



中南大學  
CENTRAL SOUTH UNIVERSITY

# 操作系统原理

## 实验报告

学生姓名	王瑛
学 号	8208190406
专业班级	计科 1902
指导教师	沈海澜
学 院	计算机学院
完成时间	2021.6.20

# 目录

一、实验目的 .....	1
二、实验内容与要求 .....	1
三、实验分析与设计 .....	1
一、实验分析 .....	1
二、总体设计思路 .....	2
三、算法流程图 .....	4
四、结构与相关代码 .....	5
(一) 数据结构设计 .....	5
(二) 添加进程模块(函数)的设计与实现 .....	6
(三) 调入进程模块(函数)的设计与实现 .....	7
(四) 执行进程模块(函数)的设计与实现 .....	8
(五) 挂起进程模块(函数)的设计与实现 .....	10
(六) 解挂进程模块(函数)的设计与实现 .....	10
(七) 阻塞进程模块(函数)的设计与实现 .....	11
(八) 唤醒进程模块(函数)的设计与实现 .....	11
五、实验运行结果 .....	12
(1) 图形界面 .....	12
(2) 添加进程 .....	13
(3) 申请内存, 调度执行 .....	13
(4) 挂起进程 .....	14
(5) 进程阻塞 .....	14
(6) 解挂进程 .....	15
(7) 结果 .....	15
(八) 具体代码: .....	15
① Dispatch.java: .....	15
② Memory.java .....	20
③ Node.java .....	20
④ PCB.java .....	21
⑤ OS.java .....	21
六、心得体会 .....	28
七、参考文献 .....	29

## 一、实验目的

- 1、掌握进程的创建、阻塞、唤醒、撤销等进程控制原语。
- 2、掌握进程的处理机调度的过程和算法。
- 3、掌握内存分配策略。

## 二、实验内容与要求

设计一个程序，模拟 OS 进程管理与内存分配。要求程序运行后，有一个模拟真实场景下的交互终端的界面，负责提交用户作业和进程状态信息(类似于 LINUX 下的 PS 命令)；另外后台程序模拟 OS，完成进程管理和内存分配。OS 部分主要包括：实现创建新进程原语、阻塞进程原语、唤醒进程原语，终止进程原语、调度进程原语、内存分配等原语。进程创建时需为进程分配内存空间，进程终止时需要回收进程的内存空间。

## 三、实验分析与设计

### 一、实验分析

该模拟系统采用 java 语言实现，要实现的功能有新建进程、进程调度、挂起进程、解挂进程、删除进程，道数和时间片大小可以由用户自己调整，有两种调度策略：按优先权调度和按时间片轮转调度。

每个进程可能有 5 种状态：新建(new)、就绪(ready)、运行(running)、阻塞(waiting)、挂起(suspend)。每个状态都有一个队列用来存放处于该状态的进程，不同的调度策略采用不同的队列实现。

当创建进程时，如果内存中的进程数还没达到规定道数，则将新建进程插入就绪队列，如果内存中进程数已经达到规定道数，则插到后备队列，后备队列中的进程的状态为 new。CPU 每次调度时都从就绪队列中取进程，在进程执行过程中如果下一个操作时 IO 操作，则将进程插入到 waiting 队列。

在系统运行过程中可以执行进程挂起操作，但执行的挂起操作时系统自动暂停运

行，在弹出窗口选择要挂起的进程后，将选中的进程从原来的队列中删除并插入到挂起队列。进行解挂操作时将选中的进程从挂起队列中删除并插入该进程原来所处的队列。

➤ 按优先级调度：

当选择按优先权调度时，所有队列都采用优先队列，优先队列采用一个有序链表实现，进程的优先权值越大代表优先级越高，优先队列中的进程按优先权从大到小排列，当新进程插入时根据该进程的优先权插入到队列中的合适位置，插入后保持队列按优先权从大到小排列，如果新进程与队列中某个进程优先权值相等，则该新进程插到那个进程后面，以遵循先来先服务的规则。当要从队列中取出进程时总是取队列中第一个进程，因为该进程的优先级最高。

➤ 按时间片轮转调度：

当选择按时间片轮转调度时，所有队列都采用先进先出队列，先进先出队列采用一个普通单向链表实现，当新进程插入时插入到队列的末尾，当要取进程时取队首进程，这样就实现了先进先出。







➤ 内存管理：

该项目基于实验一完成，核心是内存的分配和回收，在实验一的基础上增加内存管理部分，在新建进程的时候增加一个输入内存大小的输入框，在进程进入内存时要分配内存，在进程销毁时要回收内存，如果进入内存时内存不足，则将进程插入到后备队列等待下次调度。系统维护一个内存表，每个表项代表一个空间，每个空间保存了该空间的起始地址和空间大小以及空间使用状态。初始时只有一个空间，当CPU启动时要分配内存，内存分配采用最先适应算法。回收内存时如果有相邻空闲空间，则要进行空闲空间合并。

## 二、总体设计思路

做这个实验时，一开始我觉得很难上手，甚至不知道从哪里做起没有一点的头绪，然后我决定采用模块化的设计，将整个程序的实现和功能分割开来，即建立不同的类，采用分而治之的思想，最后将其合并在一起，实现整个实验的运行。

我将整个实验分成下面几个 class：

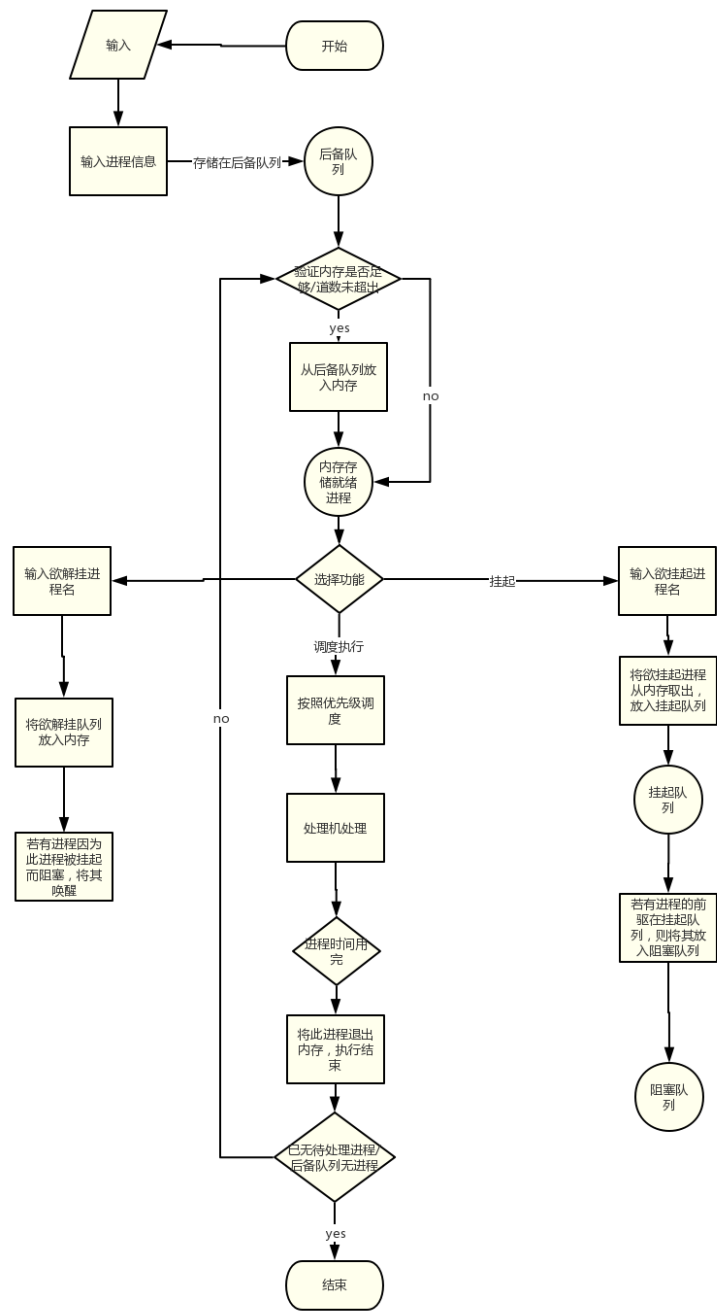
- ▼  os
  - >  dispatch.java
  - >  Memory.java
  - >  node.java
  - >  OS.java
  - >  PCB.java

- (1) PCB Class:定义 PCB 内容
- (2) Node Class:定义进程结点内容，定义进程名，运行时间，运行状态，优先级，所需内存，前趋指针等属性
- (3) Memory Class:定义内存空间
- (4) dispatch Class:定义处理进程的相关函数，包括进程排序，载入进程，进程调度，进程挂起，解挂进程，进程阻塞等
- (5) OS Class:图形界面，最后的测试运行

功能模块	功能描述	操作对象
创建进程	输入进程的信息，包括进程名、运行时间（单位时间）、优先权、状态、前驱，添加到后备队列中	后备队列
进程就绪	将后备队列中的进程添加到就绪队列中，从外存调入内存	后备队列，就绪队列
进程执行	根据调度结果，处理机执行内存中的某个进程，被执行的进程优先权-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	就绪队列，内存
进程挂起	输入欲挂起的进程名，将内存中的进程挂起，放入到挂起队列中	就绪队列，挂起队列，内存
进程阻塞	扫描内存中的进程，若某进程的前驱进程在挂起队列中，将其阻塞	就绪队列，阻塞队列，内存
进程解挂	输入欲解挂的进程名，将挂起队列中的进程解挂，放入到内存中	就绪队列，挂起队列，内存

进程排序	根据优先级，把进程按顺序排列，实现首次适应算法	后备队列，就绪队列
------	-------------------------	-----------

三、算法流程图



## 四 、 结构设计及相关代码

### （一）数据结构设计

PCB:

```
public class PCB {  
    String name;//进程名  
    int time;//执行时间  
    int status=5;//1 就绪, 2运行, 3阻塞, 4挂起, 5后备  
    int priority;//进程的优先级  
    int memory_size;//进程所占的优先级大小  
    int memory_point;//进程第一个块指向的内存位置  
    boolean cooperation;//若为true, 则为同步进程, 反之, 为false, 为独立  
    进程  
}
```

进程 Node:

```
import java.util.ArrayList;  
public class node {  
    PCB data=new PCB();  
    String pre;  
    ArrayList<node> next;  
    public node(String name,int time,int status,int priority,int  
memory_size,String prename)  
    {  
        this.data.name=name;  
        this.data.time=time;  
        this.data.status=status;  
        this.data.priority=priority;  
        this.data.memory_size=memory_size;
```

```

        this.pre = prename;
    }
}

```

内存空间:

```

public class Memory {

    int me[] ;

    int num =0;

    int maxnum;

    public Memory(int size,int maxnum)

    {

        me = new int[size];

        this.maxnum = maxnum;

    }

}

```

后备队列:

```

ArrayList<node> reserve_array = new ArrayList<node>();

```

//用 java 中的 ArrayList 数组, 结点数据为 node, 来存储后备进程

就绪队列:

```

ArrayList<node> ready_array = new ArrayList<node>();

```

//用 java 中的 ArrayList 数组, 结点数据为 node, 来存储就绪进程

阻塞队列:

```

ArrayList<node> pend_array = new ArrayList<node>();

```

//用 java 中的 ArrayList 数组, 结点数据为 node, 来存储挂起进程

挂起队列:

```

ArrayList<node> wait_array = new ArrayList<node>();

```

//用 java 中的 ArrayList 数组, 结点数据为 node, 来存阻塞进程

## (二) 添加进程模块(函数)的设计与实现

主要函数: `void addnode2reserve(node node)` //输入进程信息



```

{
    reserve_array.add(node); //将进程加入后备队列中
    sortqueue(reserve_array); //添加好的进程在后备队列，按优先级排列
}

```

输入进程的信息，包括进程名、运行时间（单位时间）、优先权、状态、前驱，添加到后备队列中。该功能依赖于图形界面中的 JTextfield 与后备队列一同完成。

在运行界面中输入数据，添加 node 对象，node 中的 PCB 存储相关信息，并将其 add（）入后备队列中。添加后，使用 nodesort（）按照优先级对队列中的进程进行排序。

若输入的进程没有前驱，前驱为-1.添加好的进程在后备队列，按优先级排列

### （三）调入进程模块（函数）的设计与实现

①按进程所需的内存大小，向内存申请一块内存，若申请条件符合，则为申请成功

```

boolean malloc(node node)
{
    int a = Integer.parseInt(node.data.name);
    int size =node.data.memory_size;
    for(int i=0;i<memory.me.length-size;i++)
    {
        boolean flag=true;
        for(int j=0;j<size;j++)
        {
            if(memory.me[i+j] != 0)
                flag=false;
        }
        if(flag)
        {
            for(int j=0;j<size;j++)
            {

```

```

        memory.me[i+j] = a;

    }

    node.data.memory_point = i;
    memory.num++;
    return true;
}

return false;

}

```

②从后备队列取出优先级最高的进程

```

boolean addnode2ready()
{

    node node = reserve_array.get(0);
    if(malloc(node)){
        node.data.status=1;
        ready_array.add(node);
        reserve_array.remove(0);
        return true;
    }
    else
        return false;
}

```

#### （四）执行进程模块（函数）的设计与实现

从就绪队列头中取出就绪进程，执行该进程，根据调度结果，处理机执行内存中的某个进程，被执行的进程优先权-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

主要函数：

```
String run() //从就绪队列队头中取出就绪进程，执行该进程
{
    node head = ready_array.get(0);
    if(!findnode(pend_array,head.pre).data.name.equals("0"))
    {
        block(head); //如果它的前趋进程被挂起，则该进程将会被阻塞
    }
    else{
        head.data.time -= 1;
        head.data.priority -= 1; //被执行的进程优先权-1，要求运行时间-1

        if (head.data.time <= 0) { //要求运行时间为0时，撤销该进程
            ready_array.remove(head);
            free(head);
            // printMemory();
            return head.data.name;
        }
    }
    return "";
}

void operate()
{
    run();
    sortqueue(ready_array); //一个时间片结束后，重新排序，进行下轮调度
}
```

## （五）挂起进程模块（函数）的设计与实现

可主动选取欲挂起进程，输入进程名，可将内存中的进程挂起，放入到挂起队列中。此时内存中与就绪队列中无该进程。

```
主要函数：void suspend(node node)

{

    ready_array.remove(node);

    node.data.status=4;

    free(node);

    pend_array.add(node);

}
```

## （六）解挂进程模块（函数）的设计与实现

可主动选取欲解挂进程，输入进程名，可将内存中的进程解挂，从挂起队列中取出，重新将此进程放入到内存中与就绪队列中。

```
主要函数：void active(node node)

{

    if(malloc(node)){

        pend_array.remove(node);

        node.data.status=1;

        ready_array.add(node);

        sortqueue(ready_array);

        wake_scan();

    }

}
```

## （七）阻塞进程模块（函数）的设计与实现

当某进程前驱被挂起时，该进程会阻塞。当该进程的前驱被解挂的时候，该进程会自动被唤醒。阻塞发生在执行前的检查时

```
void block(node node)
{
    ready_array.remove(node);
    node.data.status=3;
    wait_array.add(node);
}

void wake_scan()
{
    for(node node : wait_array )
    {
        if(findnode(pend_array,node.pre).data.name.equals("0"))
            wakeup(node);
    }
    sortqueue(ready_array);
}
```

## （八）唤醒进程模块（函数）的设计与实现

将阻塞队列中的进程唤醒到就绪队列

```
void wakeup(node node)
{
    wait_array.remove(node);
    node.data.status=1;
    ready_array.add(node);
}
```

## （九）进程排序模块（函数）的设计与实现

调用 Java 中的 Collections.sort（）函数，依据优先级，将进程排列。

排序方式：降序

```
void sortqueue(ArrayList<node> array)
{
    Collections.sort(array, new Comparator<node>()
    {
        @Override
        public int compare(node o1, node o2) {
            if(o1.data.priority<=o2.data.priority)
                return 1;
            else
                return -1;
        }
    });
}
```

## 五 、实验运行结果

### (1) 图形界面



## (2) 添加进程

输入进程名，时间，状态，优先级，大小，前趋进程。若没有前趋进程则为-1，添加好的进程在后备队列中，按优先级排列



## (3) 申请内存，调度执行

点击运行按钮，将后备队列中的进程添加到内存中

分配好内存，分配好执行的处理器，按照优先调度算法实现调度



#### (4) 挂起进程

输入欲挂起的进程名，点击挂起按钮



此时进程 1 被挂起，从内存中取出

#### (5) 进程阻塞

当运行到进程 2 时，因为进程 2 的前趋队列是进程 1，而进程 1 被挂起在挂起队列中，所以进程 2 被阻塞





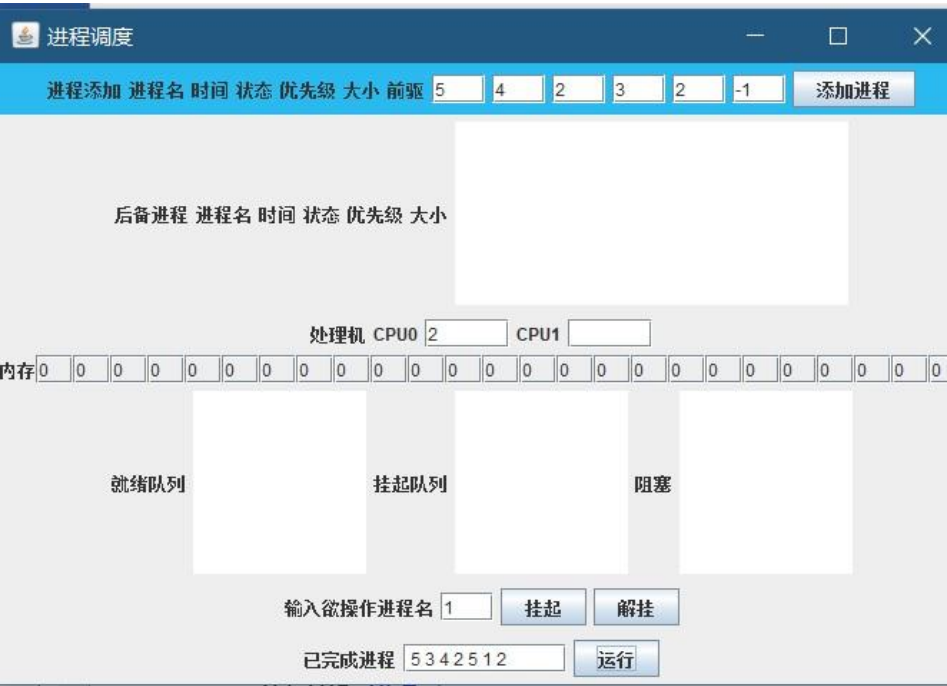
(6) 解挂进程

将进程 1 解挂，调度运行，进程 2 也会被唤醒



(7) 结果

要求执行时间为 0 的进程执行完毕，在底部显示：



(八) 具体代码：

① Dispatch. java:

```
package os;
import java.util.*;
public class dispatch {
```

ArrayList<node> ready\_array = new ArrayList<node>();//结点数据为node，来存储就绪进程

ArrayList<node> pend\_array = new ArrayList<node>();//结点数据为node，来存储挂起进程

ArrayList<node> reserve\_array = new ArrayList<node>();//结点数据为node，来存储后备进程

ArrayList<node> wait\_array = new ArrayList<node>();//结点数据为node，来存储阻塞进程

Memory memory;

int segment = 6;

public dispatch(Memory memory,int segment)

{

    this.memory = memory;

    this.segment =segment;

}

boolean malloc(node node)

{

    int a = Integer.parseInt(node.data.name);

    int size =node.data.memory\_size;

    for(int i=0;i<memory.me.length-size;i++)

    {

        boolean flag=true;

        for(int j=0;j<size;j++)

        {

            if(memory.me[i+j] != 0)

                flag=false;

        }

        if(flag)

        {

            for(int j=0;j<size;j++)

            {

                memory.me[i+j] = a;

            }

            node.data.memory\_point =i;

            memory.num++;

            return true;

        }

    }

    return false;

}

void free(node node)

{

    for(int i

=node.data.memory\_point;i<node.data.memory\_point+node.data.memory\_size;i++)

    {

```

        memory.me[i]=0;
    }
    memory.num--;
}
boolean addnode2ready()
{
    node node = reserve_array.get(0);
    if(malloc(node)){ //按进程所需的内存大小，向内存申请一块内存，若申请条件符合，
    则为申请成功。
        //从后备队列取出最优先的进程
        node.data.status=1;
        ready_array.add(node);
        reserve_array.remove(0);
        return true;
    }
    else
        return false;
}
void sortqueue(ArrayList<node> array)//调用Java中的Collections.sort（）函数，
依据优先级，将进程排列
{
    Collections.sort(array, new Comparator<node>()
    {
        @Override
        public int compare(node o1, node o2) {
            if(o1.data.priority<=o2.data.priority)
                return 1;
            else
                return -1;
        }
    });
}
String run() //从就绪队列队头中取出就绪进程，执行该进程
{
    node head = ready_array.get(0);
    if(!findnode(pend_array,head.pre).data.name.equals("0"))
    {
        block(head); //如果它的前趋进程被挂起，则该进程将会被阻塞
    }
    else{
        head.data.time -= 1;
        head.data.priority -= 1; //被执行的进程优先权-1，要求运行时间-1
        if (head.data.time <= 0) { //要求运行时间为0时，撤销该进程
            ready_array.remove(head);
            free(head);
        }
    }
}

```

```

        // printMemory();
        return head.data.name;
    }
}
return "";
}
void operate()
{
    run();
    sortqueue(ready_array); //一个时间片结束后，重新排序，进行下轮调度
}
void printList()
{
    for(node eachnode: ready_array)
    {
        System.out.println(eachnode.data.name+" "
            +eachnode.data.time+" "+eachnode.data.priority);
    }
}
void printnode()
{
    node head = ready_array.get(0);
    System.out.println(head.data.name+" "
        +head.data.time+" "+head.data.priority);
}
void printMemory()
{
    System.out.println("-----");
    for(int i=0;i<memory.me.length;i++)
    {
        System.out.print(memory.me[i]);
    }
    System.out.println();
    System.out.println("-----");
}
void block(node node)//当某进程前驱被挂起时，该进程会阻塞。当该进程的前驱被解挂的
时候，该进程会自动被唤醒。
{
    ready_array.remove(node);
    node.data.status=3;
    wait_array.add(node);
}
void wakeup(node node)
{
    wait_array.remove(node);

```

```

        node.data.status=1;
        ready_array.add(node);
    }
    void wake_scan()//阻塞发生在执行前的检查时
    {
        for(node node : wait_array )
        {
            if(findnode(pend_array,node.pre).data.name.equals("0"))
                wakeup(node);
        }
        sortqueue(ready_array);
    }
    void suspend(node node)//可主动选取欲挂起进程，输入进程名，可将内存中的进程挂起，
    放入到挂起队列中
    {
        ready_array.remove(node);
        node.data.status=4;
        free(node);
        pend_array.add(node);
    }

    void active(node node)//可主动选取欲解挂进程，输入进程名，可将进程解挂，从挂起队
    列中取出，重新将此进程放入到内存中与就绪队列中
    {
        if(malloc(node)){
            pend_array.remove(node);
            node.data.status=1;
            ready_array.add(node);
            sortqueue(ready_array);
            wake_scan();
        }
    }
    public node findnode(ArrayList<node> array, String name){
        for(node eachnode : array)
        {
            if(eachnode.data.name.equals(name))
            {
                return eachnode;
            }
        }
        node n = new node("0",0,0,0,0,"0");
        return n;
    }
    void addnode2reserve(node node)//输入进程信息
    {

```

```

        reserve_array.add(node); //将进程加入后备队列中
        sortqueue(reserve_array); //添加好的进程在后备队列，按优先级排列
    }
    void loadnode()
    {
        while(ready_array.size()<segment&&reserve_array.size()>0
            && addnode2ready())
        { }
    }
}

```

## ② Memory. java

```

package os;
public class Memory {
    int me[] ;
    int num =0;
    int maxnum;
    public Memory(int size,int maxnum)
    {
        me = new int[size];
        this.maxnum = maxnum;
    }
}

```

## ③ Node. java

```

package os;

import java.util.ArrayList;

public class node {
    PCB data=new PCB();
    String pre;
    ArrayList<node> next;

    public node(String name,int time,int status,int priority,
        int memory_size,String prename)
    {
        this.data.name=name;
        this.data.time=time;
        this.data.status=status;
        this.data.priority=priority;
    }
}

```

```

        this.data.memory_size=memory_size;
        this.pre = prename;
    }
}

```

#### ④ PCB. java

```

package os;
public class PCB {
    String name;//进程名
    int time;//执行时间
    int status=5;//1 就绪, 2运行, 3阻塞, 4挂起, 5后备
    int priority;//进程的优先级
    int memory_size;//进程所占的优先级大小
    int memory_point;//进程第一个块指向的内存位置
    boolean cooperation;//若为true, 则为同步进程, 反之, 为false, 为独立进程
}

```

#### ⑤ OS. java

```

package os;
import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.ImageObserver;
import java.text.AttributedCharacterIterator;
public class OS implements ActionListener{
    Memory memory = new Memory(25,5);
    dispatch dis = new dispatch(memory,5);
    private JFrame f;
    private JLabel Label1;
    private JLabel Label2;
    private JLabel Label3;
    private JLabel Label4;
    private JLabel Label5;
    private JLabel Label6;
    private JLabel Label7;
    private JLabel Label8;
    private JLabel Labelname = new JLabel("进程名");
    private JLabel Labelpriority = new JLabel("时间");
    private JLabel Labelt = new JLabel("状态");
    private JLabel Labelpar= new JLabel("优先级");
    private JLabel Labelsize= new JLabel("大小");
}

```

```

private JButton runButton;
private JButton pendButton;
private JButton activeButton;
private JButton addButton;
private JTextField text1 = new JTextField(3);
private JTextField text2 = new JTextField(3);
private JTextField text3 = new JTextField(3);
private JTextField text4 = new JTextField(3);
private JTextField text5 = new JTextField(3);
private JTextField text6 = new JTextField(3);//p_add
private JTextField text7 = new JTextField(5);
private JTextField text8 = new JTextField(5);//cpu
private JTextField text9 = new JTextField(3);//挂起
private JTextField text10 = new JTextField(10);
private JTextField text11 = new JTextField(30);
private JTextField text12 = new JTextField();
private JTextField text13 = new JTextField();
JTextArea ta1;
JTextArea ta2=new JTextArea(8,11);
JTextArea ta3=new JTextArea(8,11);
JTextArea ta4=new JTextArea(8,11);
private JOptionPane box;
JPanel panel_add = new JPanel();
JPanel panel_reserve = new JPanel();
JPanel panel_memory = new JPanel();
JPanel panel_queue = new JPanel();
JPanel panel_cpu = new JPanel();
JPanel panel_finish = new JPanel();
JPanel panel_pend = new JPanel();
Color tColor[] = new Color[10];
private JTextField memoryText[] = new JTextField[25];
void initcolor()
{
    tColor[0] = new Color(255,255,128);
    tColor[1] = new Color(0,255,255);
    tColor[2] = new Color(255,128,128);
    tColor[3] = new Color(128,255,128);
    tColor[4] = new Color(255,0,0);
    tColor[5] = new Color(255,0,255);
    tColor[6] = new Color(255,128,0);
    tColor[7] = new Color(0,128,255);
    tColor[8] = new Color(208,208,0);
    tColor[9] = new Color(215,174,174);
}
public void init()

```



```

{
    JFrame f = new JFrame("进程调度");
    f.setSize(800,1000);
    f.setLocation(200,200);
    int r = 43;
    int g = 186;
    int b = 240;
    Color bgColor = new Color(r, g, b);
    int j =1;
    initcolor();
    // 设置背景颜色
    f.setBackground(bgColor);
    panel_add.setLayout(new FlowLayout());//使用流式布局法
    panel_add.setBackground(bgColor);
    runButton = new JButton("运行");
    pendButton = new JButton("挂起");
    activeButton = new JButton("解挂");
    addButton =new JButton("添加进程");
    addButton.addActionListener(this);
    pendButton.addActionListener(this);
    activeButton.addActionListener(this);
    runButton.addActionListener(this);
    Label1=new JLabel("后备队列");
    Label2=new JLabel("就绪队列");
    Label3=new JLabel("挂起队列");
    Label4=new JLabel("阻塞队列");
    Label5=new JLabel("CPU0");
    Label6=new JLabel("CPU1");
    Label7=new JLabel();
    Label8 =new JLabel();
    panel_add.add(new JLabel("进程添加"));
    panel_add.add(Labelname);
    panel_add.add(Labelpriority);
    panel_add.add(Labelt);
    panel_add.add(Labelpar);
    panel_add.add(Labelsize);
    panel_add.add(new JLabel("前驱"));
    panel_add.add(text1);
    panel_add.add(text2);
    panel_add.add(text3);
    panel_add.add(text4);
    panel_add.add(text5);
    panel_add.add(text6);
    panel_add.add(addButton);
    panel_reserve.add(new JLabel("后备进程"));
}

```

```

panel_reserve.add( new JLabel("进程名"));
panel_reserve.add(new JLabel("时间"));
panel_reserve.add(new JLabel("状态"));
panel_reserve.add(new JLabel("优先级"));
panel_reserve.add(new JLabel("大小"));
panel_reserve.add(ta1=new JTextArea(8,25));
panel_cpu.add(new JLabel("处理机"));
panel_cpu.add(new JLabel("CPU0"));
panel_cpu.add(text7);
panel_cpu.add(new JLabel("CPU1"));
panel_cpu.add(text8);
panel_queue.add(new JLabel("就绪队列"));
panel_queue.add(ta2);
panel_queue.add(new JLabel("挂起队列"));
panel_queue.add(ta3);
panel_queue.add(new JLabel("阻塞"));
panel_queue.add(ta4);
panel_pend.add(new JLabel("输入欲操作进程名"));
panel_pend.add(text9);
panel_pend.add(pendButton);
panel_pend.add(activeButton);
panel_finish.add(new JLabel("已完成进程"));
panel_finish.add(text10);
panel_finish.add(runButton);
panel_memory.add(new JLabel("内存"));
//panel_memory.add(text11);
panel_memory.setLayout(new GridLayout());
for(int i=0;i<memoryText.length;i++)
{
    memoryText[i] = new JTextField(2);
    memoryText[i].setText(i + "");
    panel_memory.add(memoryText[i]);
}
Box vBox = Box.createVerticalBox();
vBox.add(panel_add);
//f.add(panel_add);
vBox.add(panel_reserve);
vBox.add(panel_cpu);
vBox.add(panel_memory);
vBox.add(panel_queue);
vBox.add(panel_pend);
vBox.add(panel_finish);
f.setContentPane(vBox);
f.pack();
f.setVisible(true);

```

```

        drawMemory();
    }
    void reflash()
    {
        ta1.setText("");
        ta2.setText("");
        ta3.setText("");
        ta4.setText("");
        for(node node : dis.reserve_array)
        {
            ta1.append(node.data.name+" "+node.data.time+" "+node.data.status+" "
                +node.data.priority+" "+node.data.memory_size+" "+node.pre+"\r\n");
        }
        for(node node : dis.ready_array)
        {
            ta2.append(node.data.name+" "+node.data.time+" "+node.data.status+" "
                +node.data.priority+" "+node.data.memory_size+" "+node.pre+"\r\n");
        }
        for(node node : dis.pend_array)
        {
            ta3.append(node.data.name+" "+node.data.time+" "+node.data.status+" "
                +node.data.priority+" "+node.data.memory_size+" "+node.pre+"\r\n");
        }
        for(node node : dis.wait_array)
        {
            ta4.append(node.data.name+" "+node.data.time+" "+node.data.status+" "
                +node.data.priority+" "+node.data.memory_size+" "+node.pre+"\r\n");
        }
        for (int i=0;i<memory.me.length;i++)
        {
            int te = memory.me[i];
            if( te != 0)
            {
                memoryText[i].setBackground(tColor[te]);
                memoryText[i].setText(""+te);
            }
            if(te == 0)
            {
                memoryText[i].setBackground(null);
                memoryText[i].setText("0");
            }
        }
        dis.wake_scan();
    }
    void drawMemory()

```

```

{
    //g.drawRect(10,10,10,10);
}
@Override
public void actionPerformed(ActionEvent e) {
    //try {
        if (e.getSource() == addButton) {
            if (!text1.getText().equals("")) {
                String name = text1.getText();
                int time = Integer.parseInt(text2.getText());
                int status = Integer.parseInt(text3.getText());
                int priority = Integer.parseInt(text4.getText());
                int memorysize = Integer.parseInt(text5.getText());
                String prename = text6.getText();
                node node = new node(name, time, status, priority, memorysize, prename);
                dis.addnode2reserve(node);
                reflash();
            }
        } else if (e.getSource() == runButton) {
            dis.loadnode();
            node node1 = dis.ready_array.get(0);
            text7.setText(node1.data.name);
            String str=node1.data.name;
            if (node1.data.time==1) {
                String s = text10.getText() + " " + str;
                text10.setText(s);
            }
            dis.run();
            dis.sortqueue(dis.ready_array);
            if (dis.ready_array.size() >1) {
                node node2 = dis.ready_array.get(0);
                if (node2.data.name.equals(node1.data.name))
                {
                    node2=dis.ready_array.get(1);
                }
                text8.setText(node2.data.name);
                String str2=node1.data.name;
                if (node2.data.time==1) {
                    String s = text10.getText() + " " + str2;
                    text10.setText(s);
                }
                dis.run();
                dis.sortqueue(dis.ready_array);
            } else
                text8.setText("");
        }
    }
}

```

```

        refresh();
    } else if (e.getSource() == pendButton) {
        node node = dis.findnode(dis.ready_array, text9.getText());
        if (!node.data.name.equals("0")) {
            dis.suspend(node);
            refresh();
        }
    } else if (e.getSource() == activeButton) {
        node node = dis.findnode(dis.pend_array, text9.getText());
        if (!node.data.name.equals("0")) {
            dis.active(node);
            refresh();
        }
    }
}

// catch(Exception ex) {}
//}

public static void main(String[] args) {
    OS os = new OS();
    os.init();
}
}

```

## 六、心得体会

(心得体会)

老师上课的时候多次强调进程是很重要必须要理解的概念,所以在学习进程和进程调度这一块的时候,我花了很多时间去理解进程调度的细节,通过 Java 写程序去模拟实现就绪队列、后备队列这些数据结构的设置,理解挂起、阻塞的细节和首次适应算法等。实现处理机调度及内存分配和回收制度,以及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。

在这次实验进行的过程中,我遇到了很多问题,但是通过上网阅读博客,把问题都解决了,对解决问题的能力有了进一步提升。学习使用 java 里面的 Collections 容器,学习它自带的排序方法,对 java 的理解有帮助。在设置数据结构的时候花费了一点时间,首先想到的是创建进程和 PCB,那就要设置 PCB 和进程 node 类,接着要对进程进行一系列操作,那就将操作都定义在一个类中,在将进程调入内存时,需要对它进行内存分配,所以要设置内存空间 Memory 类,构思的时候觉得很清楚,动手去写的时候发现 bug 一大堆,有的时候逻辑有错误会一直 error, debug 需要很长时间,但这也是在训练代码能力和解决问题的能力吧,也算培养了自己的耐心。

这次实验也让自己对 java 语言更熟悉,尤其是图形界面这一块,用 java 写图形界面也确实非常方便。通过这个实验,也让我更加形象地了解了进程的调度过程,加深了对于优先权调度和时间片轮转调度的理解,并不像从前一样仅仅停留在概念上。

通过次实验,我对内存分配和内存回收有了更深刻的了解,我们平时用电脑时简单的一个动作对内存来说却要做出如此多的反应,找到一个空闲并且大小合适的空间进行内存分配。本次实验使我对内存分配的了解有了很大的帮助。在这次编程中我也出现了很多程序上的简单错误,都是因为我动手写程序比较少造成的,这也让我了解到要多次锻炼才能顺手成章。在以后的练习中,我要努力培养先想清楚,有了大致的结构再动手的习惯,努力做到编程序不用 debug。

## 七、参考文献

- [1] 张尧学等. 计算机操作系统教程. 清华大学出版社, 2013
- [2] 陈向群等. Windows内核实验教程. 机械工业出版社, 2004
- [3] 罗宇等. 操作系统课程设计. 机械工业出版社, 2005