
操作系统实验

第一期 工程开发实现

实验名称 EXP.1 CPU Scheduling

Exp.2 Allocation & Reclaim

实验内容

- 一， 选择一个调度算法，实现处理机调度；
- 二， 处理机调度过程中，主存储器空间的分配和回收；

实验目的

- 一,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,以加深理解处理机调度的工作;
- 二,帮助理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;

实验题目

- 一, (1) 设计一个按照优先权调度算法实现处理机调度的程序;
- (2) 设计按时间片轮转实现处理机调度的程序;
- 二, 在可变分区管理方式下,采用最先适应算法实现主存空间的分配和回收;

实验要求

- 一, (a), PCB 内容: 进程名/PID; 要求运行时间 (单位时间); 优先权; 状态; PCB 指针; —— (因课程内容原因, 这个指针在设计中没用)
- 1, 可随机输入假设干进程, 并按优先权排序;
- 2, 从就绪队列首选进程运行: 优先权-1/ 要求运行时间-1; 要求运行时间=0 时, 撤销该进程;
- 3, 重新排序, 进展下一轮调度;
- (b), 最好采用图形界面;

{c} ,可随时增加进程;

{d}, 规定道数, 设置后备队列和挂起状态。假设内存中进程数少于规定道数, 可自动从后备队列调度一作业进入。被挂起进程如=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;

{e}, 每次调度后, 显示各进程状态;

二, {a}, 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〕

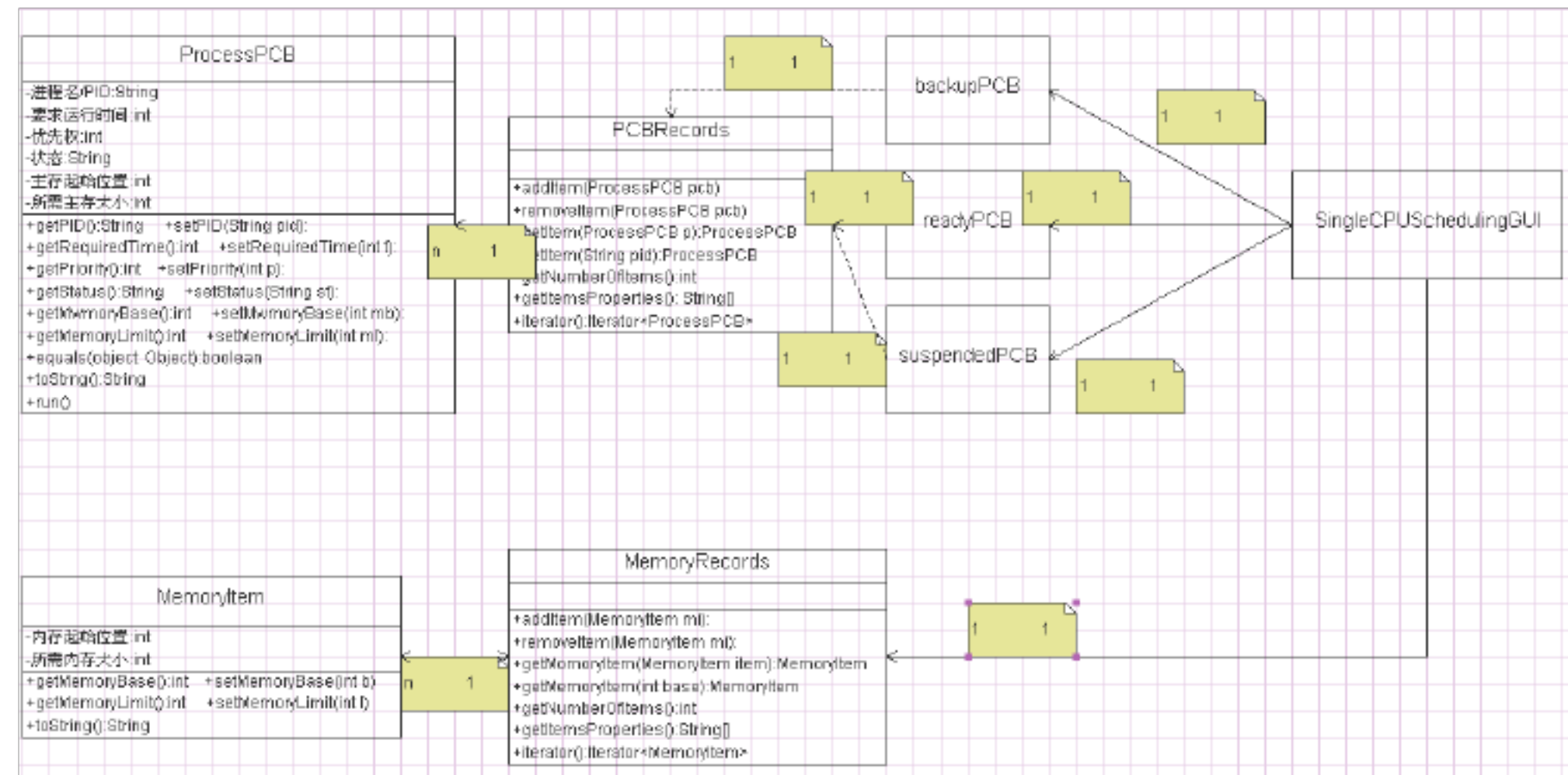
{b}, 结合实验一, PCB 增加为: {PID, 要求运行时间, 优先权, 状态, 所需内存大小, 主存起始位置, PCB 指针〔失效〕};

{C} 采用最先适应算法分配主存空间;

{D}, 进程完成后, 回收主存, 并与相邻空闲分区合并;

实验过程及分析

1, 初步设计:



2, 详细设计:

{a}, 操作系统知识回忆:

(1) 作业进入内存中, 由 CPU 分配产生 PCB 属性, 并通过 PCB 记录进程状态, 实验即以 PCB 代表进程模拟调度过程;

(2) 在多道系统中, 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，本实验采用优先级；

{3}，进程调度时，规定假设就绪队列进程数少于 6 个，那么自动从后备队列调入一个作业；

{4}，系统会将占有较多资源、预期结果不符合要求的进程自动挂起，并回收所占资源，而本实验设置为手动挂起；

{5}，在适宜条件下，系统会将挂起的进程自动解挂，而且只解挂到就绪队列；本实验为简化操作，设置为手动解挂，假设解挂条件适宜（即 CPU 各种资源可用），那么解挂到就绪队列，并分配内存；假设解挂条件不适宜，那么解挂至后备队列，但不分配内存（实际上这是不对的，因为作业进入内存，由 CPU 标记 PCB 后，不能撤销 PCB 再返回内存，除非该进程执行完毕，但本程序为表达解挂的意思，还是错误地设计为可以解挂到后备队列，读者需注意，这个功能可以在代码中注销，另外也希望有高手可以改进）；

(b)，实验程序设计：

{1}，本实验采用 java 语言编程，并实现 GUI 界面显示；

{2}，为表达 java 语言面对对象程序设计的特点，实验设计为 ProcessPCB、MemoryItem 类封装 PCB 和所分配的内存各自的属性与方法；用 ProcessRecords、MemoryRecords 类封装数组方法；用 SingleCPUScheduling 实现 GUI 界面显示；

{3}，ProcessPCB 类中，定义 PCB 的进程名、要求运行时间、优先级、状态、主存起始位置、所需内存大小这 6 个属性，并定义各属性的 get 和 set 方法，定义 equals 方法用于比照类的属性，定义 toString 方法得到类属性的字符串，定义 run 方法封装优先权-1/要求运行时间-1 的过程；MemoryItem 类中，定义可分分区表每一可分记录的主存起始位置、内存大小及其 get 和 set 方法，定义 toString 方法得到可在界面显示的字符串；

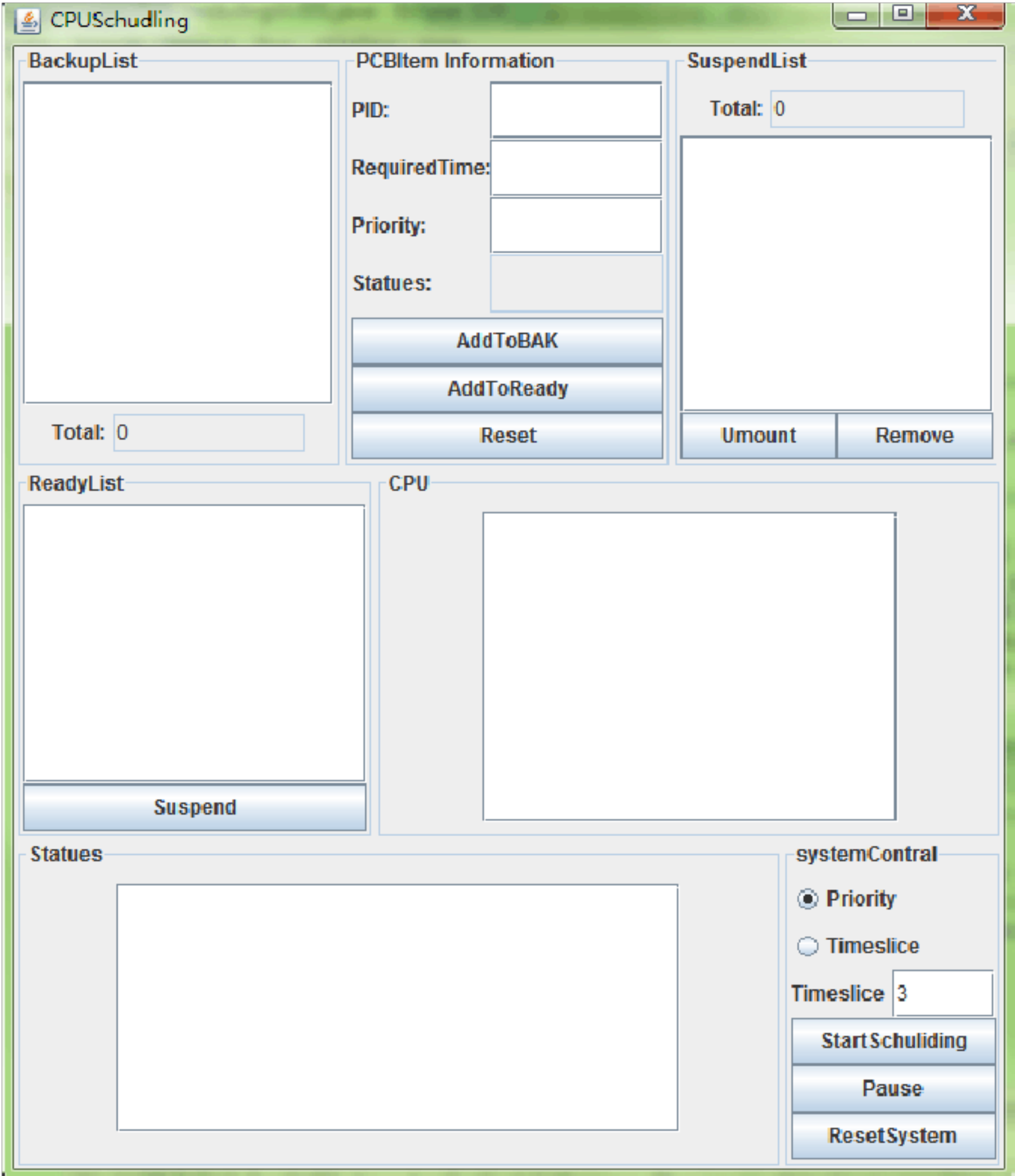
{4}，ProcessRecords 封装 PCB 数组的添加元素 addItem 和删除元素 removeItem 方法，并构造函数 getItem 通过参数 ProcessPCB 和 String 查找数组元素，定义 getNumberOfItems 取数组大小，定义 getItemsPriorities 方法取所有数组元素的 toString 方法用于界面显示，定义 iterator 方法获得数组的迭代器；

{5}，MemoryRecords 用同样的设计思想封装以 MemoryItem 为数组元素的各属性和方法；

{6} SingleCPUScheduling 类继承 JFrame 类，实现界面化显示；与上面相对应，实例化 ProcessRecords（3 次）和 MemoryRecords（1 次）作为私有变量，分别作为后

备队列、就绪队列、挂起队列和内存可分分区表；在界面设计中，设计后备队列、挂起队列〔附带解挂 **umount** 按钮〕、就绪队列〔附带挂起 **suspend** 按钮〕可分分区表列表显示框，设置 **PCB** 添加框，附带添加至后备队列〔**addToBackup** 〕、添加至就绪队列〔**addToReady** 〕按钮，以及 **CPU** 当前执行状态显示框、系统日志显示框，和开始调度〔**systemStart** 〕按钮，优先级和时间片单项选择按钮，以及时间片显示标签和文本编辑框；

〔7〕界面设计详解；后备队列显示框用于显示已添加至后备队列的 **ProcessRecords** 属性信息，其中主存起始位置默认为-1，表示未分配；挂起队列显示框用于显示从就绪队列挂起的 **PCB**，其中属性“主存起始位置”〔**MemoryBase** 〕将由非负数变为-1，表示挂起后收回内存；就绪队列显示框中显示就绪队列属性，其中“主存起始位置”均为非负，表示一分配内存；**PCB** 信息添加框分列 **PCB6** 个属性显示标签和可编辑文本框，



和添加按钮,用于添加 **PCB**；系统日志显示框附属时间片显示标签和可编辑文本编辑框,可由用户决定时间片大小；对于实验一,界面如下：

以上由 SingleCPUScheduling001.java

(8) 附属功能添加完善；最重要的是为程序添加线程，是程序能以停顿一段时间的频率自动运行；后备队列、挂起队列添加 total 显示标签和不可编辑文本显示框，用于显示各自数组中元素数目，挂起队列附属删除（remove）按钮，可删除挂起队列中的元素；后备、挂起、就绪队列均添加监听器，用于响应用户单击操作，可以在 PCB 信息添加框显示用户单击的那一条 PCB 的信息；PCB 信息添加框附属 reset 按钮，用于一键清空信息框中信息，方便输入；系统日志面板附属系统暂停（systemPause）和系统重置（systemReset）按钮，分别用于暂停运行（方便用户观察当前运行结果）和重置系统（方便用户重复使用程序，免去关闭后重启本程序的费事）；最终界面如图：

CPUSchudling

BackupList

Total: 0

PCBItem Information

PID:

RequiredTime:

Priority:

Statues:

Base:

Limit:

AddToBAK

AddToReady

Reset

SuspendList

Total: 0

Umount

Remove

ReadyList

Suspend

CPU

MemoryList

0_3000

Statues

systemControl

Priority

Timeslice

Timeslice

3

StartSchuliding

Pause

ResetSystem

实验结果报告级分析

- 1， 程序完成了实验所有的根本要求；
- 2， 本程序还存在一些技术上的问题，使得程序不能尽善尽美；如，PCB 信息添加框没有“随机置入就绪队列”功能，添加 PCB 信息仍显得繁琐；就绪队列的挂起功能在程序自动运行时，存在反响异常（反响延迟或直接无反映）；可分分区表只显示了当前可分的内存，没有显示已分的 PCB 及其对应内存使用情况，且没有利用

图形和丰富的颜色来更好的展示；时间片设计还需要改进，使用效率不高；系统重置功能存在响应延迟的问题；另外，界面不够美观；还需要不断改进；

实验感想

通过这次实验，我对操作系统的进程调度和内存分配管理有了更加深化的理解，对操作系统内部的工作原理有了进一步的认识；

通过编程，也稳固了我的程序设计和代码编写的才能，实验过程中遇到的各种问题以及解决问题的过程与方法，都是我获益匪浅；

同时，程序的不完善，也将促使我在课程之后，继续学习、理解课程内容，并尽一切努力不断完善程序，做到尽善尽美；

程序代码

完好版〔初学 java，菜鸟级别，当然是将所学的全部照办照抄，实为臃肿，可为初学者引以为戒，注意代码质量！〕

这里谨贴出非常臃肿的代码，仅供初学者交流经历，重在开发的思想，理解开发的流程，而 01 版〔精简版〕代码在后面；

ProcessPCB.java

```
package src;

public class ProcessPCB {
    // backupBAK 后备 ready 就绪 suspend 挂起 memory 内存
    private String PID ;
    private int RequiredTime ;
    // private String Priority;
    private int Priority ;
    private String Status ;
    private int MwmoryBase = 0000;
    private int MemoryLimit ;
    // private String PCBPointer;

    public ProcessPCB(String initpID, int initRTime, int initpriority,
        String status, int initBase, int initLimit) {

        this . PID = initpID;
        this . RequiredTime = initRTime;
        this . Priority = initpriority;
        this . Status = status;
        this . MwmoryBase = initBase;
```

```
        this . MemoryLimit    = initLimit;
    }

    public String getPID() {
        if (this .PID == null )
            return
        else
            return this .PID;
    }

    public void setPID(String pid) {
        if (pid == null )
            this
        else
            this .PID = pid;
    }

    public int getRequiredTime() {
        return this . RequiredTime ;
    }

    public void setRequiredTime( int time) {
        this . RequiredTime    = time;
    }

    public int getPriority() {
        return this . Priority ;
    }

    public void setPriority( int priority) {
        this . Priority    = priority;
    }

    public String getStatus() {
        if (this . Status == null )
            return ;
        else
            return this . Status ;
    }

    public void setStatues(String statues) {
        if (statues == null )
```

```

        this . Status      =      ;
    else
        this . Status      = statues;
    }

    public int  getMemoryBase() {
        return this . MwmoryBase ;
    }

    public void setMemoryBase( int  base) {
        this . MwmoryBase = base;
    }

    public int  getMemoryLimit() {
        return this . MemoryLimit ;
    }

    public void setMemoryLimit( int  limit) {
        this . MemoryLimit = limit;
    }

    public boolean equals(ProcessPCB pcb) {

        if (pcb.getPID() == this .getPID()) {
            return true ;
        }
        else return false ;
    }

    public String toString() {
        return this .getPID()      +      + this .getRequiredTime()      +      + this .getPriority()
+
        + this .getStatus() +      + this .getMemoryBase() +      +
this .getMemoryLimit() +      ;
    }

    public void run() {
        this . RequiredTime      = this . RequiredTime      -1;
        this . Priority          = this . Priority          -1;
    }

}

```

MemoryItem.java

```
public class MemoryItem {
    private int memoryBase =0;
    private int memoryLimit =0;
    private int availableStatus =0;

    public MemoryItem( int initMemoryBase, int initMemoryLimit) {
        this . memoryBase = initMemoryBase;
        this . memoryLimit = initMemoryLimit;
    }

    public int getMemoryBase() {
        return this . memoryBase ;
    }

    public void setMemoryBase( int base) {
        this . memoryBase = base;
    }

    public int getMemoryLimit() {
        return this . memoryLimit ;
    }

    public void setMemoryLimit( int limit) {
        this . memoryLimit = limit;
    }

    public int getStatus() {
        return this . availableStatus ;
    }

    public void setStatus( int status) {
        this . memoryBase = status;
    }

    public String toString() {
        return this .getMemoryBase() + " " + this .getMemoryLimit() + " " + this .getAvailableStatus();
    }
}
```

PCBRecords.java
package src;

```
public class PCBRecords implements Iterable<ProcessPCB> {

    private ArrayList<ProcessPCB> PCBItems;

    public ArrayList<ProcessPCB> getPCBItems() {
        return this.PCBItems;
    }

    public PCBRecords() {
        this.PCBItems = new ArrayList<ProcessPCB>();
    }

    public void addItem(ProcessPCB PcbItem) {

    }

    public void removeItem(ProcessPCB PCbItem) {

    }

    public ProcessPCB getItem(ProcessPCB processPCB) {
        for (ProcessPCB pCbItem : this.PCBItems) {
            if (pCbItem.equals(processPCB)) {
                return pCbItem;
            }
        }
        return null;
    }

    public ProcessPCB getItem(String pid) {
        for (ProcessPCB pcBItem : this.PCBItems) {
            if (pcBItem.getPID().equals(pid)) {
                return pcBItem;
            }
        }
        return null;
    }

    public int getNumberOfItems() {

    }

    public String[] getItemsProperties() {
        String itemsProperties[] = new String[getNumberOfItems()];

        int i = 0;
        for(Iterator iterator1 = PCBItems.iterator(); iterator1.hasNext();)
        {
            ProcessPCB stu_Item = (ProcessPCB)iterator1.next();
            itemsProperties[i++] = stu_Item.toString();
        }
        return itemsProperties;
    }

    public Iterator<ProcessPCB> iterator() {
```

```
}  
}
```

MemoryRecords.java
package src;

```
public class MemoryRecords implements Iterable<MemoryItem> {  
    private ArrayList<MemoryItem> memoryItems;  
  
    public Iterator<MemoryItem> iterator() {  
        // TODO Auto-generated method stub  
  
    }  
  
    public ArrayList<MemoryItem> getMemoryItems() {  
  
        return this.memoryItems;  
    }  
  
    public MemoryRecords() {  
        this.memoryItems = new ArrayList<MemoryItem>();  
    }  
  
    public void addItem(MemoryItem newMemoryItem) {  
  
    }  
    public void removeItem(MemoryItem momoryItem) {  
  
    }  
  
    public MemoryItem getMomoryItem(MemoryItem item) {  
        for(MemoryItem mItem : this.memoryItems) {  
            if(mItem.equals(item)) {  
                return mItem;  
            }  
        }  
        return null;  
    }  
    public MemoryItem getMemoryItem(int base) {  
        for(MemoryItem mItem : this.memoryItems) {  
            if(mItem.getMemoryBase() == base) {  
                return mItem;  
            }  
        }  
        return null;  
    }  
  
    public int getNumberOfItems() {  
  
    }  
  
    public String[] getItemsProperties() {  
        String itemsProperties[] = new String[getNumberOfItems()];  
        int i=0;
```

```

        MemoryItem mmItem = (MemoryItem) iterator1.next();
        itemsProperties[i++] = mmItem.toString();//
    }

    if(itemsProperties == null) {

    }
    return itemsProperties;
}
}

```

SingleCPUSchedulingGUI001.Java

```

public class SingleCPUSchedulingGUI001 extends JFrame {

    private int systemStatues;    //    static 属性
    /*define  0--system prepare  status--system reset and re-prepare  1--system start  2--system pause
3--system stop*/
    /* Standar error stream */
    static private PrintWriter    stderr = new    PrintWriter(System.err, true);

    static private int WIDTH = 600, HEIGHT = 700;                //the size of the Frame 主面板

    /* 各列表对应的面板规格*/
    /* 对应各名词释义 backupBAK 后备 ready 就绪 suspend 挂起 memory 内存 */
    static private int BackupBAK_CELL_SIZE = 250, BackupBAK_LIST_ROWS = 10;    //后备队列
    static private int Suspend_CELL_SIZE = 250, Suspend_LIST_ROWS = 10;        //挂起队列
    static private int Ready_CELL_SIZE = 200, Ready_LIST_ROWS = 6;            //就绪队列
    static private int CPU_ROWS = 10, CPU_COLS = 22;                        //CPU 面板
    static private int STATUS_ROWS = 8, STATUS_COLS = 30;                //系统状态面板
    private int timeslice = 1;                //设置时间片大小
    private int systemStatus=0;                //设置系统状态 0——系统预备状态,等待开始,1——系统运行状态,
2——系统暂停状态
    static private int TOTAL__TEXTFIELD_SIZE = 10;                // Size total text field 记录各队列元素个数

    private JList backupList, suspendList, readyList;    //各队列相对应的数组列表
    // 进程添加框中的 添加至后备队列 , 添加至就绪队列 , 重置

```

```

private JButton addToBAKButton, addToReadyButton, resetButton;
    //          挂起 , 挂起队列框中的 解挂 , 删除
private JButton suspendButton, umountButton, removeButton;
    //Status 面板中的 启动系统 , 重置系统
private JButton startButton, pauseButton, resetSyatemButton;
    //优先级和时间片单项选择钮及时间片显示框
private JRadioButton priorityJRB, timesliceJRB;
private JLabel timesliceSizeLabel;
private JTextField timesliceJtf;
    //后备面板、进程添加面板、挂起面板、内存面板
private JPanel backupBAKPanel, PCBItemPanel, suspendedPanel;
    //后备队列、挂起队列元素总数标签
private JLabel backupTotalLabel, suspendTotalLabel;
    //进程信息标签 进程编号 PID,所需运行时间 requiredTime, 优先级 priority, 当前状态 statues, 内存中的基
    址 base,所需内存大小 limit
private JLabel PIDLabel, requiredTimeLabel, priorityLabel, statuesLabel;
    //后备队列、挂起队列元素总数文本框〔不可编辑〕
private JTextField backupTotalTextField, suspendTotalTextField;
    //进程信息文本框 PID(可编辑),requiredTime(可编辑),priority(可编辑),status(不可编辑),base(不可编
    辑),limit(可编辑)
private JTextField PIDTextField, requiredTimeTextField, priorityTextField, statusTextField;
    //CPU 状态显示文本域(不可编辑), status 信息文本域(用于现实程序每一步的操作和影响, 不可
    编辑)
private JTextArea CPUTextArea, statuesTextArea;
    //后备队列 PCB 数组,就绪、挂起, ——内存〔可分分区表〕
PCBRecords backupPCB, readyPCB, suspendedPCB;

public static void main(String[] args) throws IOException {
    // TODO Auto-generated method stub
    new SingleCPUSchedulingGUI001().initFrame();
}

public void initFrame() {

    backupList = new JList();
    backupList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    backupList.setVisibleRowCount(BackupBAK_LIST_ROWS);
    backupList.setFixedCellWidth(BackupBAK_CELL_SIZE);
    suspendList = new JList();
    suspendList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    suspendList.setVisibleRowCount(Suspend_LIST_ROWS);
    suspendList.setFixedCellWidth(Suspend_CELL_SIZE);
    readyList = new JList();
    readyList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    readyList.setVisibleRowCount(Ready_LIST_ROWS);
    readyList.setFixedCellWidth(Ready_CELL_SIZE);

```

```
backupTotalTextField.setEditable(false);

suspendTotalTextField.setEditable(false);

timesliceJtf.setEditable(true);

CPUTextArea = new JTextArea(CPU_ROWS, CPU_COLS);
CPUTextArea.setEditable(false);
statuesTextArea = new JTextArea(STATUS_ROWS, STATUS_COLS);
statuesTextArea.setEditable(false);

/* north panel*/
// JPanel northPanel = new JPanel(new BorderLayout());
JPanel northPanel = new JPanel(new GridLayout(1, 3));
//
JPanel north = new JPanel(new BorderLayout());

    // ProcessPCB item information Panel
PCBItemPanel = new JPanel(new BorderLayout());
PCBItemPanel.setBorder(

JPanel PCBItemButtonJPanel = new JPanel(new GridLayout(3, 1));
PCBItemButtonJPanel.add(addToBAKButton);
PCBItemButtonJPanel.add(addToReadyButton);
PCBItemButtonJPanel.add(resetButton);

PCBItemPanel.add(this.initPCBItemPanel(), BorderLayout.CENTER);
PCBItemPanel.add(PCBItemButtonJPanel, BorderLayout.SOUTH);

    //backupBAKList Panel
backupBAKPanel = new JPanel(new BorderLayout());

JPanel backupTotalPanel = new JPanel();
backupTotalPanel.add(backupTotalLabel);
backupTotalPanel.add(backupTotalTextField);
backupBAKPanel.add (
    new JScrollPane(backupList,
        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED),
BorderLayout.CENTER);
    backupBAKPanel.add(backupTotalPanel, BorderLayout.SOUTH);

//
//    north.add(backupBAKPanel, BorderLayout.WEST);
//    north.add(PCBItemPanel, BorderLayout.CENTER);

    // SuspendList Panel
suspendedPanel = new JPanel(new BorderLayout());

JPanel suspendedTotalPanel = new JPanel();
suspendedTotalPanel.add(suspendTotalLabel);
suspendedTotalPanel.add(suspendTotalTextField);
```

```

JPanel suspendComponentPanel = new JPanel(new GridLayout(1, 2));
suspendComponentPanel.add(umountButton);
suspendComponentPanel.add(removeButton);

suspendedPanel.add (suspendedTotalPanel, BorderLayout.NORTH);
suspendedPanel.add (
    new JScrollPane(suspendList,
        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED),
BorderLayout.CENTER);
suspendedPanel.add(suspendComponentPanel, BorderLayout.SOUTH);

//    northPanel.add(north, BorderLayout.CENTER);
//    northPanel.add(suspendedPanel, BorderLayout.EAST);
northPanel.add(backupBAKPanel);
northPanel.add(PCBItemPanel);
northPanel.add(suspendedPanel);

/* center Panel*/
JPanel centrePanel = new JPanel(new BorderLayout());
//    JPanel centrePanel = new JPanel(new GridLayout(1, 3));

    // readyList panel
JPanel readyListPanel = new JPanel(new BorderLayout());

readyListPanel.add (
    new JScrollPane(readyList,
        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED));//,
BorderLayout.CENTER
readyListPanel.add (suspendButton, BorderLayout.SOUTH);
    // CPU panel
JPanel CPUPanel = new JPanel();

CPUPanel.add (new JScrollPane(CPUTextArea,
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED));

centrePanel.add(readyListPanel, BorderLayout.WEST);
centrePanel.add(CPUPanel, BorderLayout.CENTER);

/*statues panel*/
JPanel southPanel = new JPanel(new BorderLayout());

JPanel statuesPanel = new JPanel();

statuesPanel.add (new JScrollPane(statuesTextArea,
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED)); //, BorderLayout.CENTER
//statuesPanel.setSize(400, 100);
//statuesPanel.setPreferredSize(new Dimension(400, 10));

JPanel systemContralButtonPanel = new JPanel(new GridLayout(6, 1));

ButtonGroup group = new ButtonGroup();
group.add(priorityJRB);
group.add(timesliceJRB);

```

```

JPanel porityPanel = new JPanel(new GridLayout(1, 2));
porityPanel.add(timesliceSizeLabel);
porityPanel.add(timesliceJtf);
systemContralButtonPanel.add(priorityJRB);
systemContralButtonPanel.add(timesliceJRB);
systemContralButtonPanel.add(porityPanel);
systemContralButtonPanel.add(startButton);
systemContralButtonPanel.add(pauseButton);
systemContralButtonPanel.add(resetSyatemButton);

southPanel.add(statuesPanel, BorderLayout.CENTER);
southPanel.add(systemContralButtonPanel, BorderLayout.EAST);

//      arrange panels in window
setLayout(new BorderLayout());
add(northPanel, BorderLayout.NORTH);
add(centrelPanel, BorderLayout.CENTER);
add(southPanel, BorderLayout.SOUTH);      //statuesPanel

//      start listening for list and buttons events
addToBAKButton.addActionListener(new AddToBAKListener());
addToReadyButton.addActionListener(new AddToReadyListener());
resetButton.addActionListener(new ResetListener());
suspendButton.addActionListener(new SuspendListener());
umountButton.addActionListener(new UmountListener());
removeButton.addActionListener(new RemoveListener());
startButton.addActionListener(new StartSystemListener());
pauseButton.addActionListener(new SystemPauseListener());
resetSyatemButton.addActionListener(new ResetSystemListener());
priorityJRB.addActionListener(new priotiryListener());
timesliceJRB.addActionListener(new timeslicListener());

backupPCB = new PCBRecords();
readyPCB = new PCBRecords();
suspendedPCB = new PCBRecords();

backupList.setListData(backupPCB.getItemsProperties());
readyList.setListData(readyPCB.getItemsProperties());
suspendList.setListData(suspendedPCB.getItemsProperties());

this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setSize(WIDTH, HEIGHT);
this.setResizable(true);
this.setVisible(true);
this.setLocation(200, 10);
}

public JPanel initPCBItemPanel() {

JPanel iniPCBItemJPanel = new JPanel(new BorderLayout());
JPanel iniNamePanel = new JPanel(new GridLayout(4, 1));
JPanel iniValuePanel = new JPanel(new GridLayout(4, 1));

```

```
        iniNamePanel.add(PIDLabel);
        iniNamePanel.add(requiredTimeLabel);
        iniNamePanel.add(priorityLabel);
        iniNamePanel.add(statusesLabel);

        PIDTextField.setEditable(true);

        requiredTimeTextField.setEditable(true);

        priorityTextField.setEditable(true);

        statusTextField.setEditable(false);
        iniValuePanel.add(PIDTextField);
        iniValuePanel.add(requiredTimeTextField);
        iniValuePanel.add(priorityTextField);
        iniValuePanel.add(statusTextField);

        iniPCBItemJPanel.add(iniNamePanel, BorderLayout.WEST);
        iniPCBItemJPanel.add(iniValuePanel, BorderLayout.CENTER);
        return iniPCBItemJPanel;
    }

    public int readInteger(String s) {
        int num = -1;
        try {
            num = Integer.parseInt(s);
        } catch (NumberFormatException numberFormatException) {

            num = -999;
        }
        return num;
    }

    public void addToBackupList(String newID, int s1, int s2, String s) {
        ProcessPCB item = backupPCB.getItem(newID);
        if(item != null) {

            while(item != null) {
                newID = s + newID;
                item = backupPCB.getItem(newID);
            }

            backupPCB.addItem(newPCB);
            backupList.setListData(backupPCB.getItemsProperties());
            backupTotalTextField.setText(Integer.toString(backupPCB.getNumberOfItems()));
        }

    }

    public void addToReadyList(String nowID, int s1, int s2, String s) {
        ProcessPCB item = readyPCB.getItem(nowID);
        if(item != null) {

            while(item != null) {
                nowID = s + nowID;
                item = backupPCB.getItem(nowID);
            }

        }
    }
}
```

```

    }
}

readyPCB.addItem(newPCB);
sortReadyPCB();
readyList.setListData(readyPCB.getItemsProperties());
}

class AddToBAKListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        String newID = PIDTextField.getText();
        String newTime = requiredTimeTextField.getText();
        String newPriority = priorityTextField.getText();
        int s1 = 0, s2 = 0, tag1=-1, tag2=-1;

        if(newTime != null)
        {
            s1 = readInteger(newTime);
            if(s1 > 0.0) tag1 = 1;
            else
            }
        if(newPriority != null)
        {
            s2 = readInteger(newPriority);
            if(s1 != -999) tag2 = 1;

        }
        if(tag1 ==1 && tag2 == 1 ) {
            if(newID == null) {

                } else {

                    reset();

                }
            }
        }
    }
}

```

```

class AddToReadyListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {

        String nowID = PIDTextField.getText();
        String time = requiredTimeTextField.getText();
        String priority = priorityTextField.getText();
        int s1 = 0, s2 = 0;
        int tag1 = -1, tag2 = -1;
        if(time != null)
        {
            s1 = readInteger(time);
            if(s1 > 0.0) tag1=1;

        }
        if(priority != null)
        {
            s2 = readInteger(priority);
            if(s2 != -999) tag2=1;

```

```

    }
    if(tag1 ==1 && tag2 == 1) {
        if(nowID == null) {

            } else {
                if(readyPCB.getNumberofItems() < 6) {

                    student record item has been added to

                } else {

                    student record item has been added to

                }
            }
        }
    }
}

public void reset() {

}

/**
 * This inner class processes <code>resetButton</code> events.
 */
class ResetListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        reset();
    }
}

/**
 * This inner class processes <code>suspendButton</code> events.
 */
//注：在挂起时，不会触发进程调度，而是在点击
class SuspendListener implements ActionListener {

    /**
     * Removes an order item from the current order.
     *
     * @param event    the event object.
     */
    public void actionPerformed(ActionEvent event) {

        String selectedReadyItem = null;

        if(readyPCB.getNumberofItems() == 0)
        {

        } else {
            selectedReadyItem = (String) readyList.getSelectedValue();

```

```

        if(selectedReadyItem == null)    {

        } else
        {

            pid = stringtokenizer.nextToken();
            ProcessPCB selectedItem = readyPCB.getItem(pid);
            if(selectedItem == null)    {

                student recorditem of Num      + pid      +      is

            } else    {
                ProcessPCB boolTtem = suspendedPCB.getItem(pid);
                if(boolTtem != null) {

                    PCB      + pid      +      has existed in Suspend

                }

                while(boolTtem != null) {

                    boolTtem = suspendedPCB.getItem(pid);

                }

                ProcessPCB newPcb = new ProcessPCB(pid, selectedItem.getRequiredTime(),

                suspendedPCB.addItem(newPcb);
                readyPCB.removeItem(selectedItem);
                sortReadyPCB();    //注意考虑一下
                suspendList.setListData(suspendedPCB.getItemsProperties());
                readyList.setListData(readyPCB.getItemsProperties());

suspendTotalTextField.setText(Integer.toString(suspendedPCB.getNumberOfItems()));
            }
        }
    }
}

public void sortReadyPCB() {
    PCBRecords currentReadyPCB = new PCBRecords();
    int num = readyPCB.getNumberOfItems();
    if(num > 0) {
        for(int i=num; i>=1; i--) {
            Iterator readyIterator = readyPCB.iterator();
            ProcessPCB currentItem = (ProcessPCB) readyIterator.next();
            for( ; readyIterator.hasNext(); ) {
                ProcessPCB nowItem = (ProcessPCB) readyIterator.next();
                if(currentItem.getPriority() < nowItem.getPriority()) {
                    currentItem = null;
                    currentItem = nowItem;
                }
            }
            currentReadyPCB.addItem(currentItem);
            readyPCB.removeItem(currentItem);
        }
        readyPCB = null;
        readyPCB = currentReadyPCB;
    }
}

```

```

/**
 * This inner class processes <code>umountButton</code> events.
 */
class UmountListener implements ActionListener {

    /**
     * Removes an order item from the current order.
     *
     * @param event    the event object.
     */
    public void actionPerformed(ActionEvent event) {

        String selectedSuspendedItem = null;

        if(suspendedPCB.getNumberOfItems() == 0)
        {

        } else {
            selectedSuspendedItem = (String) suspendList.getSelectedValue();
            if(selectedSuspendedItem == null) {

            } else
            {

                id = stringtokenizer.nextToken();
                ProcessPCB selectedItem = suspendedPCB.getItem(id);
                if(selectedItem == null) {

                    student recorditem of Num + id + is

                } else {
                    if(readyPCB.getNumberOfItems() < 6) {

                        ProcessPCB selectedToItem = readyPCB.getItem(id);
                        if(selectedToItem != null) {

                            while(selectedToItem != null) {

                                selectedToItem = readyPCB.getItem(id);
                            }
                        }
                        ProcessPCB newToItemPCB = new ProcessPCB(id,

                        readyPCB.addItem(newToItemPCB);
                        suspendedPCB.removeItem(selectedItem);
                        suspendList.setListData(suspendedPCB.getItemsProperties());

suspendTotalTextField.setText(Integer.toString(suspendedPCB.getNumberOfItems()));
                        sortReadyPCB();    /注意考虑
                        readyList.setListData(readyPCB.getItemsProperties());
                                                product has been Umounted to Ready

                    } else {

                        The new ProcessPCB will

                        ProcessPCB selectedToItem = backupPCB.getItem(id);
                        if(selectedToItem != null) {

```

```

        while(selectedToItem != null) {

            selectedToItem = backupPCB.getItem(id);
        }
    }
    ProcessPCB newToItemPCB = new ProcessPCB(id,

    backupPCB.addItem(newToItemPCB);
    suspendedPCB.removeItem(selectedItem);
    suspendList.setListData(suspendedPCB.getItemsProperties());
    backupList.setListData(backupPCB.getItemsProperties());

suspendTotalTextField.setText(Integer.toString(suspendedPCB.getNumberOfItems()));
                                product has been Umounted to Backup
    }
    }
    }
}
/**
 * This inner class processes <code>removeButton</code> events.
 */

class RemoveListener implements ActionListener {

    public void actionPerformed(ActionEvent event) {

        String selectedRemovsItem = null;

        if(suspendedPCB.getNumberOfItems() == 0)
        {

        } else
        {
            selectedRemovsItem = (String) suspendList.getSelectedValue();
            if(selectedRemovsItem == null) {

            } else {

                id = stringtokenizer.nextToken();
                ProcessPCB suspendedItem = suspendedPCB.getItem(id);
                if(suspendedItem == null) {
                    student recorditem of Num + id + is
                } else {

                    suspendedPCB.removeItem(suspendedItem);
                    suspendList.setListData(suspendedPCB.getItemsProperties());

suspendTotalTextField.setText(Integer.toString(suspendedPCB.getNumberOfItems()));
                }
            }
        }
    }
}

```

```

    }
}

public void PCBSchudling() {
    while(readyPCB.getNumberOfItems() < 6) {
        if(backupPCB.getNumberOfItems() > 0) {
            Iterator bwakeupPCBIterator = backupPCB.iterator();
            ProcessPCB newItem = (ProcessPCB) bwakeupPCBIterator.next();
            readyPCB.addItem(newItem);
            backupPCB.removeItem(newItem);
            //注意考虑要不要排序
            sortReadyPCB();
            readyList.setListData(readyPCB.getItemsProperties());
            backupList.setListData(backupPCB.getItemsProperties());
            backupTotalTextField.setText(Integer.toString(backupPCB.getNumberOfItems()));
        } else break;
    }
}

public void run() {

    PCBSchudling();
    if(readyPCB.getNumberOfItems() > 0) {
        Iterator readyIterator = readyPCB.iterator();
        ProcessPCB runningItem = (ProcessPCB) readyIterator.next();
        if(runningItem.getRequiredTime()>0) {
            runningItem.run();

        }
        if(runningItem.getRequiredTime() <= 0) {

            readyPCB.removeItem(runningItem);
            runningItem = null;
            //注意
            PCBSchudling();
        }
        sortReadyPCB();
        readyList.setListData(readyPCB.getItemsProperties());
    } else {
        this.systemStatues = 0;
        System.exit(0);
    }
}

boolean getSystemStatues() {
    if(this.systemStatues == 1)
        return true;
    else return false;
}

class StartSystemListener implements ActionListener {

    public void actionPerformed(ActionEvent event) {
        systemStatues = 1;
        run();
    }
}

```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/886055213102010103>