

**课 程 设 计 报 告**

( 2019--2020年度第二学期)

**课程名称: 操作系统课程设计**

**课设题目:** SP00LING假脱机输入输出技术模拟

**院 系:** **控制与计算机工程学院**

**班 级:** 计算1702

**姓 名:** 杨秉学

**学 号:** 120171080212

**指导教师:** 李为老师

**设计周数:**  **一周**

**成 绩:**

**2020年6月21日**

设计报告内容

# 需求分析

## 1．设计一个实现SP001ING技术的进程

要求设计一个SP00LING输出进程和两个请求输出的用户进程，以及一个SP00LING输出服务程序。当请求输出的用户进程希望输出一系列信息时，调用输出服务程序，由输出服务程序将该信息送入输出井。待遇到一个输出结束标志时，表示进程该次的输出文件输出结束。之后，申请一个输出请求块(用来记录请求输出的用户进程的名字、信息在输出井中的位置、要输出信息的长度等)，等待SP00LING进程进行输出。

SP00LING输出进程工作时，根据请求块记录的各进程要输出的信息，将其实际输出到打印机或显示器。这里，SP00LING输出进程与请求输出的用户进程可并发运行。

## 2．设计进程调度算法

进程调度采用随机算法，这与进程输出信息的随机性相一致。两个请求输出的用户进程的调度概率各为45％，SP00LING输出进程为10％，这由随机数发生器产生的随机数来模拟决定。

## 3．进程状态

进程基本状态有3种，分别为可执行、等待和结束。可执行态就是进程正在运行或等待调度的状态；等待状态又分为等待状态1、等待状态2和等待状态3。

状态变化的条件为：

①进程执行完成时，置为“结束”态。

②服务程序在将输出信息送输出井时，如发现输出井已满，将调用进程置为“等待状态1”。

③SP00LING进程在进行输出时，若输出井空，则进入“等待状态2”。

④SP00LING进程输出一个信息块后，应立即释放该信息块所占的输出井空间，并将正在等待输出的进程置为“可执行状态”。

⑤服务程序在输出信息到输出井并形成输出请求信息块后，若SP00LING进程处于等待态，则将其置为“可执行状态”。

⑥当用户进程申请请求输出块时，若没有可用请求块时，调用进程进人“等待状态3”。

# 整体功能及设计（功能划分及流程图）

## 数据结构

1. PCB

public class PCB {

    int ID;               //进程标识数

    int status;          //进程状态

    int outputFileCount; //要输出的文件数

    int outPut\_X;        //进程输出时的临时变量

}

1. 请求输出块

public class requireBlock {

    int requireName;        //请求进程名

    int length;             //本次输出信息长度

    int outputHeadAddress;  //信息在输出井的首地址

}

1. 其他控制变量

int buffer[][];                 //定义输出井Buffer

    int outputBufferSpace[];        //可使用的输出井buffer空间

    int outputBufferPointer[][];    //输出井buffer空闲和满指针

    int requireblockNumber;         //requireblock的剩余数量

    int requireblockOutputPointer;  //要输出的第一个reqblock指针

    int requireblockFreePointer;    //第一个空闲reqblock指针

    double random;                  //用于调度三个进程的控制随机数

    int output1;                    //用户进程1已生成的文件数

    int output2;                    //用户进程2已生成的文件数

    int output\_1;                   //用户进程1已输出的文件数

    int output\_2;                   //用户进程2已输出的文件数

    int x;                          //随机生成的数据0～9

    int i;                          //临时控制变量

    Random x1;                      //辅助生成随机数据x：0～9

## 主要类和函数

1. **主类 public class Spooling**

public static void main( String args[] ) 主函数，用来生成窗体类

1. **窗体类 public class Window**

public Window(); 构造函数，规划界面，并且为button加上事件监听

public void actionPerformed(ActionEvent e)单击事件响应函数

public void begin(): 点击"重新开始"按钮时，重新初始化界面

public void run()： 点击”开始运行”时，转入调度函数

1. **调度类 public class Manage**

public Manage(Window window)：构造函数，对进程的数据初始化

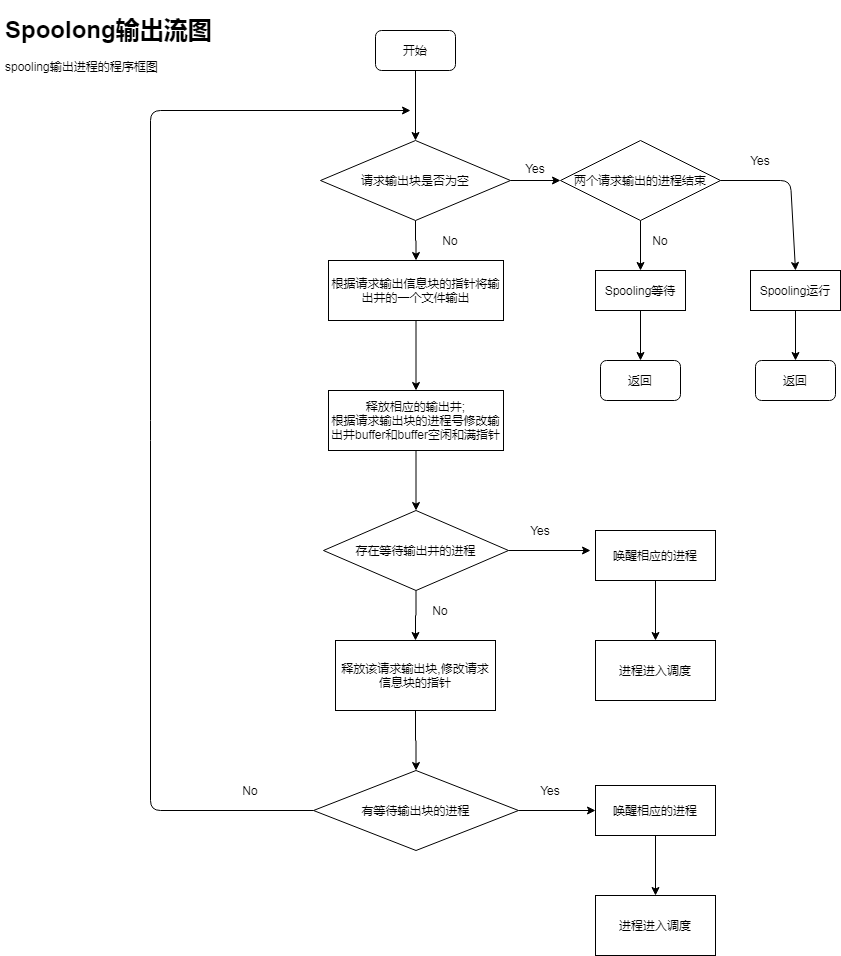
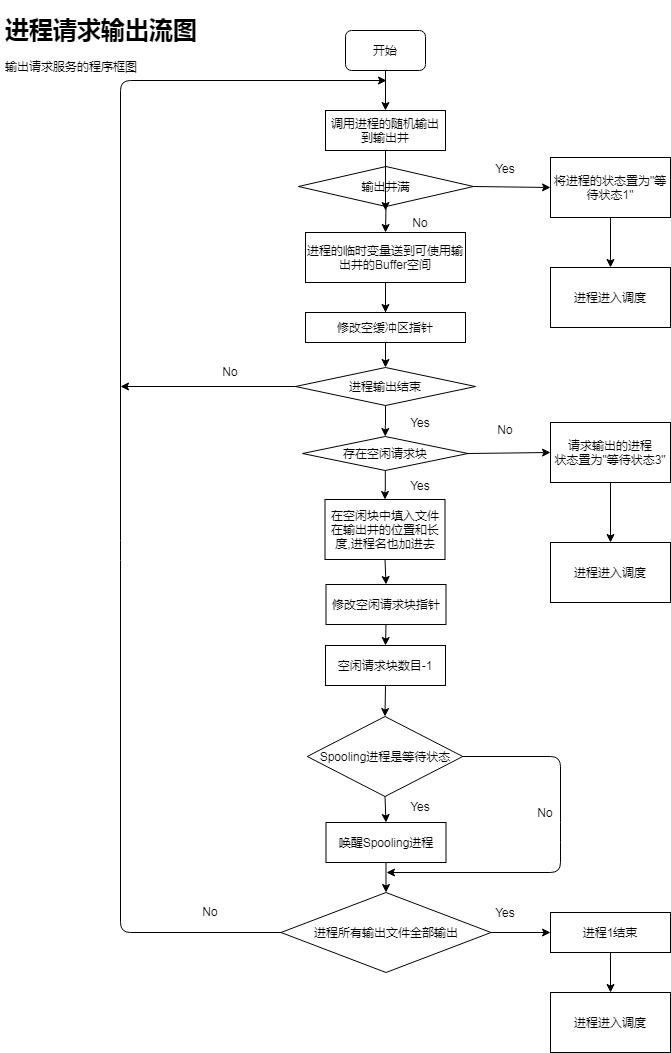
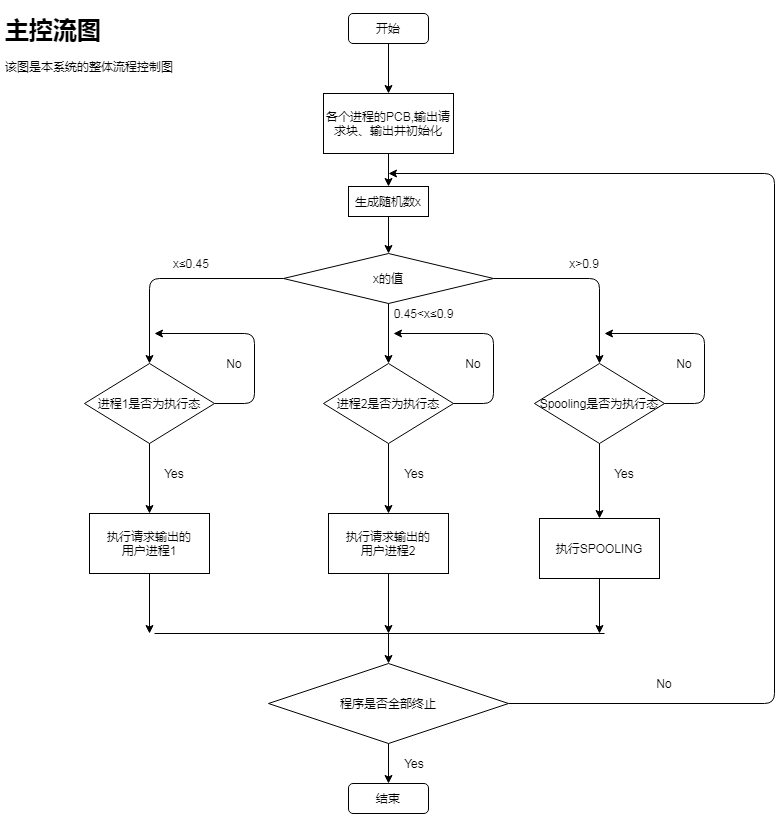
public void run()：主控函数，用随机数来决定调度的进程，使得两个请求输出的用户进程的调度概率各为45％，SP00LING输出进程为10％。

public int user(int name,int out,JTextAreatextarea jta, JTextFieldfield jtf)：SP00LING输出服务程序，当请求输出的用户进程希望输出一系列信息时，通过传参调用输出服务程序，由输出服务程序将该信息送入输出井。

public void spooling1()： SPOOLING输出进程，根据请求块记录的各进程要输出的信息，把信息输出到文本框里。

1. **PCB类 public class PCB**
2. **请求输出块类 public class requireBlock**

**3.流程图**



# [编程实现（要求有注释）](https://github.com/IammyselfYBX/SPOOLING_Test)

## 1.条款协议

本程序借鉴采纳GPLv2.0条款的代码，所以根据法律，本程序也要符合GPLv2.0协议。

## 2.PCB.java

public class PCB {

    int ID;               //进程标识数

    int status;         //进程状态

    int outputFileCount;//要输出的文件数

    int outPut\_X;       //进程输出时的临时变量

}

## 3.requireBlock.java

public class requireBlock {

    int requireName;            //请求进程名

    int length;                   //本次输出信息长度

    int outputHeadAddress;  //信息在输出井的首地址

}

## 4.Spooling.java

public class Spooling{

    public static void main( String args[] ){

        Window window = new Window();

    }

}

## 5.Window.java

/\*\*

 \* @author:杨秉学

 \* Time:2020,6,24

 \* Function:

 \*      1.负责窗体的布局UI设计

 \*      2.进行事件响应

 \* @version 1.2

 \*/

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.lang.reflect.Field;

public class Window extends JFrame implements ActionListener, Runnable{

    JPanel jp1, jp2, jp3;

    JTextField jtf1, jtf2;

    JScrollPane p1, p2, p3, p4;

    JTextArea jta1, jta2, jta3, jta4;

    JButton btn1, btn2, btn3;

//  Manage manage;

    /\*\*

     \* 界面构造函数

     \*/

    public Window(){

        Container container = this.getContentPane();

        container.setLayout( new BorderLayout() );

        /\*\*

         \* 设置panel1

         \*/

        jtf1 = new JTextField( 3 );

        jtf2 = new JTextField( 3 );

        btn1 = new JButton( "开始运行" );

        btn1.addActionListener( this );

        btn2 = new JButton( "退出程序" );

        btn2.addActionListener( this );

        btn3 = new JButton( "重新开始" );

        btn3.addActionListener( this );

        jp1 = new JPanel();

        jp1.setLayout( new FlowLayout() );

        jp1.add( new JLabel( "User Process1 文件数：", SwingConstants.RIGHT ) );

        jp1.add( jtf1 );

        jp1.add( new JLabel( "User Process2 文件数：", SwingConstants.RIGHT) );

        jp1.add( jtf2 );

        jp1.add( btn1 );

        jp1.add( btn2 );

        jp1.add( btn3 );

        /\*\*

         \* 设置panel2

         \*/

        jta1 = new JTextArea( 80,100 );//设置panel2

        jta2 = new JTextArea( 112,400 );

        jta1.append( "用户进程1的文件生成情况及状态变换\n文件序号   位    置  剩余空间    状态\n" );

        jta2.append("用户进程2的文件生成情况及状态变换\n文件序号    位    置  剩余空间    状态\n");

        p1 = new JScrollPane( jta1 );

        p2 = new JScrollPane( jta2 );

        jp2 = new JPanel();

        jp2.setLayout ( new GridLayout( 1,2 ) );

        jp2.add( p1 );

        jp2.add( p2 );

        /\*\*

         \* 设置panel3

         \*/

        jta3 = new JTextArea( 15,400 );//设置panel3

        jta4 = new JTextArea( 15,150 );

        jta3.append( "Spooling调度输出\n" );

        jta4.append("主程序调度\n");

        p3 = new JScrollPane( jta3 );

        p4 = new JScrollPane( jta4 );

        jp3 = new JPanel();

        jp3.setLayout( new GridLayout( 1,2 ) );

        jp3.add( p3 );

        jp3.add( p4 );

        container.add( jp1, BorderLayout.NORTH );//设置窗口

        container.add( jp2 ,BorderLayout.CENTER );

        container.add( jp3, BorderLayout.SOUTH );

        this.setSize( 1200,600 );

        this.setLocation( 100,100 );

        this.setTitle( "SPOOLING假脱机输入输出技术模拟" );

        this.setVisible( true );

        this.setDefaultCloseOperation( JFrame.EXIT\_ON\_CLOSE );//设置窗口完毕

    }

    public void actionPerformed( ActionEvent e ){

     if(e.getSource() == btn1)

     {

        run();

     }

     if(e.getSource() == btn2)

     {

        System.exit(-1);

     }

     if(e.getSource() == btn3)

     {

        begin();

     }

    }

  public void run()

  {

         Manage manage = new Manage(this);

     manage.start();

  }

  public void begin()

  {

     jtf1.setText("");

     jtf2.setText("");

         jta1.setText( "用户进程1的文件生成情况及状态变换\n文件序号 位    置  剩余空间    状态\n" );

         jta2.setText("用户进程2的文件生成情况及状态变换\n文件序号  位    置  剩余空间    状态\n");

         jta3.setText( "Spooling调度输出\n" );

         jta4.setText("主程序调度\n");

  }

}

## 6.Manage.java

import java.util.Random;

import javax.swing.\*;

public class Manage extends Thread{

    PCB pcb[];

    requireBlock requireblock[];

    int buffer[][];                 //定义输出井Buffer

    int outputBufferSpace[];            //可使用的输出井buffer空间

    int outputBufferPointer[][];      //输出井buffer空闲和满指针

    int requireblockNumber;           //requireblock的剩余数量

    int requireblockOutputPointer;  //要输出的第一个reqblock指针

    int requireblockFreePointer;      //第一个空闲reqblock指针

    double random;                        //用于调度三个进程的控制随机数

    int output1;                              //用户进程1已生成的文件数

    int output2;                              //用户进程2已生成的文件数

    int output\_1;                           //用户进程1已输出的文件数

    int output\_2;                           //用户进程2已输出的文件数

    int x;                                      //随机生成的数据0～9

    int i;                                      //临时控制变量

    Random x1;                                //辅助生成随机数据x：0～9

    Window window;                  //创建窗体UI

    /\*\*

     \* 这个方法是对两个进程进行数据初始化

     \* @param window

     \*/

    public Manage( Window window ){

        output1 = 0;

        output2 = 0;

        output\_1 = 0;

        output\_2 = 0;

        pcb = new PCB[4];

        requireblock = new requireBlock[10];

        buffer = new int[3][100];     //buffer[1][100]表示用户1有100个,buffer[2][100]表示用户2有100个

        outputBufferSpace = new int[3];

        outputBufferSpace[1] = 100;

        outputBufferSpace[2] = 100;

        outputBufferPointer = new int[3][2];

        outputBufferPointer[1][0] = 0;

        outputBufferPointer[2][0] = 0;

        requireblockNumber = 10;

        requireblockOutputPointer = 0;

        requireblockFreePointer = 0;

        x1 = new Random();

        for( i = 0; i < 4; i++ ){

            pcb[i] = new PCB();

        }

        for( i = 0; i < 10; i++ ){

            requireblock[i] = new requireBlock();

        }

        for( i = 1; i <=3; i++ ){

            pcb[i].status = 0;

        }

        this.window = window;   //对各进程的数据初始化完毕  把这个传过来有助于更新界面内容

    }

    public void run(){

    //进程调度

        do{

            random = Math.random();//产生一个随机数，控制进程调度，令用户进程概率为45%，Spooling进程为10%

            if( random <= 0.45 &&  pcb[ 1 ].status == 0 ){ //调度用户进程1

                window.jta4.append( "调度用户进程1\n" );

                try{

                    sleep( 500 );

                }

                catch( InterruptedException e ){

                    e.printStackTrace();

                }

                ///调用用户函数去生成文件填入输出井生成请求块

                output1 = user( 1, output1, window.jta1, window.jtf1 );

            }

            else if( random >0.45 && random <= 0.9 && pcb[ 2 ].status == 0 ){ //调度用户进程2

                window.jta4.append( "调度用户进程2\n" );

                try{

                    sleep( 500 );

                }

                catch( InterruptedException e ){

                    e.printStackTrace();

                }

                output2 = user( 2, output2, window.jta2, window.jtf2 );

            }

            else if( random >0.9 && random <= 1 && pcb[3].status == 0 ){ //调度spooling进程

                window.jta4.append("调度Spooling进程\n");

                try{

                    sleep( 500 );

                }

                catch( InterruptedException e ){

                    e.printStackTrace();

                }

                spooling1();

            }

        }while( pcb[1].status != 4 || pcb[2].status != 4 || pcb[3].status != 4 ); // 当都是结束状态曾程序运行完成

    //进程调度结束

        window.jta4.append("程序结束\n");

    }

    /\*\*

   \* fuction:显示用户进程已经生成的文件数目

   \*

   \*/

  public int user( int name, int out, JTextArea jta, JTextField field ){

    pcb[name].ID = name;

        pcb[name].outputFileCount = Integer.parseInt( field.getText() );

        while( out != pcb[ name ].outputFileCount ){//判断进程所要输出的文件是否输出完毕的while循环

            outputBufferPointer[ name ][ 1 ] = outputBufferPointer[ name ][ 0 ];

            do{ //判断进程的一个文件是否输出完毕的while循环

                x = x1.nextInt(9);//x为每次随机生成的数据0～9，送入pcb.x

                pcb[name].outPut\_X = x;

                if( outputBufferSpace[name] == 0 ){ //若输出井buffer满，变为等待状态1，转调度程序

                    pcb[name].status = 1;

                    //一个文件的长度是未知的在创建的过程中如果发现井满了要退回去

                    if( outputBufferPointer[ name ][ 0 ] >= outputBufferPointer[ name ][1] ){

                        outputBufferSpace[ name ] = outputBufferSpace[ name ] + outputBufferPointer[ name ][ 0] - outputBufferPointer[ name ][ 1 ];//将空间释放

                    }

                    else{

                        outputBufferSpace[ name ] = outputBufferSpace[ name ] + 100 + outputBufferPointer[ name ][ 0] + outputBufferPointer[ name ][ 1 ];

                    }

                    outputBufferPointer[ name ][ 0 ] = outputBufferPointer[ name ][ 1 ]; //将空间释放

                    jta.append( "第" + ( out + 1 ) + "个文件缺少输出井" );

                    jta.append( "进入等待状态1\n" );

                    try{

                        sleep( 500 );

                    }

                    catch( InterruptedException e ){

                        e.printStackTrace();

                    }

                    return out;

                }

                else{//若输出井没满

                    buffer[ name ][ outputBufferPointer[ name ][ 0 ] ] = pcb[ name ].outPut\_X;  //进程的输出信息PCB[i].x送buffer[i][ outputBufferPointer[i][0]]

                    outputBufferSpace[ name ] = outputBufferSpace[ name ] - 1; //输出井空闲个数减1

                    outputBufferPointer[ name ][ 0 ] = ( outputBufferPointer[ name ][ 0 ] + 1 ) % 100;  //修改空缓冲区指针outputBufferPointer[i][0]前进1

                }

            }

            while( x != 0 ); //判断进程的一个文件是否输出完毕的while循环结束

            if( outputBufferPointer[ name ][ 0 ] == 0 ){

                jta.append( (out + 1)  + " " +  outputBufferPointer[ name ][ 1 ] + " ~ " + " 99 " + outputBufferSpace[ name ] + " ");

            }

            else{

                jta.append( (out + 1)  + " " +  outputBufferPointer[ name ][ 1 ] + " ~ " + ( outputBufferPointer[ name ][ 0 ] - 1 ) + " " + outputBufferSpace[ name ] + " " );

            }

            try{

                sleep( 500 );

            }

            catch( InterruptedException e ){

                e.printStackTrace();

            }

            out++; //成功生成了一个文件

            //接下来要生成相应的请求块

            if( requireblockNumber == 0 ){//若没有空闲请求输出块，转为等待状态3

                pcb[ name ].status = 3;

                jta.append( "缺少请求输出块" );

                jta.append( "进入等待状态3 \n" );

                try{

                    sleep( 500 );

                }

                catch( InterruptedException e ){

                    e.printStackTrace();

                }

                return out;

            }

            else{ //若有空闲请求输出块

                requireblock[ requireblockFreePointer ].outputHeadAddress = outputBufferPointer[ name ][ 1 ]; //将文件在输出井的位置填入空闲请求块

                if( outputBufferPointer[ name ][ 0 ] >= outputBufferPointer[ name ][ 1 ] ){ //将文件在输出井的长度填入空闲请求块

                    requireblock[ requireblockFreePointer ].length = outputBufferPointer[ name ][ 0 ] - outputBufferPointer[ name ][ 1 ];

                }

                else{

                    requireblock[ requireblockFreePointer ].length = 100 - outputBufferPointer[ name ][ 1 ] + outputBufferPointer[ name ][ 0 ];

                }

                requireblock[ requireblockFreePointer ].requireName = name; //将进程名i填入请求块

                jta.append( "获得请求输出块" + Integer.toString( requireblockFreePointer +1 )  + "\n");

                requireblockFreePointer = ( requireblockFreePointer + 1 ) % 10;//修改空闲请求块指针

                requireblockNumber--;

                if( pcb[ 3 ].status == 2 ){ //若SPOOLING进程是等待状态，则唤醒SPOOLING进程

                    pcb[ 3 ].status = 0;

                }

            }

        }//判断进程所要输出的文件是否输出完毕的while循环结束

        jta.append( "进程" + name + "输出完毕！" );//文件输出完毕，修改状态为结束，转进程调度

        pcb[ name ].status = 4;

        return out;

    }

    public void spooling1(){

        while( requireblockNumber != 10 ){ //判断请求输出块是否为空的while循环

            //若请求输出块不为空

            StringBuffer stringBuffer = new StringBuffer( 100 );

            for ( i = 0; i<requireblock[ requireblockOutputPointer ].length; i++) {//按该请求输出信息块requireblock[]的指针requireblockOutputPointer将输出井中的一个文件的内容放入临时buffer1中

                stringBuffer.append( buffer[ requireblock[ requireblockOutputPointer ].requireName ][ requireblock[ requireblockOutputPointer ].outputHeadAddress] );

                requireblock[ requireblockOutputPointer ].outputHeadAddress = ( requireblock[ requireblockOutputPointer ].outputHeadAddress + 1 )%100;

            }

            if( requireblock[ requireblockOutputPointer ].requireName == 1){

                output\_1++;

                window.jta3.append( "User1第" + output\_1 + "个文件:");

            }

            else{

                output\_2++;

                window.jta3.append( "User2第" + output\_2 + "个文件:");

            }

            window.jta3.append( stringBuffer.toString() + "\n" );

            try{

                sleep( 500 );

            }

            catch( InterruptedException e ){

                e.printStackTrace();

            }

            //释放相应输出井，即修改相应的输出井计数outputBufferSpace

            outputBufferSpace[ requireblock[ requireblockOutputPointer ].requireName ] = outputBufferSpace[ requireblock[ requireblockOutputPointer ].requireName ] + requireblock[ requireblockOutputPointer ].length;

            requireblockOutputPointer = ( requireblockOutputPointer + 1 )%10;

            requireblockNumber++;

            for( int k = 1; k <= 2; k++ ){

                if( pcb[ k ].status == 3 ){ //有等待请求输出块的进程，唤醒没有请求块的进程   一次输出进程完成只能解禁一个请求块所以要完成一个就切断

                    //应该看先是否有等待请求块的进程，这样可以尽早释放相应的井空间

                    pcb[ k ].status = 0;

                    requireblock[ requireblockFreePointer ].outputHeadAddress = outputBufferPointer[ k ][ 1 ];//将文件在输出井的位置填入空闲请求块

                    if( outputBufferPointer[ k ][ 0 ] >= outputBufferPointer[ k ][ 1 ] ){ //将文件在输出井的长度填入空闲请求块

                        requireblock[ requireblockFreePointer ].length = outputBufferPointer[ k ][ 1 ];

                    }

                    else{

                        requireblock[ requireblockFreePointer ].length = 100 - outputBufferPointer[ k ][ 1] + outputBufferPointer[ k ][ 0 ];

                    }

                    requireblock[ requireblockFreePointer ].requireName = k;//将进程名i填入请求块

                    if( k ==1 ){

                        window.jta1.append("第" + output1 + "个文件获得请求输出块" + ( Integer.toString( requireblockFreePointer + 1 ) ) + "\n" );

                    }

                    if( k == 2 ){

                        window.jta2.append("第" + output2 + "个文件获得请求输出块" + ( Integer.toString( requireblockFreePointer + 1 ) ) + "\n" );

                    }

                    requireblockFreePointer = ( requireblockFreePointer +1 )%10;//修改空闲请求块指针

                    requireblockNumber--; //空闲请求块数减1

                    if( pcb[ 3 ].status == 2 ){ //若SPOOLING进程是等待状态，则唤醒SPOOLING进程

                        pcb[ 3 ].status = 0;

                    }

                    return; //完成一个就切断, 可能两个都等待请求块,但先满足一个吧

                }

            }

            for( int k =1; k <=2; k++ ){

                if( pcb[ k ].status ==1 ){ //有等待输出井的进程，唤醒相应进程

                    pcb[ k ].status = 0;

                    return;

                }

            }

        }//判断请求输出块是否为空的while循环结束

        if( pcb[ 1 ].status ==4 && pcb[ 2 ].status ==4 ){//进程1、2结束后输出进程结束

            pcb[ 3 ].status = 4;

            window.jta3.append( " Spooling输出进程结束" );

            return;

        }

        else{

            window.jta3.append( "调度spooling输出程序，请求块空，输出进程转入状态2\n" );

            pcb[ 3 ].status = 2;

        }

    }

}

# 使用说明

## 1.[视频演示](https://www.bilibili.com/video/BV1aK4y147zC?from=search&seid=3356019704329602222)

有图有真相。

## C:\Users\tony\Downloads\Splloing.jpg2.图文讲解

图4.2-1运行结果示意图

1、在“**User Process1文件数**”和“**UserProcess2文件数**”的文本框里填入要输出的文件数。

2、点击“**开始运行**”按钮开始运行。

3、右下角的“**主程序调度**”文本区会显示每次调度的进程。

4、左下角的“**Spooling调度输出**”文本区会把请求输出块的指针所示的信息输出。

5、左上角的“**用户进程1的文件生成情况及状态变换**”文本区会显示用户进程1每个文件的占用输出井和请求输出块的情况。

6、右上角的“**用户进程2的文件生成情况及状态变换**”文本区会显示用户进程2每个文件的占用输出井和请求输出块的情况。

7、点击“**重新开始**”按钮，回到最初界面。

8、点击“**退出程序**”按钮，关闭运行窗口。

# 五、结果分析

在“**User Process1文件数**”和“**UserProcess2文件数**”的文本框里分别填入3和8，点击“**开始运行**”按钮，得到的结果如上图4.2-1所示，从图中的数据可以分析出运行的顺序为：

1、主程序先选择调度用户进程1，“用户进程1的输出”显示第1～3个文件成功得到输出井资源buffer[1][0~23]和请求输出块资源requireblock[0~2]，进程1输出完毕，转调度函数。；然后调度用户进程2，“用户进程2的输出”显示第1～7个文件成功得到输出井资源buffer[2][0~60]和请求输出块资源requireblock[3~9]。第8个文件缺少requireblock，申请失败，进入等待状态3，转调度函数。

2、主程序选择调度用户进程1，“用户进程1的输出”显示第1个文件缺少requireblock，申请失败，进入等待状态3，转调度函数。

3、用户进程1、2都处于等待状态3，主程序只能选择调度Spooling进程，“Spooling的调度”显示第1～10个requireblock所指的信息，即用户进程1的第1～3个文件，用户进程2的1~7个文件，唤醒等待输出井和请求块的进程，用户进程1、2都变为执行状态0。

4、主程序选择调度用户进程2，“用户进程2的输出”显示第8个文件成功得到输出井资源buffer[2][61~70]和请求输出块资源requireblock[3]。进程1输出完毕，转调度函数。

6、用户进程1、2都处于结束状态4，主程序只能选择调度Spooling进程，“Spooling的调度”显示第1～4个requireblock所指的信息，即用户进程1的第1～3个文件和用户进程2的第11个文件。Requireblock为空且用户进程1、2都结束，所以Spooling进程变成结束状态4，转调度函数。

7、用户进程1、2和Spooling进程都处于结束状态4，主程序结束，“主程序调度过程”中显示“**程序结束**”。

由以上分析，可知程序能够正确运行。