 需求分析

（一）了解基本原理

①抢占式优先权调度算法：当一个进程正在处理机上运行时，若有某个优先级更高的进程进入就绪队列，则立即暂停正在运行的进程，将处理机分配给优先级更高的进程。而根据进程创建后其优先级是否可以改变，可以将进程优先级分为以下两种：

静态优先级。优先级是在创建进程时确定的，且在进程的整个运行期间保持不变。确定静态优先级的主要依据有进程类型、进程对资源的要求、用户要求。

动态优先级。在进程运行过程中，根据进程情况的变化动态调整优先级。动态调整优先级的主要依据有进程占有 CPU 时间的长短、就绪进程等待 CPU 时间的长短。

②动态分区分配:就是在处理作业的过程中，建立分区，依用户请求的大小分配分区。在分区回收的过程中会涉及一个空间利用效率相关的放置策略，即选择空闲区的策略。

内存空间分配采用首次适应算法：空闲分区以地址递附的次序链接。分配内存时，从链首开始顺序查找，找到大小能满足要求的第 个空闲分区分配给作业。

（二）确定软件基本功能

①可随机输入若干进程，并按优先权排序。

②PCB 的内容包括 PID，运行时间，优先权，状态，前趋，后继，在内存中的起始位置，需要的内存空间的大小。

③设置后备队列和挂起状态。若内存空间足够，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制定挂起进程解挂入就绪队列。

④从就绪队首选进程运行：优先权-1、要求运行时间-1、要求运行时间=0 时，撤销该进程

⑤预设操作系统所占大小并构造未分分区表。表目内容：起址、长度、状态。

⑥进程完成后，回收主存，并与相邻空闲分区合并。

（三）查找相关资料

①主程序模块分为四类：1. 图形界面管理模块 2. 按钮事件触发模块 3. 进程控制模块（PCB）4. 处理机调度模块

②可视化设计主要使用了 javax.swing（轻量组件）和 java.awt（组件绘制及管理）这两个工具包。

三 总体设计

输入进程名、优先级、运行时间、需要空间，创建作业→进入后备队列→由作业调度为作业创建进程，分配内存资源，创建PCB→进入就绪队列→循环进行进程调度→某进程完成→回收内存，将阻塞的后继进程调回就绪队列→循环进行程序调度

……

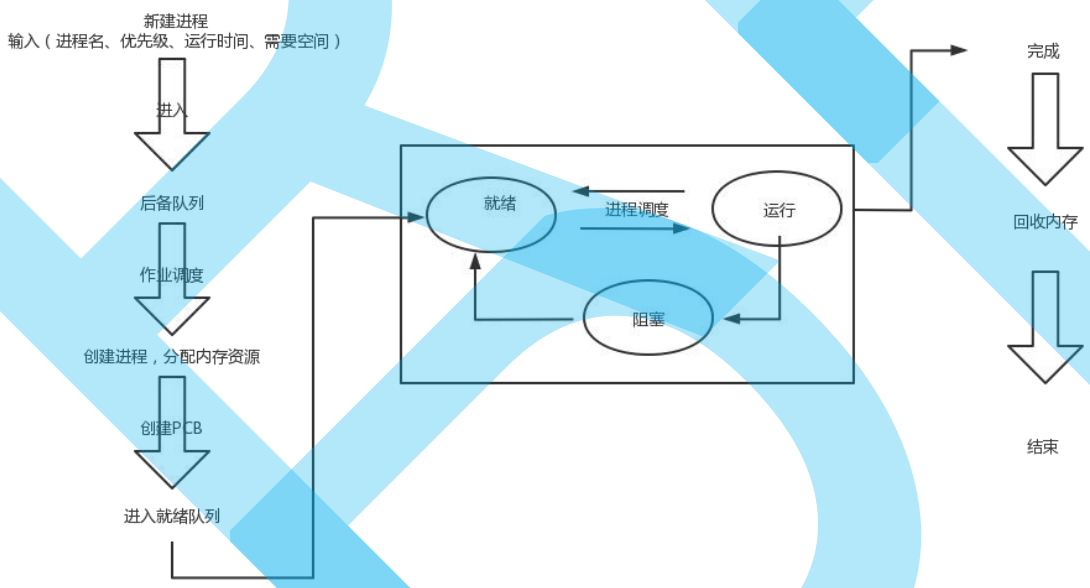

图 3-1 总体流程图

四 详细设计

（一）进程管理——PCB 进程控制块

```
/*****************************进程管理PCB进程块*********************************/
package OSmin;
public class PCB {
    static final int 运行态 = 1;   //运行态=1
    static final int 就绪态 = 2;   //就绪态=2
    static final int 就绪挂起 = 3; //  就绪挂起=3
    private String processName;    //进程名称
    private int runTime;           //进程运行时间
    private int prority;           //进程优先级
    private int processState;      //进程状态
    private int base;              //进程基址
    private int limit;             //限制长度
    public PCB(String name, int time,int pro,int base,int limit)
    {//-----------------------------------PCB包含进程参数
        this.processName = name;
        this.runTime = time;
        this.prority = pro;
        this.processState = 0;
        this.limit = limit;
        this.base = base;
    }
    public PCB(){}
    public void setProcessName(String name)//设置进程名
    {
        this.processName = name;
    }
    public String getProcessName()//---------获取进程名
    {
        return processName;
    }
    public void setRunTime(int time)//-------设置运行时间
    {
        this.runTime = time;
    }
    public int getRunTime()//----------------获取运行时间
    {
        return this.runTime;
    }
    public void setPrority(int prority)//----设置优先级
```

```
    {
        this.prority = prority;
    }
    public int getPrority()//----------------获取优先级
    {
        return this.prority;
    }
    public void setProcessState(int state)//-设置运行状态
    {
        this.processState = state;
    }
    public String getProcessState()//--------获取运行状态
    {
        String s = new String();
        if(this.processState == 1)
        {
            s = "运行态";
        }
        else if(this.processState == 2)
        {
            s = "就绪态";
        }
        else if(this.processState == 3)
        {
            s = "就绪挂起";
        }
        return s;
    }
    public void setBase(int base)//----------设置进程基址
    {
        this.base = base;
    }
    public int getBase()//------------------获取进程基址
    {
        return this.base;
    }
    public void setLimit(int limit)//--------设置进程长度
    {
        this.limit = limit;
    }
    public int getLimit()//------------------获取进程长度
    {
        return this.limit;
    }
}
```

（二）内存管理——动态可变分区的首次适应算法

```
/**********************首次适应算法的动态可变分区分配*****************************/
package OSmin;
public class MMA {//Memory Management and Allocation内存管理与动态分区分配
    private int Base;    //分区基址
    private int length;  //分区长度
    private int divFlag; //分区标志
    public MMA(int Base,int length,int divFlag)
    {
        this.Base = Base;
        this.divFlag = divFlag;
        this.length = length;
    }
    public MMA(){}
    public void setDivBase(int base)//-设置分区基址
    {
        this.Base = base;
    }
    public int getDivBase()//----------获取分区基址
    {
        return this.Base;
    }
    public void setLength(int length)//设置分区长度
    {
        this.length = length;
    }
    public int getLength()//-----------获取分区长度
    {
        return this.length;
    }
    public void setDivFlag(int flag)//-设置分区标志
    {
        this.divFlag = flag;
    }
    public int getDivFalg()//----------获取分区标志
    {
        return this.divFlag;
    }
}
```

（三）图形界面管理总类——各个函数模块和可视化界面管理

```
/**********************************相关包导入**********************************/
package OSmin;
import javax.swing.*;   //轻量级组件
import java.util.*;     //通用工具包
import java.awt.*;      //用于创建用户界面和绘制图形图像
import java.awt.event.*;//可由awt组件所激发的各类事件的接口和类
/**********************************图形界面管理总类**********************************/
public class Main {
    private JList jobList;    //作业框
    private JList readyList;  //就绪框
    private JList waitingList;//等待框

    private JButton susButton;  //挂起
    private JButton relaxButton;//解挂
    private JButton startButton;//开始
    private JButton newButton;  //新建进程

    private JLabel nameLabel;   //进程名
    private JLabel proritylabel;//优先级
    private JLabel timeLabel;   //运行时间
    private JLabel jobLabel;    //需要空间
    private JLabel readyLabel;  //就绪队列
    private JLabel waitingLabel;//等待队列
    private JLabel runningLabel; //1号处理机
    private JLabel runningLabel1;//2号处理机
    private JLabel spaceLabel;   //内存空间
    private JLabel divLabel;     //未分分区表
    private JLabel allocLabel;   //内存分配表

    private JTable readyTable;  //就绪队列表
    private JTable runningTable;//1号处理机表
    private JTable runningTable1;//2号处理机表
    private JTable divTable;    //未分分区表
    private JTable allocTable;  //内存分配表

    private JTextField nameText;//进程名文本框
    private JTextField timeText;//运行时间文本框
    private JTextField spaceText;//空间文本框
    private JComboBox proritycom;//优先级文本框

    private JPanel newPanel;    //新建面版容器（上）
    private JPanel readyPanel;  //就绪面版容器（中）
    private JPanel waitingPanel;//等待面版容器（下）
```

```
/*vector矢量队列，它是一个队列，具有添加、删除、修改、遍历等功能，支持随机访问，Vector可实现自动增长的对象数组*/
    Vector<String> jobVectorName;
    Vector<PCB> jobDtoVector;
    Vector<String> waitingVectorName;
    Vector<PCB> waitingDtoVector;
    //PCB进程控制块
    PCB[] readyDtoArray;
    PCB[] newDtoArray;
    MMA[] divDtoArray;
    PCB[] newSort;
    //对象类
    Object[][] readydata;
    Object[][] runningdata; //1号处理机
    Object[][] runningdata1;//2号处理机
    Object[][] divdata;
    Object[][] allocdata;

    int first=0;
    int end;
    int point = 0;
    int cpu2 = 0;
    PCB a;
    PCB la;
    int aa = 0;  //内存基址
    int bb = 360;//内存长度
    int max1 = -1;
/**********************************总面板页面管理*****************************/
    @SuppressWarnings({ "rawtypes", "unchecked" })
    public Main() {
        a = new PCB();
        la = new PCB();
        first = 0;
        end = 0;
        point = 0;
        JFrame jf = new JFrame("操作系统实验-大数据2103-LukiRyan");//主面板名
        Container c = jf.getContentPane();
        c.setLayout(null);
        c.setBackground(Color.orange);//闲置区块设为橙色
/*************************面板管理颜色设置*************************************/
        newPanel = new JPanel();
        newPanel.setLayout(null);
        newPanel.setBackground(Color.pink);//新建面板背景（上）粉色
        readyPanel = new JPanel();
```

```
        readyPanel.setLayout(null);
        readyPanel.setBackground(Color.white);//运行队列块（中）白
        waitingPanel = new JPanel();
        waitingPanel.setLayout(null);
        waitingPanel.setBackground(Color.white);//等待队列块（下）白
/*****************************按钮管理颜色设置*********************************/
        susButton = new JButton("挂起");
        relaxButton = new JButton("解挂");
        startButton = new JButton("开始");
        newButton = new JButton("新建进程");
        newButton.setBackground(Color.cyan);
        susButton.setBackground(Color.cyan);
        relaxButton.setBackground(Color.cyan);
        startButton.setBackground(Color.cyan);
/*****************************标签管理设置*********************************/
        nameLabel = new JLabel("进程名: ");
        proritylabel = new JLabel("优先级: ");
        timeLabel = new JLabel("运行时间: ");
        spaceLabel = new JLabel("需要空间: ");
        jobLabel = new JLabel("后备队列");
        readyLabel = new JLabel("就绪队列");
        waitingLabel = new JLabel("等待队列");
        runningLabel = new JLabel("处理机1号");
        runningLabel1 = new JLabel("处理机2号");
        divLabel = new JLabel("未分分区表");
        allocLabel = new JLabel("内存分配表");
        //文本框管理---新建进程
        nameText = new JTextField();
        timeText = new JTextField();
        spaceText = new JTextField();
        proritycom = new JComboBox();
        proritycom.setToolTipText("优先级");
        for (int i = 1; i <= 20; i++) { //设置优先1-20
            proritycom.addItem(i);// 相当于proritylArray[i] = i
        }
/*****************************各种控件详细管理*********************************/
        readyDtoArray = new PCB[7];//--------------------------就绪队列
        newSort = new PCB[7];
        for (int i = 0; i < 7; i++) {//小于7个进程
            newSort[i] = new PCB();
        }
        newDtoArray = new PCB[100];//--------------------------后备队列
        jobDtoVector = new Vector();
        jobVectorName = new Vector();
        waitingDtoVector = new Vector();
```

```
        waitingVectorName = new Vector();
        divDtoArray = new MMA[20];//-----------------------未分分区队列
        for (int i = 0; i < 20; i++) {//小于20
            divDtoArray[i] = new MMA();
            divDtoArray[i].setDivFlag(0);
        }
        divDtoArray[0].setDivFlag(1);//---------------分区标志
        divDtoArray[0].setDivBase(aa);//---------------内存基址
        divDtoArray[0].setLength(bb);//----------------内存长度
        readydata = new Object[7][4];//----------------就绪数据六行四列
        runningdata = new Object[1][3];//--------------运行数据一行三列
        runningdata1 = new Object[1][3];//-------------运行数据-一行三列
        divdata = new Object[20][3];//-----------------未分分区数据-二十行三列
        allocdata = new Object[20][3];//---------------内存分配数据-二十行三列
        String[] col1 = { "进程名", "时间", "优先级", "状态" };//--就绪队列表头
        String[] col2 = { "进程名", "时间", "优先级" };//----------处理机表头
        String[] col3 = { "起址", "长度", "状态" };//------------未分分区表头
        String[] col4 = { "起址", "长度", "占用进程" };//--------内存分配表头
        readyTable = new JTable(readydata, col1);
        runningTable = new JTable(runningdata, col2);//---------1号处理机
        runningTable.setRowHeight(24);//行高
        runningTable.setEnabled(false);

        runningTable1 = new JTable(runningdata1, col2);//------2号处理机
        runningTable1.setRowHeight(24);//行高
        runningTable1.setEnabled(false);

        allocTable = new JTable(allocdata, col4);
        allocTable.setEnabled(false);
        divTable = new JTable(divdata, col3);
        divTable.setEnabled(false);
        divTable.setValueAt(String.valueOf(0), 0, 0);
        divTable.setValueAt(String.valueOf(360), 0, 1);
        divTable.setValueAt(String.valueOf("空闲"), 0, 2);//标志空闲
/**********************************滚动条布置**************************************/
        JScrollPane runningSP = new JScrollPane();
        JScrollPane runningSP1 = new JScrollPane();
        JScrollPane readySP2 = new JScrollPane();
        JScrollPane divSP = new JScrollPane();
        JScrollPane allocSP = new JScrollPane();
        runningSP.getViewport().add(runningTable);
        runningSP1.getViewport().add(runningTable1);
        readySP2.getViewport().add(readyTable);
        divSP.getViewport().add(divTable);
        allocSP.getViewport().add(allocTable);
```

```
        jobList = new JList();
        waitingList = new JList();
        JScrollPane jobSP = new JScrollPane(jobList);
        JScrollPane waitingSP = new JScrollPane(waitingList);
/*************************新建进程的界面布置（上）*******************************/
        newPanel.setSize(950, 90);//总大小
        newPanel.setLocation(0, 0);//总位置
        nameLabel.setSize(80, 30);
        nameLabel.setLocation(20, 30);
        nameText.setSize(80, 30);
        nameText.setLocation(66, 30);//进程名
        proritylabel.setSize(80, 30);
        proritylabel.setLocation(210, 30);
        proritycom.setSize(80, 30);
        proritycom.setLocation(256, 30);//优先级
        timeLabel.setSize(80, 30);
        timeLabel.setLocation(410, 30);
        timeText.setSize(80,30);
        timeText.setLocation(470, 30);//运行时间
        spaceLabel.setSize(80, 30);
        spaceLabel.setLocation(590, 30);
        spaceText.setSize(80, 30);
        spaceText.setLocation(650, 30); //需要空间
        newButton.setSize(100, 30);
        newButton.setLocation(785, 30); //新建进程
/*************************进程运行块的界面布置（中）*******************************/
        readyPanel.setSize(950, 180);//总大小
        readyPanel.setLocation(0, 90);//总位置
        readyLabel.setSize(150, 22);
        readyLabel.setLocation(102, 2);//标签位置
        allocLabel.setSize(100, 20);
        allocLabel.setLocation(750, 0);//内存分配标签
        startButton.setSize(80, 25);
        startButton.setLocation(280, 40);//开始键位置
        susButton.setSize(80, 25);
        susButton.setLocation(280, 100);//挂起键位置
        readySP2.setSize(250, 135);
        readySP2.setLocation(10, 25);//就绪框位置
        runningLabel.setSize(100, 22);
        runningLabel.setLocation(475, 75);//处理机1标签
        runningSP.setSize(250, 49);
        runningSP.setLocation(380, 103);//处理机1位置
        allocSP.setSize(250, 135);
        allocSP.setLocation(660, 25);//内存分配表位置
```

```
/*************************后备等待未分区解挂的界面布置(下)************************/
        waitingPanel.setSize(950, 190);//总大小
        waitingPanel.setLocation(0, 270);//总位置
        jobLabel.setSize(100, 20);
        jobLabel.setLocation(40, 5);
        jobSP.setSize(110, 150);
        jobSP.setLocation(10, 25);//后备队列
        waitingLabel.setSize(100, 20);
        waitingLabel.setLocation(180, 5);
        waitingSP.setSize(110,150);
        waitingSP.setLocation(150, 25);//等待队列
        divLabel.setSize(100, 20);
        divLabel.setLocation(750, 5);
        divSP.setSize(250, 135);
        divSP.setLocation(660, 30);//未分分区表
        relaxButton.setSize(80, 25);
        relaxButton.setLocation(280, 25);//解挂按钮
        runningLabel1.setSize(100, 22);
        runningLabel1.setLocation(475, 0);//处理机2标签
        runningSP1.setSize(250, 49);
        runningSP1.setLocation(380, 28);//处理机2位置
/*********************************添加页面所有标签*********************************/
        c.add(newPanel);
        newPanel.add(nameLabel);
        newPanel.add(nameText);
        newPanel.add(proritylabel);
        newPanel.add(proritycom);
        newPanel.add(timeText);
        newPanel.add(timeLabel);
        newPanel.add(newButton);
        newPanel.add(spaceLabel);
        newPanel.add(spaceText);//新建进程区块
        c.add(readyPanel);
        readyPanel.add(readyLabel);
        readyPanel.add(allocLabel);
        readyPanel.add(runningLabel);
        readyPanel.add(startButton);
        readyPanel.add(susButton);
        readyPanel.add(allocSP);
        readyPanel.add(runningSP);
        readyPanel.add(readySP2);//运行进程区块
        c.add(waitingPanel);
        waitingPanel.add(runningLabel1);//为了美观把处理机2号放下面
        waitingPanel.add(runningSP1);
        waitingPanel.add(jobLabel);
```

```
        waitingPanel.add(jobSP);
        waitingPanel.add(waitingLabel);
        waitingPanel.add(waitingSP);
        waitingPanel.add(divLabel);
        waitingPanel.add(divSP);
        waitingPanel.add(relaxButton);//等待后备区块
/************************主面板的界面布置+按钮事件触发*****************************/
        jf.setSize(950, 496);//大小
        jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        jf.setLocationRelativeTo(null);//把窗口位置设置到屏幕中心
        jf.setVisible(true);
        jf.setBackground(Color.orange);//闲置区块设为橙色
        startButton.addActionListener(new MyActionListener());
        newButton.addActionListener(new MyActionListener());
        susButton.addActionListener(new MyActionListener());
        relaxButton.addActionListener(new MyActionListener());
    }
/*********************************挂起操作方法*************************************/
    public void sus() {
        try {
            Thread.sleep(1000);
        } catch (Exception ex) {
        }
    }
/*************************构造按钮触发事件类*************************************/
    class MyActionListener implements ActionListener {
        @SuppressWarnings("unchecked")
        public void actionPerformed(ActionEvent e) {
            int count = 0;
            PCB test = new PCB();
            JButton jb = (JButton) e.getSource();
            int max = -1;
            max1 = -1;
/*******************************构造开始按钮的触发事件*****************************/
            if (jb == startButton) {
                int runAllocFlag = -1;
                if (((String) runningTable.getValueAt(0, 0) == null
                        || (String) runningTable.getValueAt(0, 0) == "")) {
                    try {
                        Thread.sleep(0);//挂起
                    } catch (Exception ex) {
                    }
                    // System.out.println("到3");
                    //状态控制 1运行 2就绪 3等待
                    int sum = 0;
```

```
                for (int j = first; j != end;) {
                    if (readyDtoArray[j].getProcessState().equals("就绪态")) {
                        max = j;//max相当于一个记录位，用来记录下一个在cpu1上运行的进
程
                        break;
                    }
                    j = (j + 1) % 6;//队列排序 先进先出 往下滚动一列
                }
                //优先级控制
                for (int j = first; j % 6 != end;) {
                    if (readyDtoArray[j].getProcessState().equals("就绪态")) {
                        sum ++;
                        if (readyDtoArray[j].getPrority() >
readyDtoArray[max].getPrority()) {
                            max = j;
                        }
                    }
                    j = (j + 1) % 6;//往下滚动一列
                }

                if (max >= 0) {//当有其他进程时
                    a = readyDtoArray[max];//--------------交换
                    readyDtoArray[max] = readyDtoArray[first];
                    readyDtoArray[first] = a;
                    readyTable.setValueAt(readyDtoArray[max].getProcessName(),
max, 0);
                    readyTable.setValueAt(readyDtoArray[max].getRunTime(),max,
1);
                    readyTable.setValueAt(readyDtoArray[max].getPrority(),max,
2);
                    readyTable.setValueAt(readyDtoArray[max].getProcessState(),
max, 3);//就绪队列显示
                    readyTable.setValueAt("", first, 0);
                    readyTable.setValueAt("", first, 1);
                    readyTable.setValueAt("", first, 2);
                    readyTable.setValueAt("", first, 3);
                    runningTable.setValueAt(a.getProcessName(), 0, 0);
                    runningTable.setValueAt(a.getRunTime(), 0, 1);
                    runningTable.setValueAt(a.getPrority(), 0, 2);//处理机显示

readyDtoArray[first].setRunTime(readyDtoArray[first].getRunTime() - 1);//运行时间减
一
                    if (0 != readyDtoArray[first].getPrority()) {

readyDtoArray[first].setPrority(readyDtoArray[first].getPrority() - 1);//优先级减一
```

```
                    }
                    first = (first + 1) % 6;//往下滚动一列
                    if(sum>1) {
                        for (int j = first; j != end;) {
                            if (readyDtoArray[j].getProcessState().equals("就绪态
")) {
                                max1 = j;//max1相当于一个记录位，用来记录下一个在cpu2
上运行的进程
                                break;
                            }
                            j = (j + 1) % 6;//队列排序 先进先出
                        }
                        //优先级控制
                        for (int j = first; j % 6 != end;) {
                            if (readyDtoArray[j].getProcessState().equals("就绪态
")  ) {
                                if (readyDtoArray[j].getPrority() >
readyDtoArray[max1].getPrority()) {
                                    max1 = j;
                                }
                            }
                            j = (j + 1) % 6;
                        }
                    }
                    if (max1 >= 0) {//启用处理机2
                        cpu2 = 1;
                        la = readyDtoArray[max1];//--------------交换
                        readyDtoArray[max1] = readyDtoArray[first];
                        readyDtoArray[first] = la;


readyTable.setValueAt(readyDtoArray[max1].getProcessName(), max1, 0);

readyTable.setValueAt(readyDtoArray[max1].getRunTime(),max1, 1);

readyTable.setValueAt(readyDtoArray[max1].getPrority(),max1, 2);

readyTable.setValueAt(readyDtoArray[max1].getProcessState(), max1, 3);
                        readyTable.setValueAt("", first, 0);
                        readyTable.setValueAt("", first, 1);
                        readyTable.setValueAt("", first, 2);
                        readyTable.setValueAt("", first, 3);


runningTable1.setValueAt(readyDtoArray[first].getProcessName(), 0, 0);
```

```
runningTable1.setValueAt(readyDtoArray[first].getRunTime(), 0, 1);

runningTable1.setValueAt(readyDtoArray[first].getPrority(), 0, 2);

readyDtoArray[first].setRunTime(readyDtoArray[first].getRunTime() - 1);
                        if (0 != readyDtoArray[first].getPrority()) {

readyDtoArray[first].setPrority(readyDtoArray[first].getPrority() - 1);
                        }
                        first = (first + 1) % 6;
                    }
                } else {
                    System.out.println("cpu正在等待中……"); //处理机上无进程时
                }
            } else {
                runningTable.setValueAt("", 0, 0);
                runningTable.setValueAt("", 0, 1);
                runningTable.setValueAt("", 0, 2);//处理机1显示为空

                runningTable1.setValueAt("", 0, 0);
                runningTable1.setValueAt("", 0, 1);
                runningTable1.setValueAt("", 0, 2);//处理机2显示为空
                if (a.getRunTime() <= 0) {//如果运行时间为0则撤销进程，否则将进程重新添加到就绪队列中
                    for (int i = 0; i < point; i++) {// 收回内存空间
                        if (newSort[i].getBase() >= a.getBase()) {
                            newSort[i] = newSort[i + 1];
                        }
                    }
                    point--;//进程数量减一
/*****************************设置内存分配表的内容*********************************/
                    for (int i = 0; i < point; i++) {

allocTable.setValueAt(String.valueOf(newSort[i].getBase()), i, 0); //基址

allocTable.setValueAt(String.valueOf(newSort[i].getLimit()), i, 1);//长度
                        allocTable.setValueAt(newSort[i].getProcessName(),i, 2);
//占用进程名
                    }//显示point个进程的内存分配情况
                    allocTable.setValueAt("", point, 0);
                    allocTable.setValueAt("", point, 1);
                    allocTable.setValueAt("", point, 2);//其他默认设置为空
                    // 把收回的内存加入到记录未分分区的数组
                    int memoryEnd = 0;
```

```
                    int location = 0;
                    int up = -1;
                    int down = -1;
                    for (int i = 0; i < 20; i++) {
                        if (divDtoArray[i].getDivFalg() == 1) {//如果分区标志为1
                            memoryEnd = divDtoArray[i].getDivBase()+
divDtoArray[i].getLength();//基址+长度=终止地址位置
                            if (memoryEnd == a.getBase()) {//如果i的终止地址==a的起
始地址
                                up = i;                    //说明i是a的上界
                            }
                            if (divDtoArray[i].getDivBase() == (a.getBase() +
a.getLimit())) {
                                down = i;//如果i的起始地址==a的终止地址，i就是a的x下
界
                            }
                        }
                    }
                    if (up >= 0 && down >= 0) {//有进程存在
                        divDtoArray[up].setLength((divDtoArray[up].getLength()+
a.getLimit() + divDtoArray[down].getLength()));
                        divDtoArray[down].setDivFlag(0);
                        for (int i = (down + 1); i < 20; i++) {
                            if (divDtoArray[i].getDivFalg() == 1) {
                                divDtoArray[i -
1].setDivBase(divDtoArray[i].getDivBase());
                                divDtoArray[i - 1].setDivFlag(1);
                                divDtoArray[i -
1].setLength(divDtoArray[i].getLength());
                                divDtoArray[i].setDivFlag(0);//处理完标志设置为0
                            }
                            else {
                                divTable.setValueAt("", i - 1, 0);
                                divTable.setValueAt("", i - 1, 1);
                                divTable.setValueAt("", i - 1, 2);//未分分区表默认为
空
                                break;
                            }
                        }
                    } else if (up >= 0 && down < 0) {
                        divDtoArray[up].setLength((divDtoArray[up].getLength() +
a.getLimit()));
                    } else if (up < 0 && down >= 0) {

divDtoArray[down].setLength((divDtoArray[down].getLength() + a.getLimit()));
```

```
                    divDtoArray[down].setDivBase(a.getBase());
                } else if (up < 0 && down < 0) {//没有进程存在的时候？
                    for (int i = 0; i < 20; i++) {
                        if (divDtoArray[i].getDivBase() > a.getBase()||
divDtoArray[i].getDivFalg() == 0) {
                            location = i;
                            break;
                        }
                    }
                    for (int i = 20; i > location; i--) {
                        if (divDtoArray[i - 1].getDivFalg() == 1) {
                            divDtoArray[i].setDivBase(divDtoArray[i -
1].getDivBase());
                            divDtoArray[i].setDivFlag(1);
                            divDtoArray[i].setLength(divDtoArray[i -
1].getLength());
                        }
                    }
                    divDtoArray[location].setDivBase(a.getBase());
                    divDtoArray[location].setDivFlag(1);
                    divDtoArray[location].setLength(a.getLimit());
                }
/*****************************设置未分分区表的内容*********************************/
                for (int i = 0; i < 20; i++) {
                    if (divDtoArray[i].getDivFalg() == 1) {

divTable.setValueAt(String.valueOf(divDtoArray[i].getDivBase()),i, 0);

divTable.setValueAt(String.valueOf(divDtoArray[i].getLength()),i, 1);
                        divTable.setValueAt(String.valueOf("空闲"),i, 2);//将
未分区表的区间都设置为空闲！！！
                    }
                }
                if (!jobDtoVector.isEmpty()) {
                    int runLength = 0;//后备队列
                    PCB jobToReady = (PCB) jobDtoVector.elementAt(0);
                    for (int i = 0; i < 20; i++) {
                        if (divDtoArray[i].getDivFalg() == 1) {
                            if (divDtoArray[i].getLength() >=
jobToReady.getLimit()) {
                                runAllocFlag = i;
                                break;
                            }
                        }
                    }
```

```
                        if (runAllocFlag >= 0) {
                            jobDtoVector.removeElementAt(0);

jobVectorName.remove(jobVectorName.indexOf(jobToReady.getProcessName()));
                            jobList.setListData(jobVectorName);
                            jobToReady.setProcessState(PCB.就绪态);

jobToReady.setBase(divDtoArray[runAllocFlag].getDivBase());
                            runLength = divDtoArray[runAllocFlag].getLength() -
jobToReady.getLimit();
                            if (runLength == 0) {
                                int i = runAllocFlag;
                                divDtoArray[i].setDivFlag(0);
                                for (; i < 19; i++) {
                                    if (divDtoArray[i + 1].getDivFalg() == 1) {
                                        divDtoArray[i] = divDtoArray[i + 1];
                                        divDtoArray[i + 1].setDivFlag(0);
                                    }

divTable.setValueAt(String.valueOf(divDtoArray[i].getDivBase()), i, 0);

divTable.setValueAt(String.valueOf(divDtoArray[i].getLength()), i, 1);
                                }

divTable.setValueAt(String.valueOf(divDtoArray[i].getDivFalg()), i, 2);//回收产生了
新的分区！
                            } else if (runLength > 0) {
                                int c2 = divDtoArray[runAllocFlag].getDivBase()+
jobToReady.getLimit();
                                divDtoArray[runAllocFlag].setDivBase(c2);
                                divDtoArray[runAllocFlag].setLength(runLength);

divTable.setValueAt(String.valueOf(c2),runAllocFlag, 0);

divTable.setValueAt(String.valueOf(runLength),runAllocFlag, 1);
                            }
                            readyDtoArray[end] = jobToReady;
                            readyTable.setValueAt(jobToReady.getProcessName(),
end, 0);
                            readyTable.setValueAt(jobToReady.getRunTime(),end,
1);
                            readyTable.setValueAt(jobToReady.getPrority(),end,
2);
                            readyTable.setValueAt(jobToReady.getProcessState(),
end, 3);
```

```
                            end = (end + 1) % 6;
                            int runi = 0;// 用于记录当前新生成的PcbDTO对象应该插入到
newSort中的位置
                            for (; runi < point; runi++) {
                                if (jobToReady.getBase() < newSort[runi].getBase())
{
                                    break;
                                }
                            }
                            // 如果不是插入到数组末尾，则把比它大的都向后挪一位并设置
JTable中的显示
                            for (int i = point; i > runi; i--) {
                                newSort[i] = newSort[i - 1];

allocTable.setValueAt(String.valueOf(newSort[i].getBase()), i,0);

allocTable.setValueAt(String.valueOf(newSort[i].getLimit()), i,1);

allocTable.setValueAt(newSort[i].getProcessName(), i, 2);
                            }
                            // 插入新生成的对象
                            newSort[runi] = jobToReady;

allocTable.setValueAt(String.valueOf(jobToReady.getBase()),runi, 0);

allocTable.setValueAt(String.valueOf(jobToReady.getLimit()),runi, 1);
                            allocTable.setValueAt(jobToReady.getProcessName(),
runi, 2);
                            point++;
                        }
                    }
                } else {

                    readyDtoArray[end] = a;
                    readyTable.setValueAt(a.getProcessName(), end, 0);
                    readyTable.setValueAt(a.getRunTime(), end, 1);
                    readyTable.setValueAt(a.getPrority(), end, 2);
                    readyTable.setValueAt(a.getProcessState(), end, 3);
                    end = (end + 1) % 6;
                }
                if(cpu2 == 1) {
                    cpu2 = 0;
                    if(la.getRunTime() <= 0) {// 收回内存空间
                        for (int i = 0; i < point; i++) {
                            if (newSort[i].getBase() >= la.getBase()) {
```

```
                    newSort[i] = newSort[i + 1];
                }
            }
            point--;
```

```
/*****************************设置内存分配表的内容*****************************/
            for (int i = 0; i < point; i++) {

allocTable.setValueAt(String.valueOf(newSort[i].getBase()), i, 0);//基址

allocTable.setValueAt(String.valueOf(newSort[i].getLimit()), i, 1);//限制长度
                allocTable.setValueAt(newSort[i].getProcessName(),i,
2);//获取进程名
            }
            allocTable.setValueAt("", point, 0);
            allocTable.setValueAt("", point, 1);
            allocTable.setValueAt("", point, 2);
            // 把收回的内存加入到记录未分分区的数组
            int memoryEnd = 0;
            int location = 0;
            int up = -1;//
            int down = -1;
            for (int i = 0; i < 20; i++) {
                if (divDtoArray[i].getDivFalg() == 1) {
                    memoryEnd = divDtoArray[i].getDivBase()+
divDtoArray[i].getLength();
                    //a的下界相邻
                    if (memoryEnd == la.getBase()) {
                        up = i;
                    }
                    //a的上界相邻
                    if (divDtoArray[i].getDivBase() == (la.getBase() +
la.getLimit())) {
                        down = i;
                    }
                }
            }
            if (up >= 0 && down >= 0) {

divDtoArray[up].setLength((divDtoArray[up].getLength()+ la.getLimit() +
                        divDtoArray[down].getLength()));
                divDtoArray[down].setDivFlag(0);
                for (int i = (down + 1); i < 20; i++) {
                    if (divDtoArray[i].getDivFalg() == 1) {
                        divDtoArray[i -
```

```
1].setDivBase(divDtoArray[i].getDivBase());
                                    divDtoArray[i - 1].setDivFlag(1);
                                    divDtoArray[i -
1].setLength(divDtoArray[i].getLength());
                                    divDtoArray[i].setDivFlag(0);
                                } else {
                                    divTable.setValueAt("", i - 1, 0);
                                    divTable.setValueAt("", i - 1, 1);
                                    break;
                                }
                            }
                        } else if (up >= 0 && down < 0) {

divDtoArray[up].setLength((divDtoArray[up].getLength() + la.getLimit()));
                        } else if (up < 0 && down >= 0) {

divDtoArray[down].setLength((divDtoArray[down].getLength() + la.getLimit()));
                            divDtoArray[down].setDivBase(la.getBase());
                        } else if (up < 0 && down < 0) {
                            for (int i = 0; i < 20; i++) {
                                if (divDtoArray[i].getDivBase() > la.getBase()
                                        || divDtoArray[i].getDivFalg() == 0) {
                                    location = i;
                                    break;
                                }
                            }
                            for (int i = 20; i > location; i--) {
                                if (divDtoArray[i - 1].getDivFalg() == 1) {
                                    divDtoArray[i].setDivBase(divDtoArray[i -
1].getDivBase());
                                    divDtoArray[i].setDivFlag(1);
                                    divDtoArray[i].setLength(divDtoArray[i -
1].getLength());
                                }
                            }
                            divDtoArray[location].setDivBase(la.getBase());
                            divDtoArray[location].setDivFlag(1);
                            divDtoArray[location].setLength(la.getLimit());

                        }
/*********************************设置就绪队列表内容*********************************/
                        for (int i = 0; i < 20; i++) {
                            if (divDtoArray[i].getDivFalg() == 1) {//如果分区标志为
1
```

```
divTable.setValueAt(String.valueOf(divDtoArray[i].getDivBase()),i, 0);

divTable.setValueAt(String.valueOf(divDtoArray[i].getLength()),i, 1);
                            }
                        }
                        if (!jobDtoVector.isEmpty()) {//如果工作队列不为空
                            int runLength = 0;//后备队列
                            PCB jobToReady = (PCB) jobDtoVector.elementAt(0);
                            for (int i = 0; i < 20; i++) {
                                if (divDtoArray[i].getDivFalg() == 1) {
                                    if (divDtoArray[i].getLength() >=
jobToReady.getLimit()) {
                                        runAllocFlag = i;
                                        break;
                                    }
                                }
                            }
                            if (runAllocFlag >= 0) {
                                jobDtoVector.removeElementAt(0);

jobVectorName.remove(jobVectorName.indexOf(jobToReady.getProcessName()));
                                jobList.setListData(jobVectorName);
                                jobToReady.setProcessState(PCB.就绪态);

jobToReady.setBase(divDtoArray[runAllocFlag].getDivBase());
                                runLength = divDtoArray[runAllocFlag].getLength() -
jobToReady.getLimit();
                                if (runLength == 0) {
                                    int i = runAllocFlag;
                                    divDtoArray[i].setDivFlag(0);
                                    for (; i < 19; i++) {
                                        if (divDtoArray[i + 1].getDivFalg() == 1) {
                                            divDtoArray[i] = divDtoArray[i + 1];
                                            divDtoArray[i + 1].setDivFlag(0);
                                        }

divTable.setValueAt(String.valueOf(divDtoArray[i].getDivBase()), i, 0);

divTable.setValueAt(String.valueOf(divDtoArray[i].getLength()), i, 1);
                                    }
                                } else if (runLength > 0) {
                                    int c2 =
divDtoArray[runAllocFlag].getDivBase()+ jobToReady.getLimit();
                                    divDtoArray[runAllocFlag].setDivBase(c2);
```

```
divDtoArray[runAllocFlag].setLength(runLength);

divTable.setValueAt(String.valueOf(c2),runAllocFlag, 0);

divTable.setValueAt(String.valueOf(runLength),runAllocFlag, 1);
                                }
                                readyDtoArray[end] = jobToReady;

readyTable.setValueAt(jobToReady.getProcessName(), end, 0);

readyTable.setValueAt(jobToReady.getRunTime(),end, 1);

readyTable.setValueAt(jobToReady.getPrority(),end, 2);

readyTable.setValueAt(jobToReady.getProcessState(), end, 3);
                                end = (end + 1) % 6;
                                int runi = 0;// 用于记录当前新生成的PcbDTO对象应该插
入到newSort中的位置
                                for (; runi < point; runi++) {
                                    if (jobToReady.getBase() <
newSort[runi].getBase()) {
                                        break;
                                    }
                                }
                                // 如果不是插入到数组末尾，则把比它大的都向后挪一位并设
置JTable中的显示
                                for (int i = point; i > runi; i--) {
                                    newSort[i] = newSort[i - 1];

allocTable.setValueAt(String.valueOf(newSort[i].getBase()), i,0);

allocTable.setValueAt(String.valueOf(newSort[i].getLimit()), i,1);

allocTable.setValueAt(newSort[i].getProcessName(), i, 2);
                                }
                                // 插入新生成的对象
                                newSort[runi] = jobToReady;


allocTable.setValueAt(String.valueOf(jobToReady.getBase()),runi, 0);

allocTable.setValueAt(String.valueOf(jobToReady.getLimit()),runi, 1);

allocTable.setValueAt(jobToReady.getProcessName(), runi, 2);
                                point++;
                            }
```

```
                    }
                }
                else {

                    readyDtoArray[end] = la;
                    readyTable.setValueAt(la.getProcessName(), end, 0);
                    readyTable.setValueAt(la.getRunTime(), end, 1);
                    readyTable.setValueAt(la.getPrority(), end, 2);
                    readyTable.setValueAt(la.getProcessState(), end, 3);
                    end = (end + 1) % 6;
                }
            }
        }
/*******************************新建按钮的触发事件*******************************/
        } else if (jb == newButton) {
            int newAllocFlag = -1;
            int newLength = 0;
            if (nameText.getText().trim().length() == 0) {
                JOptionPane.showMessageDialog(null, "进程名不能为空！");
            } else if (timeText.getText().trim().length() == 0) {
                JOptionPane.showMessageDialog(null, "运行时间不能为空");
            } else if (spaceText.getText().trim().length() == 0) {
                JOptionPane.showMessageDialog(null, "空间不能为空");
            } else if (Integer.valueOf(spaceText.getText().trim()) > bb) {
                JOptionPane.showMessageDialog(null, "该进程空间过大！");
            } else {//添加新进程test
                test.setRunTime(Integer.parseInt(timeText.getText()));
                test.setLimit(Integer.parseInt(spaceText.getText()));
                String s = prorityCom.getSelectedItem().toString();
                test.setPrority(Integer.parseInt(s));
                test.setProcessName(nameText.getText().trim());
                newDtoArray[count] = test;
                jobDtoVector.add(newDtoArray[count]);
                jobVectorName.add(newDtoArray[count].getProcessName());
                jobList.setListData(jobVectorName);
                count++;
                nameText.setText("");
                timeText.setText("");
                spaceText.setText("");
                PCB b = (PCB) jobDtoVector.elementAt(0);
                for (int i = 0; i < 20; i++) {
                    if (divDtoArray[i].getDivFalg() == 1) {
                        if (divDtoArray[i].getLength() >= b.getLimit()) {
                            newAllocFlag = i;
                            break;
```

```
                        }
                    }
                }
                if ((end+1) % 6 != first && newAllocFlag >= 0) {// 在就绪队列未满且内存有足够空间时
                    jobDtoVector.removeElementAt(0);
                    b.setProcessState(PCB.就绪态);//将后备队列jobDtoVetor中的对象添加到就绪队列中
                    b.setBase(divDtoArray[newAllocFlag].getDivBase());
                    newLength = divDtoArray[newAllocFlag].getLength()-
b.getLimit();
                    if (newLength == 0) {
                        int i = newAllocFlag;
                        divDtoArray[i].setDivFlag(0);
                        for (; i <= 19; i++) {
                            if (divDtoArray[i + 1].getDivFalg() == 1) {
                                divDtoArray[i] = divDtoArray[i + 1];
                                divDtoArray[i + 1].setDivFlag(0);

divTable.setValueAt(String.valueOf(divDtoArray[i].getDivBase()),i, 0);

divTable.setValueAt(String.valueOf(divDtoArray[i].getLength()),i, 1);
                            }
                        }
                    } else if (newLength > 0) {
                        int c1 = divDtoArray[newAllocFlag].getDivBase()+
b.getLimit();
                        divDtoArray[newAllocFlag].setDivBase(c1);
                        divDtoArray[newAllocFlag].setLength(newLength);
                        divTable.setValueAt(String.valueOf(c1),newAllocFlag, 0);

divTable.setValueAt(String.valueOf(newLength),newAllocFlag, 1);
                    }
                    readyDtoArray[end] = b;

jobVectorName.remove(jobVectorName.indexOf(b.getProcessName()));
                    readyTable.setValueAt(b.getProcessName(), end, 0);
                    readyTable.setValueAt(b.getRunTime(), end, 1);
                    readyTable.setValueAt(b.getPrority(), end, 2);
                    readyTable.setValueAt("就绪态", end, 3);//第四列默认值
                    end = (end + 1) % 6;
                    int newi = 0;// 用于记录当前新生成的PcbDTO对象应该插入到newSort中的位置

                    for (; newi < point; newi++) {
                        if (b.getBase() < newSort[newi].getBase()) {
```

```
                        break;
                    }
                }
                // 如果不是插入到数组末尾，则把比它大的都向后挪一位并设置JTable中的显示
                for (int i = point; i > newi; i--) {
                    newSort[i] = newSort[i - 1];

allocTable.setValueAt(String.valueOf(newSort[i].getBase()), i, 0);

allocTable.setValueAt(String.valueOf(newSort[i].getLimit()), i, 1);
                    allocTable.setValueAt(newSort[i].getProcessName(),i, 2);
                }
                // 插入新生成的对象
                newSort[newi] = b;
                allocTable.setValueAt(String.valueOf(b.getBase()),newi, 0);
                allocTable.setValueAt(String.valueOf(b.getLimit()),newi, 1);
                allocTable.setValueAt(b.getProcessName(), newi, 2);
                point++;
            }
        }
/*********************************挂起按钮的触发事件*********************************/
        } else if (jb == susButton) {
            if (readyTable.getSelectedRow() >= 0 && readyTable.getSelectedRow() <
6) {
                if
(!readyDtoArray[readyTable.getSelectedRow()].getProcessState().equals("就绪挂起"))
{

readyDtoArray[readyTable.getSelectedRow()].setProcessState(PCB.就绪挂起);
                    readyTable.setValueAt("就绪挂起",readyTable.getSelectedRow(),
3);

waitingDtoVector.add(readyDtoArray[readyTable.getSelectedRow()]);

waitingVectorName.add(readyDtoArray[readyTable.getSelectedRow()].getProcessName())
;
                    waitingList.setListData(waitingVectorName);
                } else {
                    System.out.println("已挂起");
                }
            } else {
                JOptionPane.showMessageDialog(null, "请先选择要挂起的进程");
            }
/*********************************解挂按钮的触发事件*********************************/
```

```
            } else if (jb == relaxButton) {
                String s = (String) waitingList.getSelectedValue();
                if (s != null) {
                    waitingVectorName.remove(s);
                    PCB p = new PCB();
                    for (int i = 0; i < waitingDtoVector.size(); i++) {
                        p = (PCB) waitingDtoVector.elementAt(i);
                        if (s.equals(p.getProcessName())) {
                            p.setProcessState(PCB.就绪态);
                            waitingDtoVector.remove(p);
                            break;
                        }
                    }
                    for (int i = 0; i < 6; i++) {
                        if (s.equals(readyDtoArray[i].getProcessName())) {
                            readyTable.setValueAt("就绪态", i, 3);//改为就绪态
                            break;
                        }
                    }
                    waitingList.setListData(waitingVectorName);
                } else {
                    JOptionPane.showMessageDialog(null, "请先选择要解挂的进程");
                }
            }
        }
    }
    public static void main(String args[]) {
        new Main();//调用运行程序
    }
}
```

五 上机编码和调试

调试操作如图所示：

图 5-1 初始图形界面

图 5-2 键入新建进程

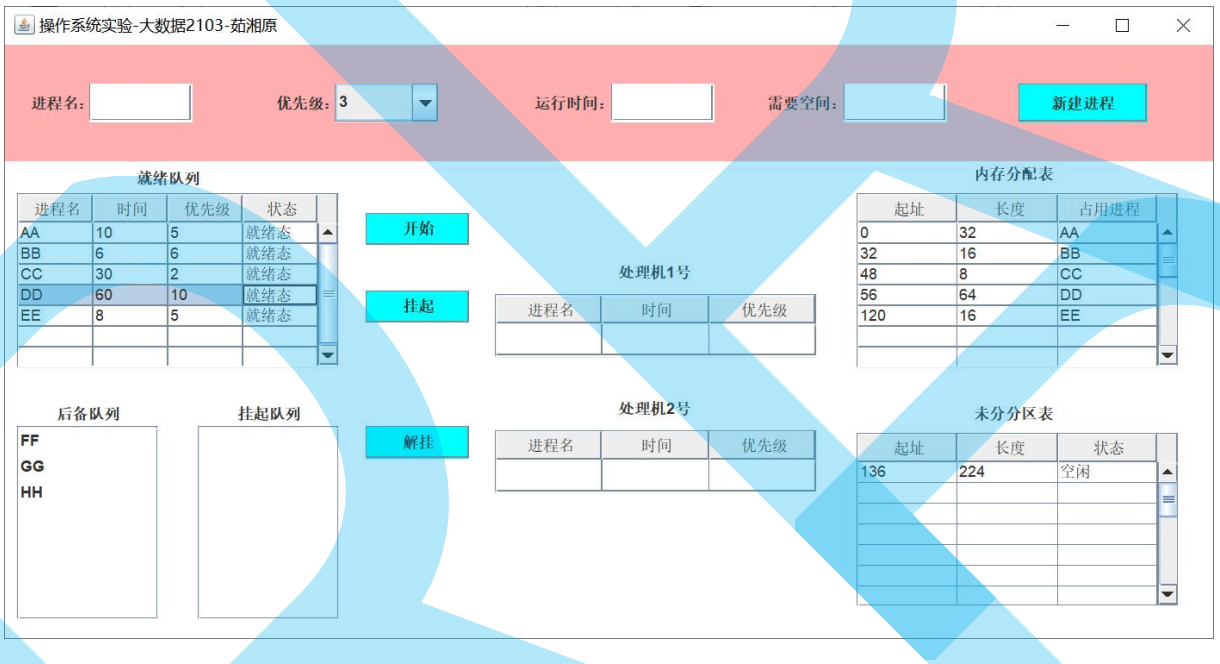

图 5-3 随机输入若干进程

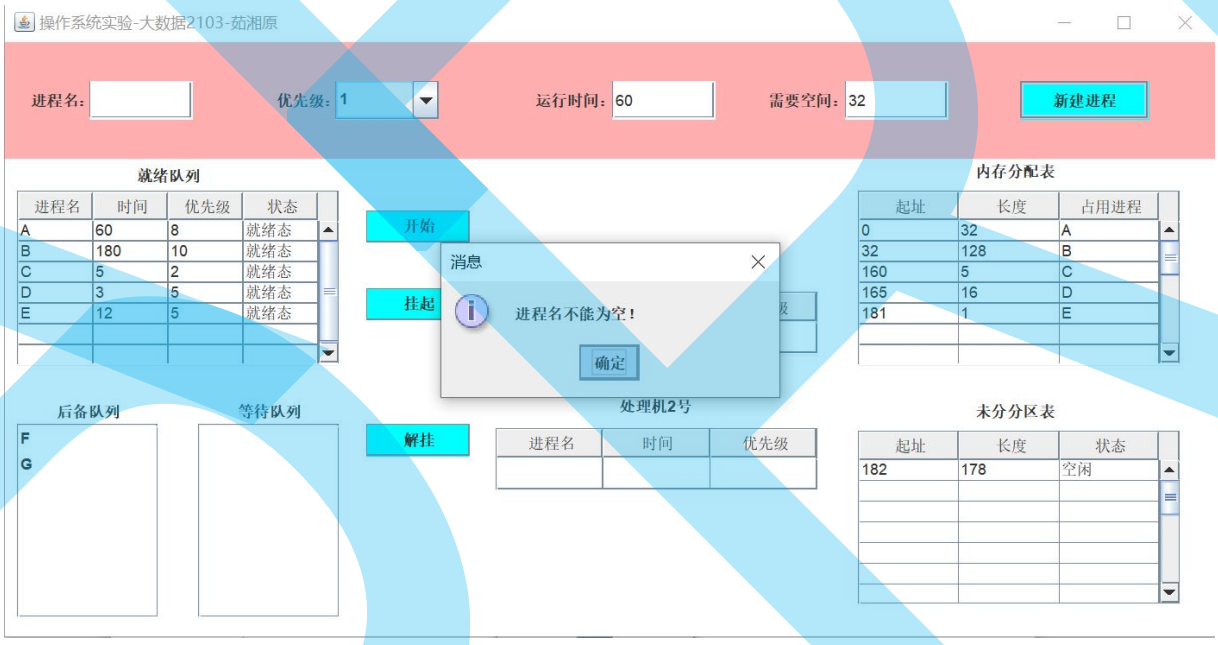

图 5-4 添加异常(1)

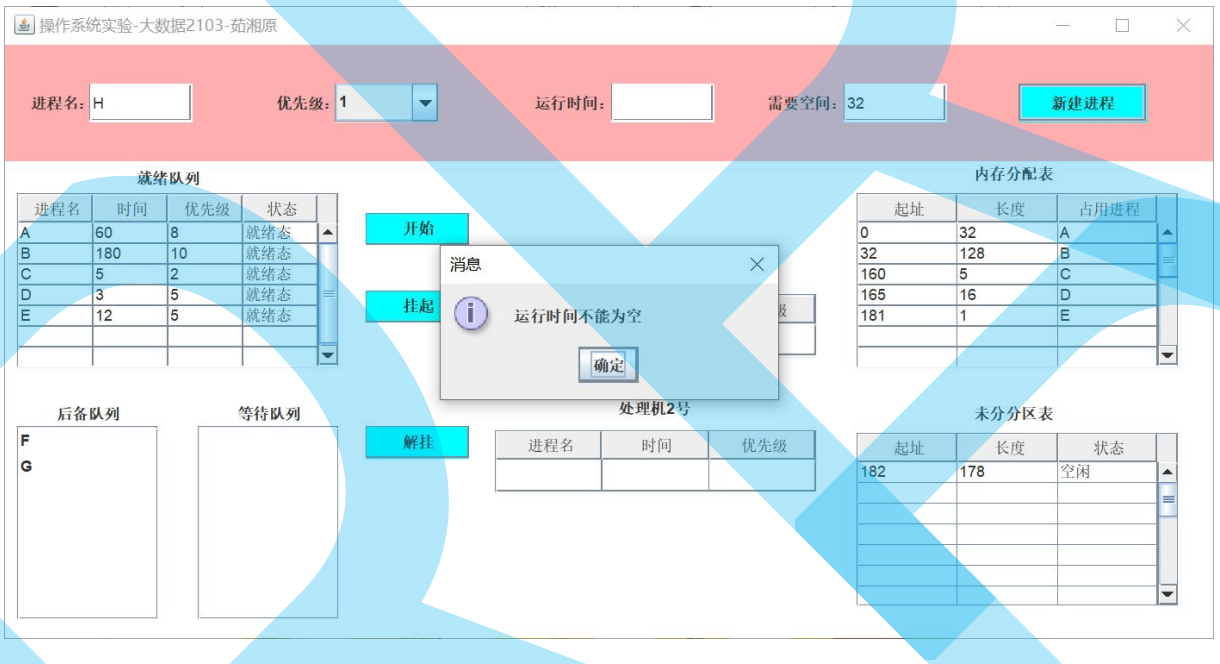

图 5-5 添加异常(2)

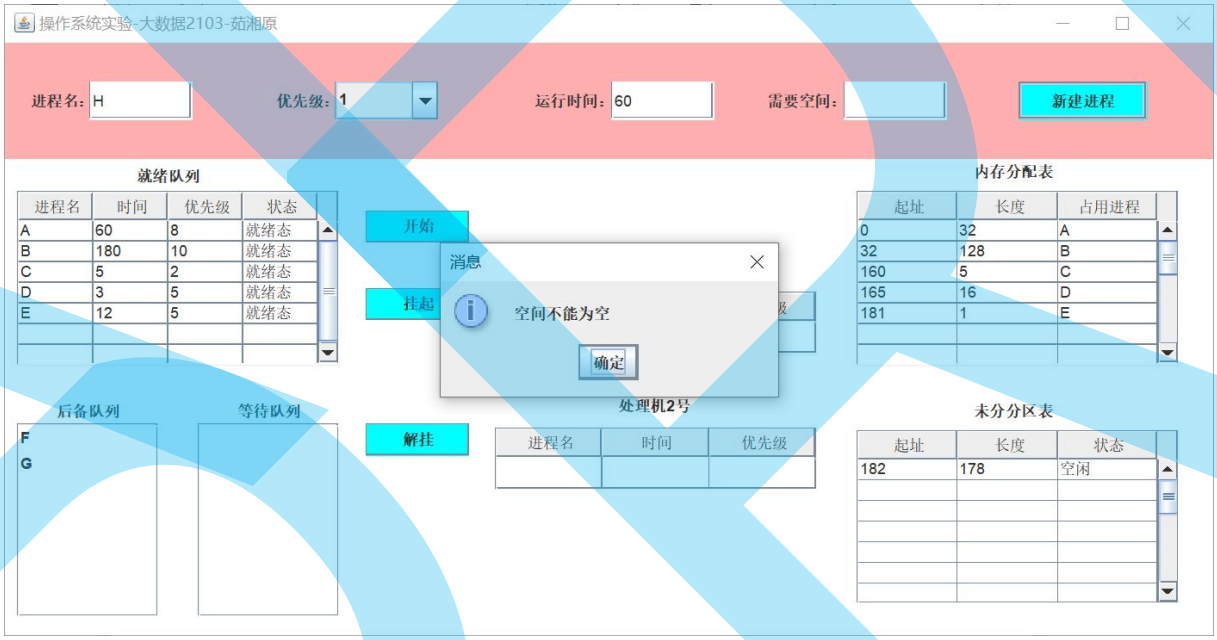

图 5-6 添加异常(3)

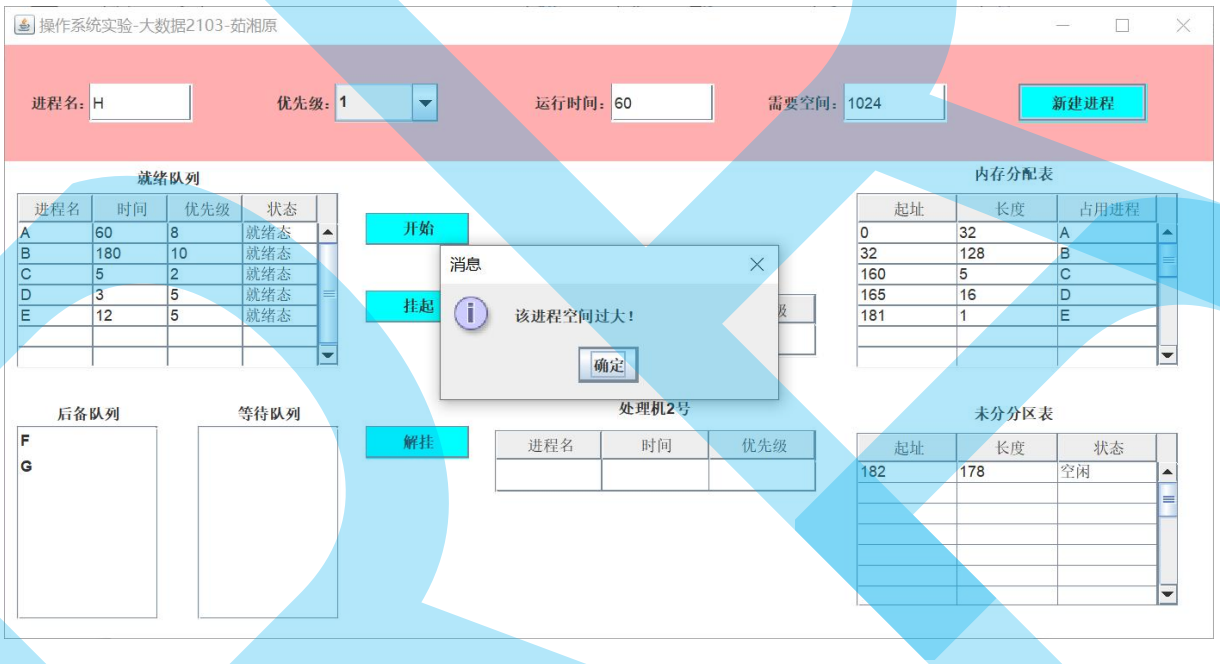

图 5-7 添加异常(4)

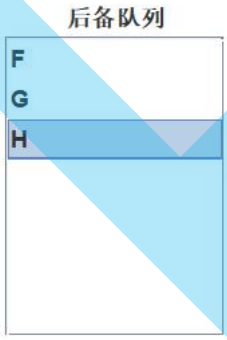

图 5-8 设置后备队列

图 5-9 设置挂起队列

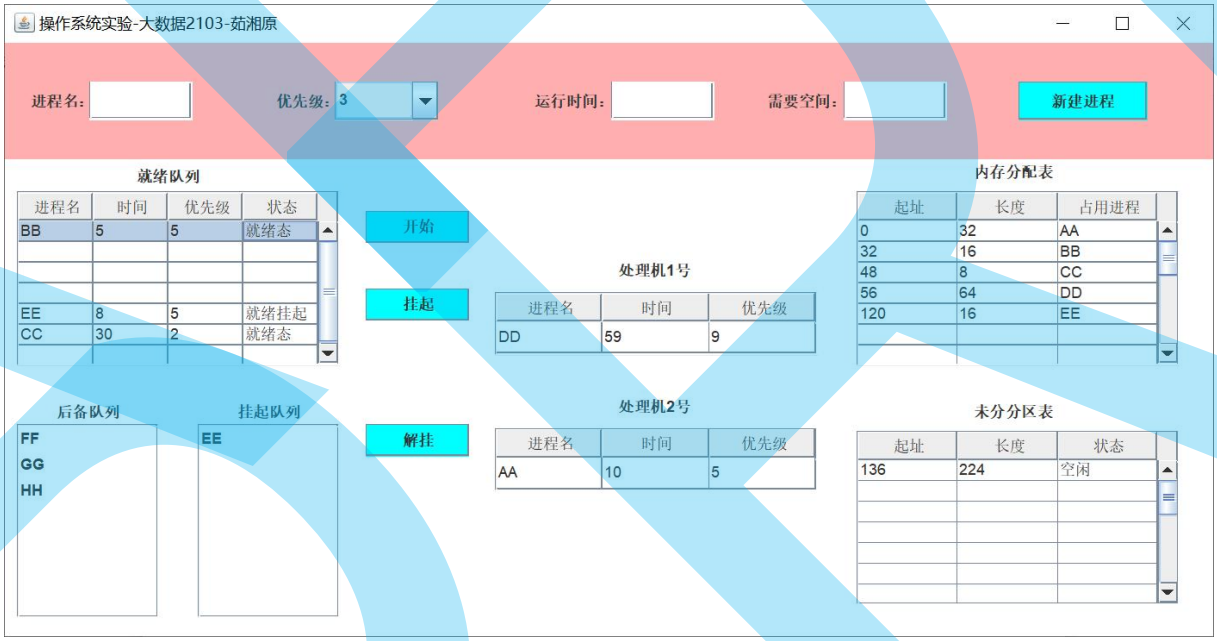

图 5-10 进程调度

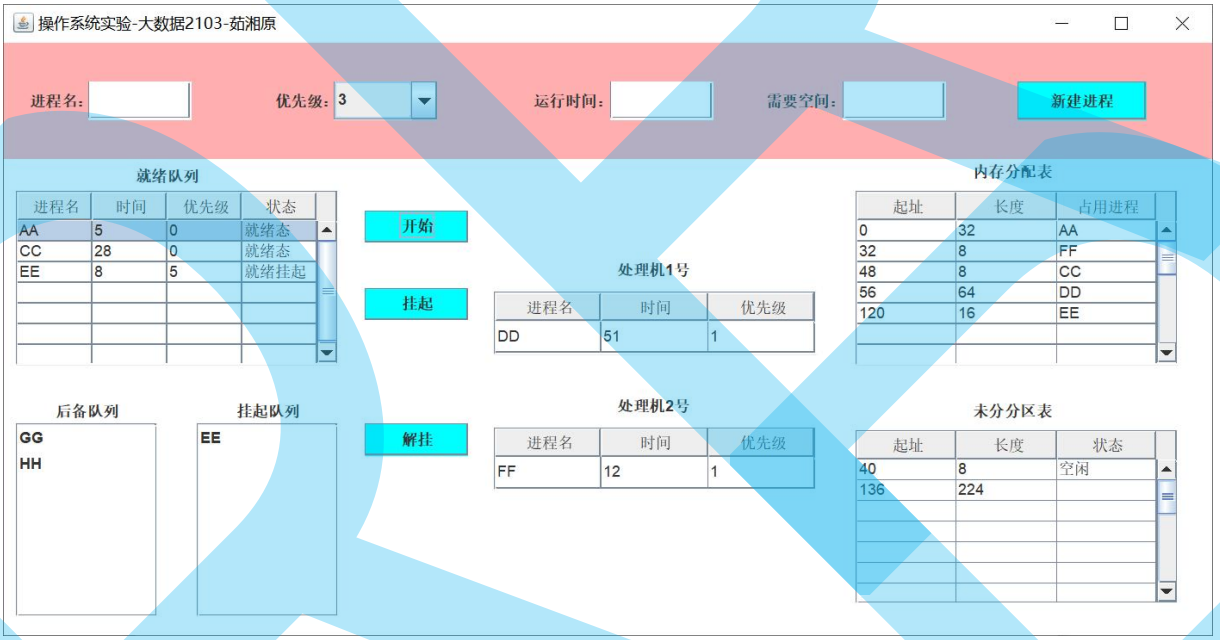

图 5-11 后备队列出队

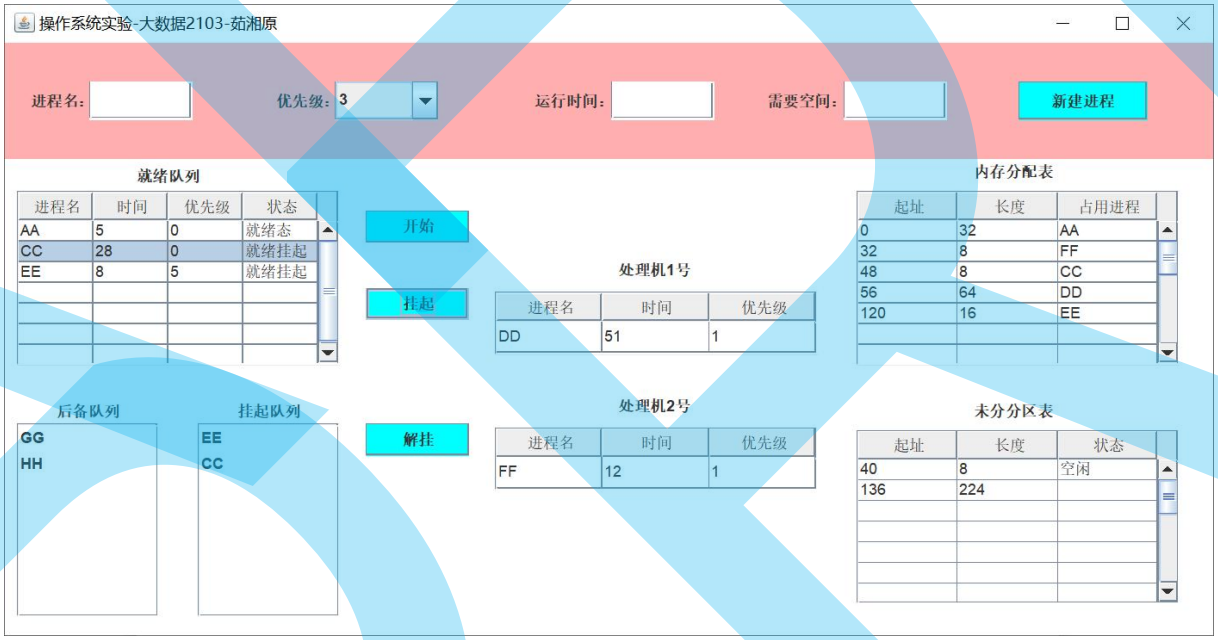

图 5-12 进程挂起

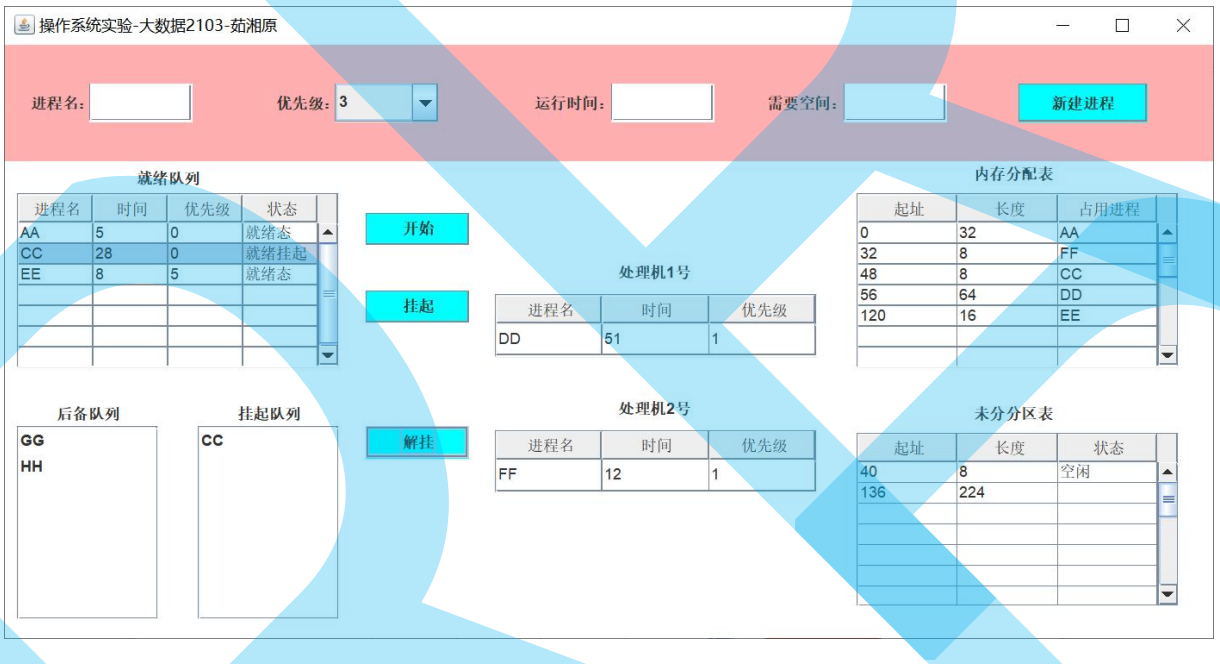

图 5-13 进程解挂

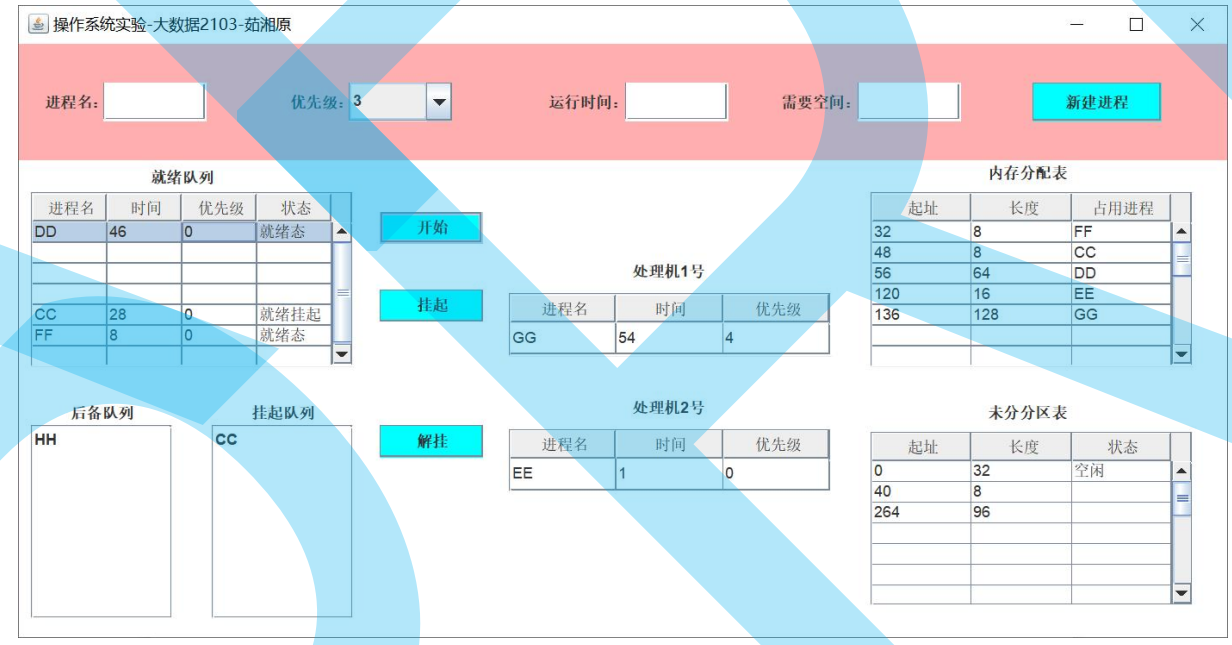

图 5-14 继续运行

内存分配表

起址	长度	占用进程
32	8	FF
48	8	CC
56	64	DD
120	16	EE
136	128	GG

未分分区表

起址	长度	状态
0	32	空闲
40	8	
264	96	

图 5-15 动态内存分配

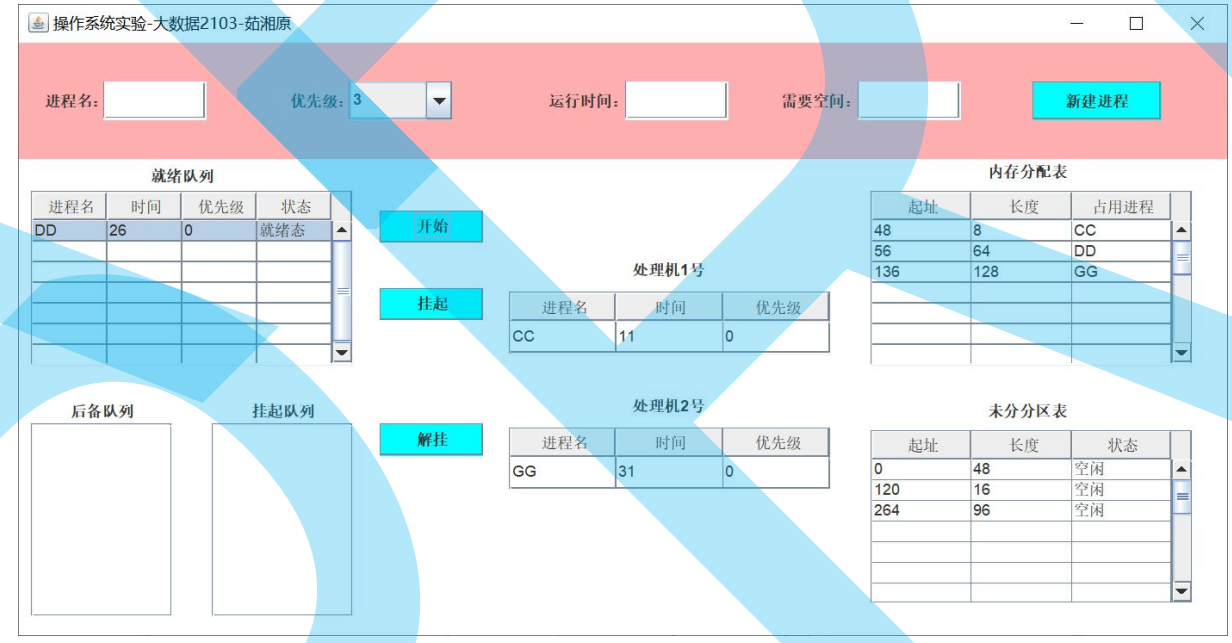

图 5-16 继续运行

内存分配表

起址	长度	占用进程

未分分区表

起址	长度	状态
0	360	空闲

图 5-17 相连空闲分区合并

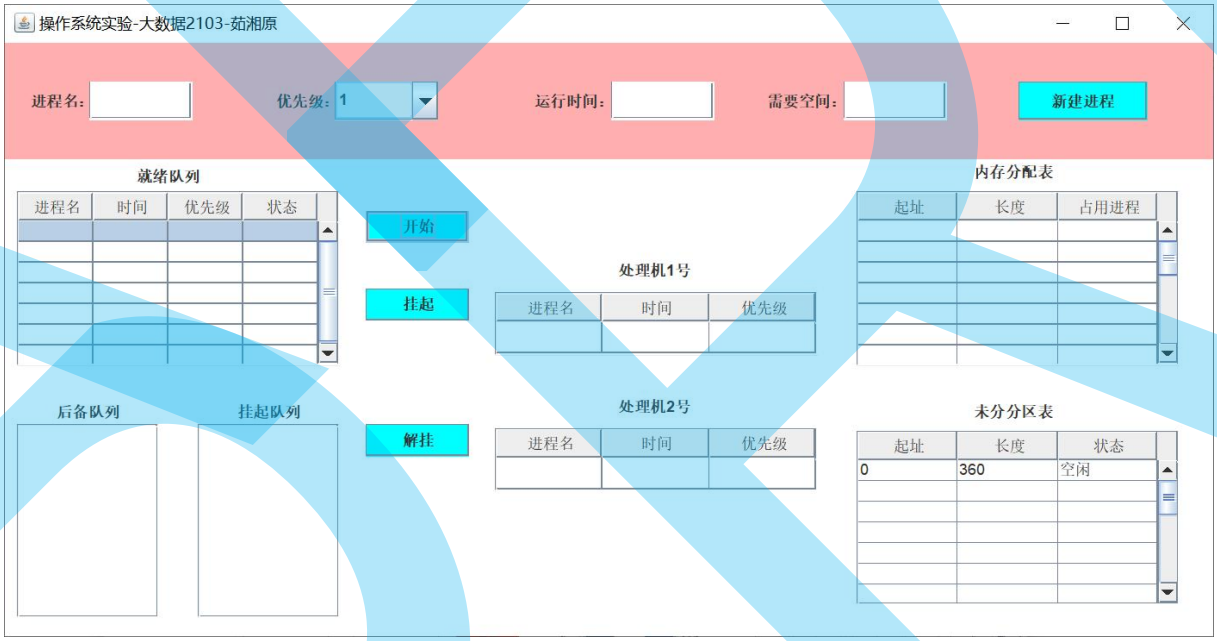

图 5-18 所有进程处理完成

结束语