华北电力大学

课 程 设 计 报 告

（ 2021--2022 年度第二学期）

课程名称：操作系统课程设计

课设题目：

院　　系：控制与计算机工程学院

班　　级：

姓　　名：

学　　号：

指导教师：

设计周数：一周

成　　绩：

2022 年　月　日

设计报告内容

一、题目

1.设计一个实现SP001ING技术的进程

要求设计一个 SPOOLING 输出进程和两个请求输出的用户进程，以及一个 SPOOLING 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 SPOOLING 进程进行输出。

SPOOLING 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，SPOOLING 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%，SPOOLING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”。
3. SPOOLING进程在进行输出时，若输出井空，则进入“等待状态2”。
4. SPOOLING 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SPOOLING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”

二、概要设计

软件分为两个部分, 逻辑代码区和图形界面 区,

1.图形界面

图形界面比较容易设计，只需要让其监视 逻辑代码 区域中的各种变量(数

据劫持)，并实时呈现出来即可

2.逻辑代码

我将该设计的题目理解为，使用软件去仿真一系列的硬件及软件，具体仿真的对象为

1.一个 CPU

这个 CPU 的作用为，运行 用户进程 userProcess 以及 Spooling 进程 spoolingProcess ，并且处理各种异常情况 Interrupt

2.一个 就绪队列 readyQ ，一个等待队列 wait1Q

这两个队列中存放各个尚未进入 CPU 的 userProcess

3.一个进程调度机关 ProcessController

这个机关负责在 readyQ 中取出 userProcess 投入至 CPU 中，负责与外界交互，即将进程添加到 readyQ 队列中

4.一组 用户进程 userProcess 和 一个 Spooling 进程 spoolingProcess

这些充当软件的作用，userProcess 调用 输出相关代码 spooling.spoolingOut2Cache 将自身的数据文件输出至缓冲区中

spoolingProcess 调用 输出相关代码 spooling.SpoolingOut2Io 将缓冲区文件输出至 io 设备中

更具体的来说，spoolingProcess 和 userProcess 本质上都是（继承自）process ，只是两者的代码区不同而已

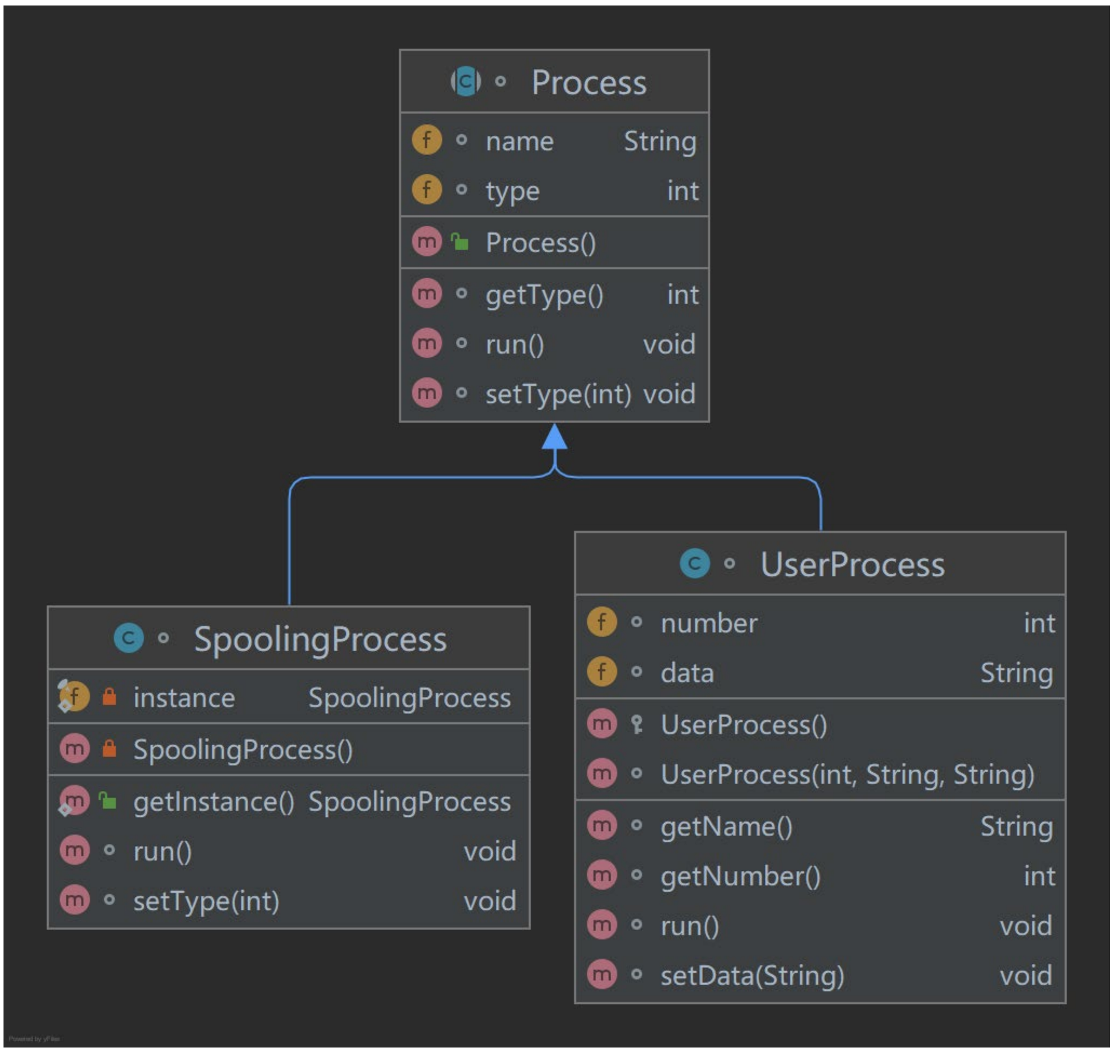
5.输出相关代码 spooling

类似于操作系统中 驱动 的身份，担任 process 与 缓冲区 和 输出设备 交互中介

三、整体功能及设计

1.Process

进程分为两类，一类是用户自定义的进程 userProcess 一类是 spoolingProcess 二者的关系如下



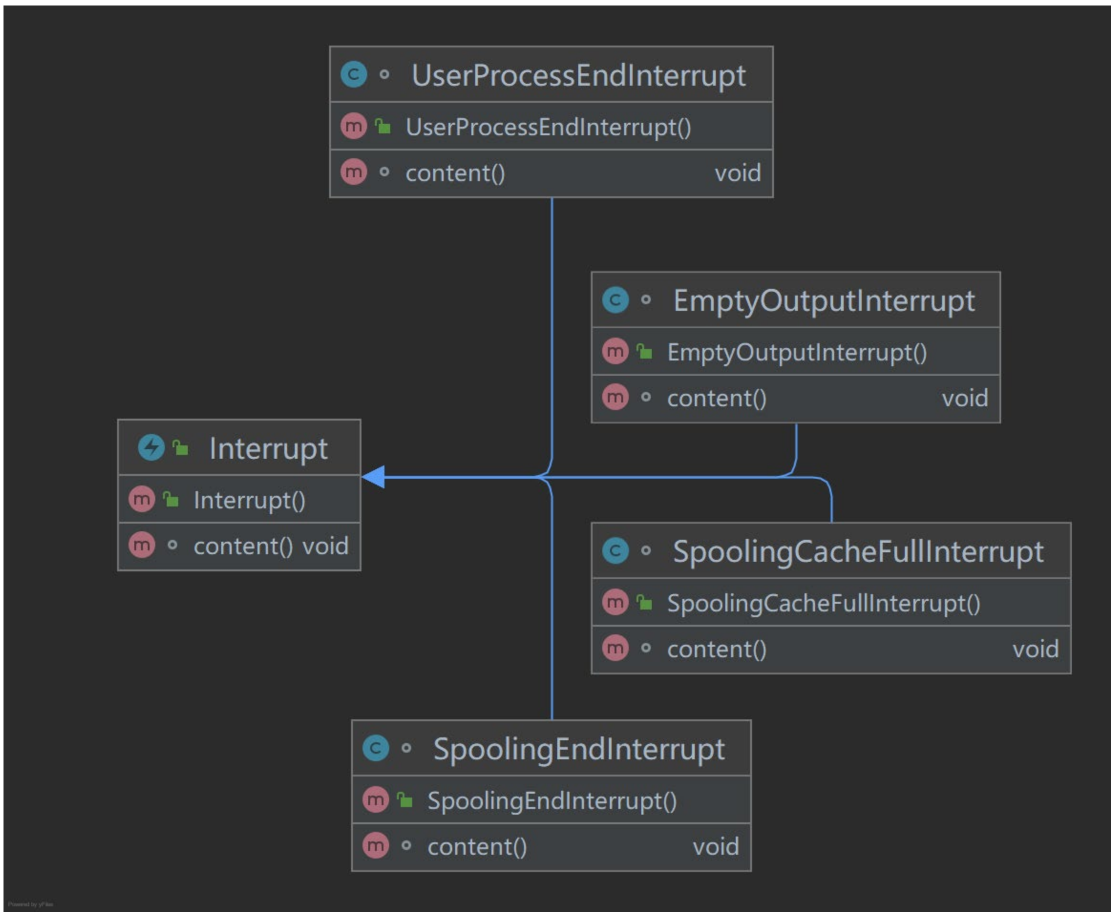
每个 process 实例使用 name 属性 进行区分.

run() 方法为 process 的代码区域, 当 CPU 拿到 process 之后, 会直接运行 run() 方法

2.process 相关 -- interrupt

interrupt 为中断区域, 在 process 执行期间, 遇到的各种异常情况(如输出缓冲区满, 程序运行结束 等情况) 需要发出中断, 指导 CPU 去处理相关异常情况.

本程序引入了四种中断



四种中断分别为:

1. 输出缓冲区空

此中断由 spoolingProcess 发出, 会将 spoolingProcess 的状态由 就绪态 转化为 wait_2 态

2. 输出缓冲区满

此中断由 输出相关代码(驱动) 发出, 会将当前 CPU 中执行的 userProcess 取出, 放入 wait_1 队列.

3. 进程结束

此中断由 userProcess 发出, 表示此进程已执行完毕(已经将需要输出的数据送至输出井), 更改 spoolingProcess 的状态为 就绪态 并指示 CPU 取下一进程

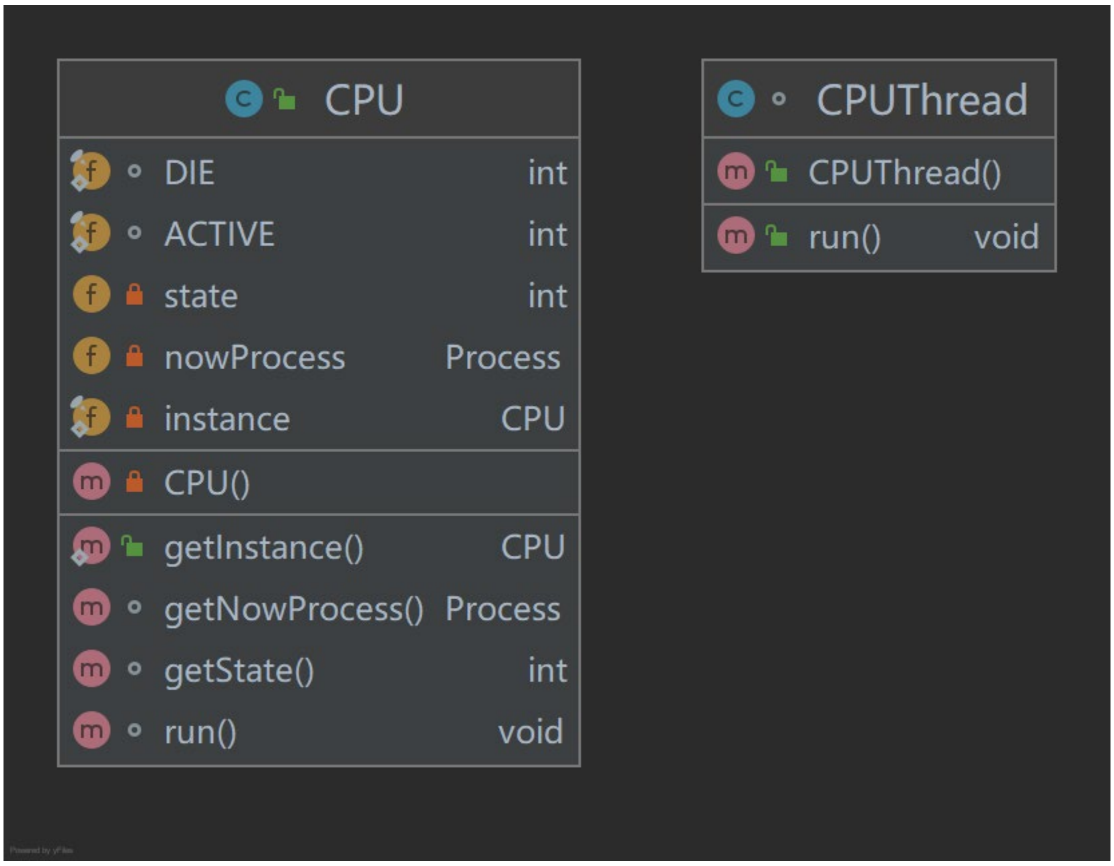
4. spooling 程序结束

此中断由 spoolingProcess 发出, 表示 spooling 进程已经将输出井中的一个缓存文件输出完毕, 会将所有等待的进程置于 就绪态

CPU 获取到中断后, 会自动执行中断的 content() 方法.

3. CPU

CPU 实际工作很简单, 向 进程调度机关 ProcessController 索要一个 process , 运行其代码区的代码, 接收中断并进行中断处理



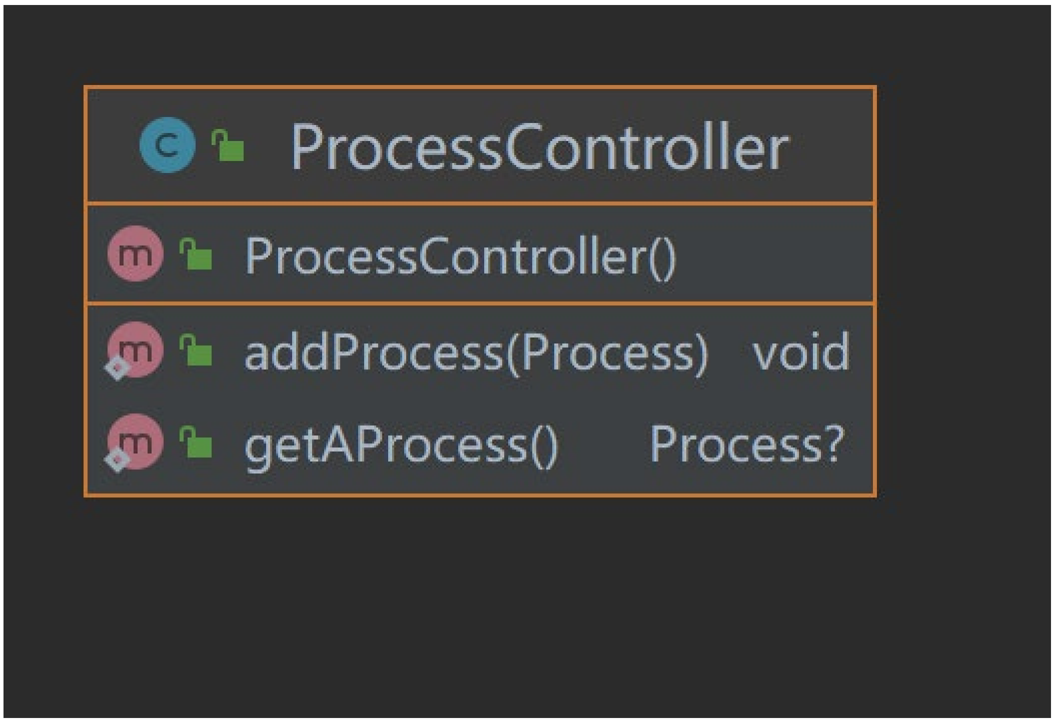
4. 进程调度机关 ProcessController

ProcessController 有两个功能

1. 根据随机数的值送出一个 process

2. 将某一新的 process 送入就绪队列

根据实际来看，该功能可能会在同一时间被同时调用，因此需要保证多线程安全



5. **输出相关代码 OutputController**

这个代码分为三部分

1. 输出控制器

该区域和两种 process 直接交互

userProcess 将自己的唯一标识 name 以及数据 data 送入，控制器负责将数据输出至缓冲区，并将得到的输出控制块挂上控制块队列

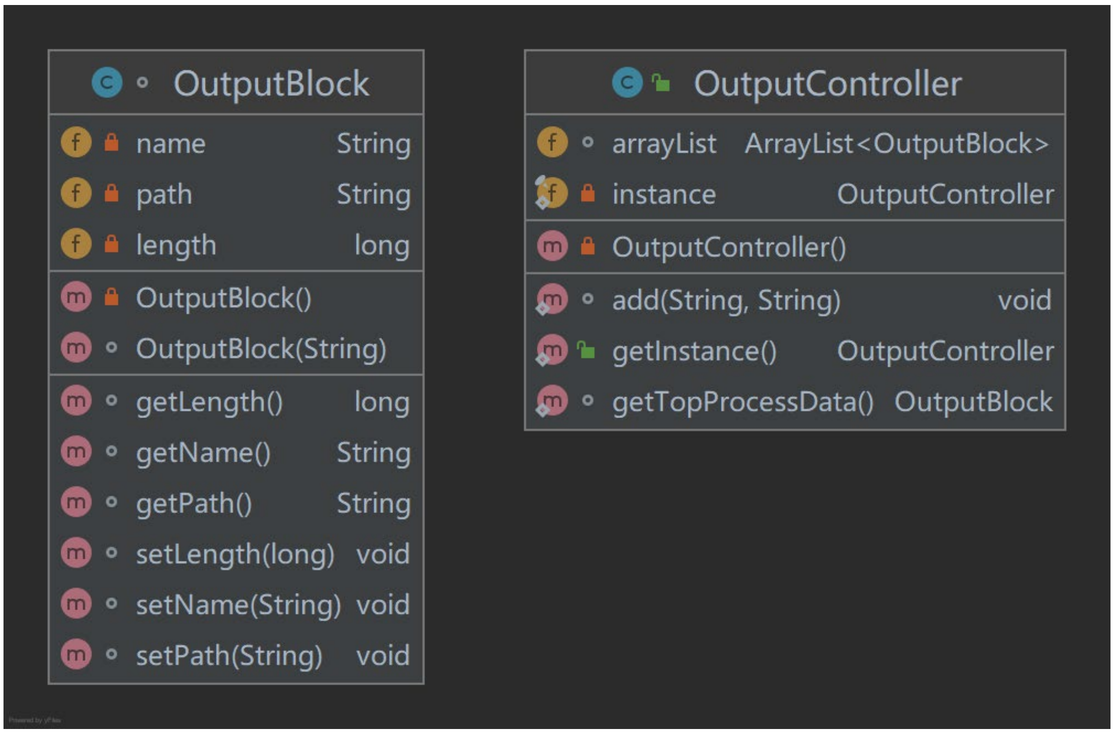
spoolingProcess 向控制器发出信号(调用函数)，控制器在输出控制块队列上取下某一个并进行 io 输出

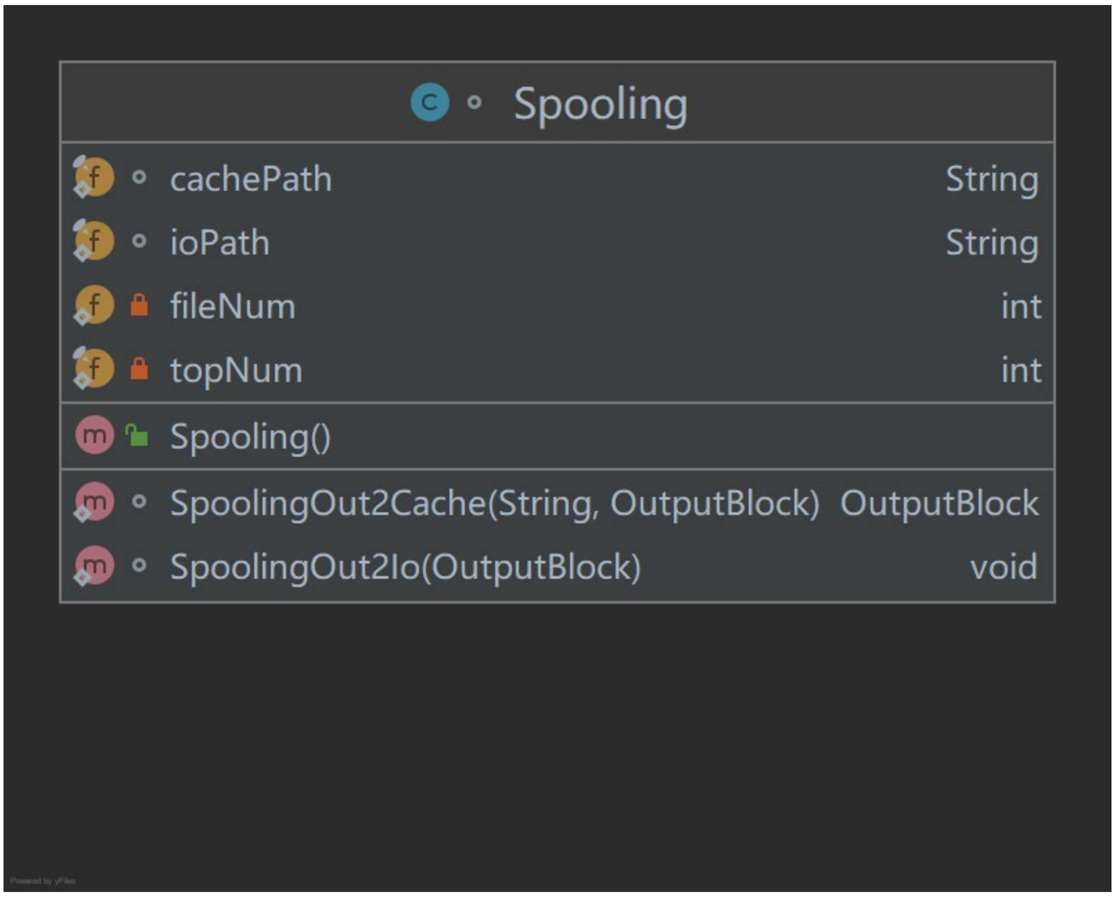
2. 输出至缓冲区部分

接收数据，返回控制块

3. 输出至 io 部分

接收控制块，开始输出


OutputBlock
name String
path String
length long
OutputBlock()
OutputBlock(String)
getLength() long
getName() String
getPath() String
setLength(long) void
setName(String) void
setPath(String) void
OutputController
arrayList ArrayList<OutputBlock>
instance OutputController
OutputController()
add(String, String) void
getInstance() OutputController
getTopProcessData() OutputBlock


Spooling
cachePath String
ioPath String
fileNum int
topNum int
Spooling()
SpoolingOut2Cache(String, OutputBlock) OutputBlock
SpoolingOut2Io(OutputBlock) void

6. 两个队列

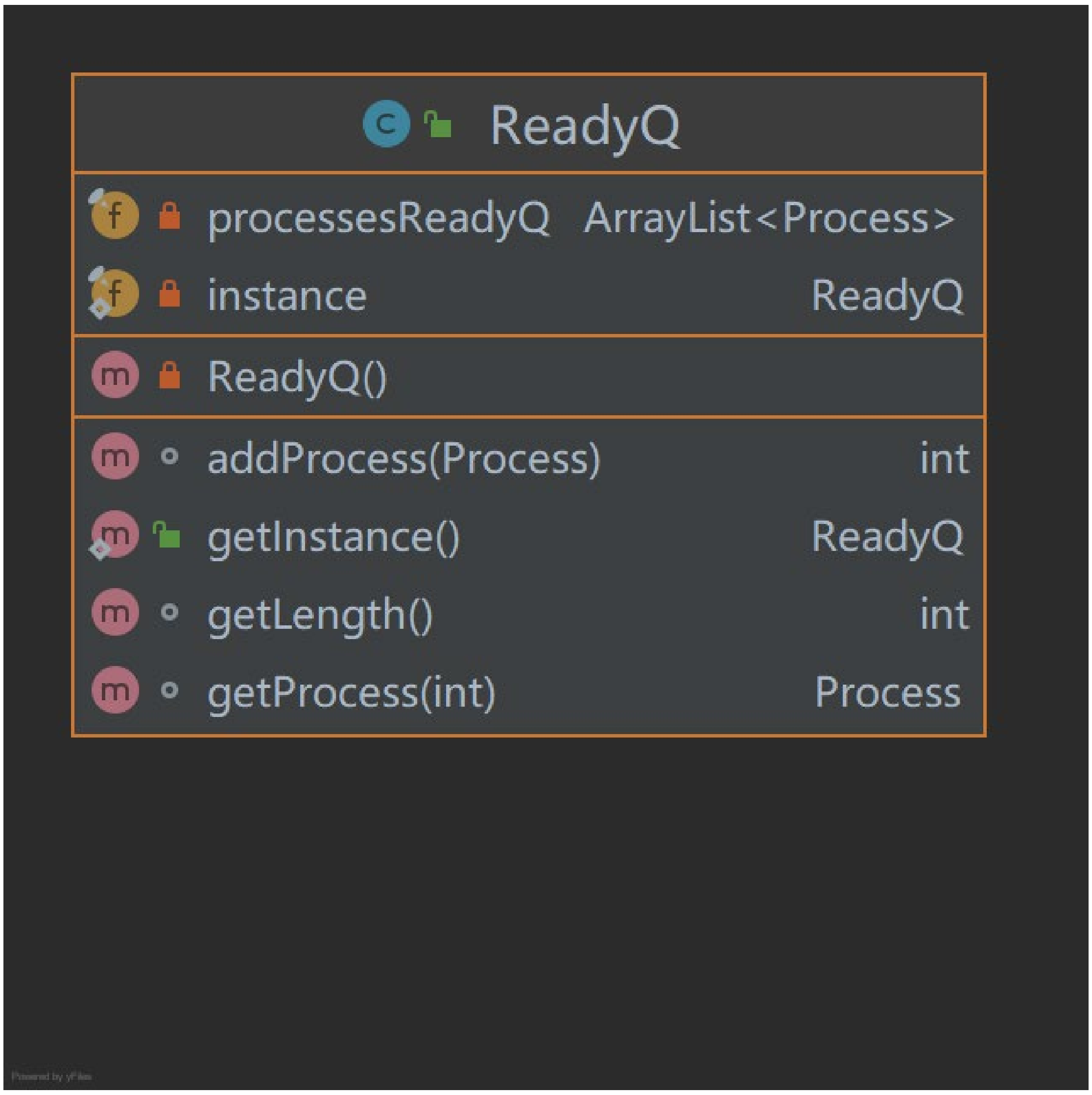
就绪队列以及等待队列，两者结构相似

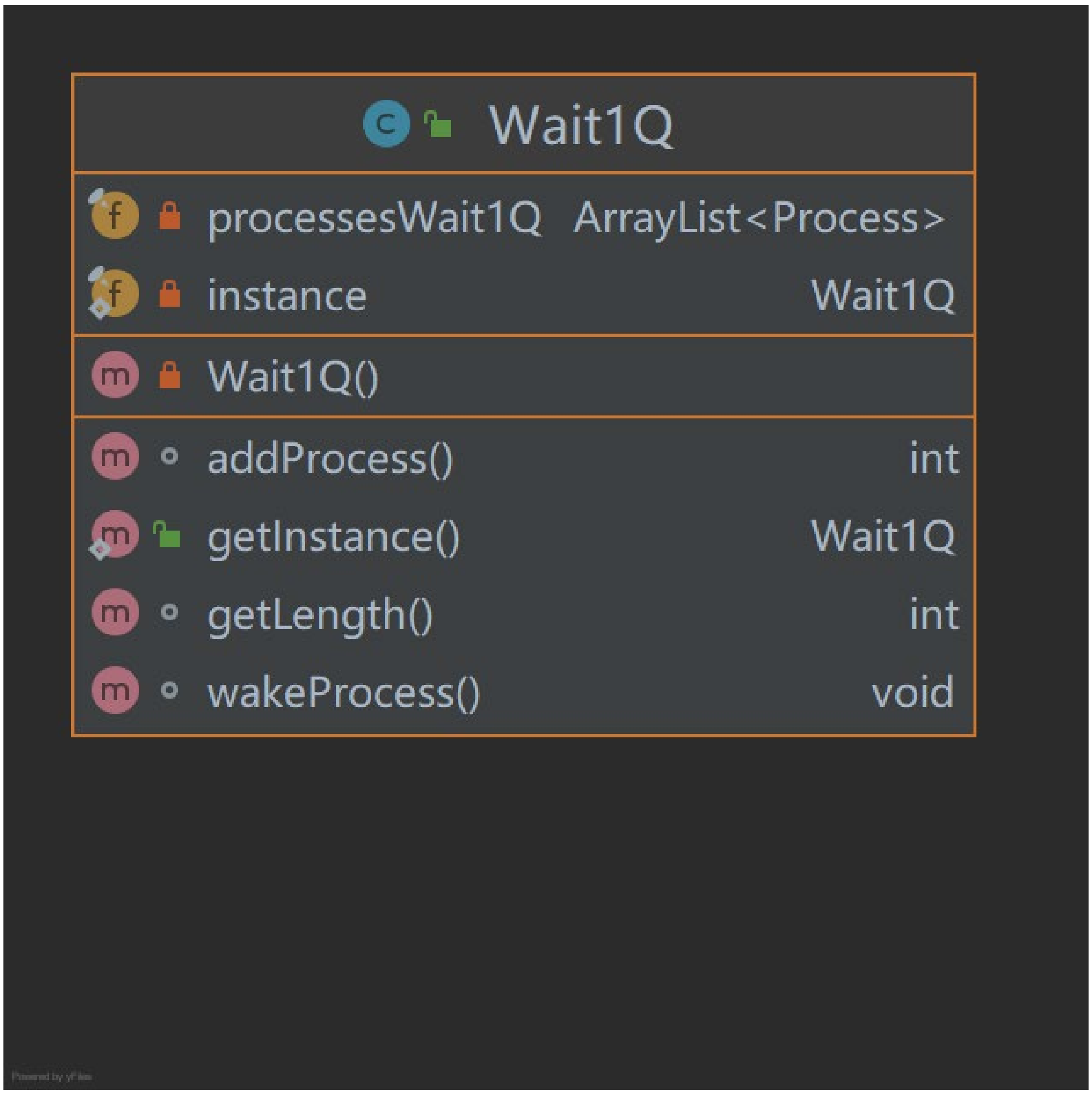
1. readyQ

添加进程以及取出进程

2. Wait1Q

添加进程，但不能逐个取出进程，所有进程同时被唤醒(wake)至就绪队列





四、编程实现

```
CPU.java


import java.util.concurrent.TimeUnit;


public class CPU {
    static final int DIE = 0;
    static final int ACTIVE = 1;
    private int state = DIE;
    private Process nowProcess;
    private static final CPU instance = new CPU();
    private CPU(){}
    public static CPU getInstance(){return instance;}

    void run(){
            Process process = ProcessController.getAProcess();

            nowProcess = process;

            if(process == null){
                System.out.println("CPU BREAK!!");
                return;
            }

            try {
                System.out.println("------Process.Process " +
process.name + " in cpu in in");

CPUPanel.getInstance().setInfo(process.getClass().getName(),pr
ocess.name);
                process.setType(Type.RUNNING);

                try {
                    TimeUnit.SECONDS.sleep(3);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                process.run();
            } catch (Interrupt e) {
                e.content();

                System.out.println("------Process.Process " +
process.name + " out cpu out out");
                CPUPanel.getInstance().setInfo("","");
                process.setType(Type.READY);
            }
    }

    Process getNowProcess(){
        return nowProcess;
    }

    int getState(){
        return state;
    }

}

class CPUThread implements Runnable{

    @Override
    public void run() {
        while (true){
            if(ReadyQ.getInstance().getLength() != 0 ||
SpoolingProcess.getInstance().type != Type.WAIT_2){
                CPU.getInstance().run();

                System.out.println("===readyQ length = " +
ReadyQ.getInstance().getLength());
            }
        }
    }
}
```

```
Process.java

import java.util.concurrent.TimeUnit;

abstract class Process {

    String name;
    int type;

    void run() throws Interrupt {}

    void setType(int type){
        this.type = type;
    }
    int getType(){
        return type;
    }
}

class UserProcess extends Process{
    int number;
    String data;

    UserProcess(int number, String name, String data){
        this.number = number;
        this.name = name;
        this.data = data;
        this.type = Type.READY;
    }

    protected UserProcess() {}

    @Override
    void run() throws Interrupt {
        super.run();

        OutputController.add(this.name,this.data);

        throw new UserProcessEndInterrupt();
    }

    int getNumber() {
        return number;
    }
    String getName() {
        return name;
    }
    void setData(String data) {
        this.data = data;
    }
}

class SpoolingProcess extends Process{
    private static final SpoolingProcess instance = new
SpoolingProcess();
    private SpoolingProcess(){
        this.type = Type.READY;
        this.name = "Spooling";
    }
    public static SpoolingProcess getInstance() {
        return instance;
    }



    @Override
    void run() throws Interrupt {
        super.run();

Spooling.SpoolingOut2Io(OutputController.getTopProcessData());
    }

    @Override
    void setType(int type) {
        super.setType(type);

        spoolingPanel.getInstance().flashJLabel(this.type);
    }
}
```

```
public class ProcessController {

    public static Process getAProcess(){
        int mooooooo = 2;
        // 这个变量将 Spooling 进程与普通进程被请求到的概率调整为 1 :
moooooooo

        ReadyQ readyQ = ReadyQ.getInstance();
        int random = RandomNumber.getNumber();
        Process spooling = SpoolingProcess.getInstance();

        if(spooling.type == Type.READY){
            random = random % (readyQ.getLength() * mooooooo + 1);
            if(random == 0)
                return spooling;
            else
                return readyQ.getProcess(random / 10);
        }
        else{
            if(readyQ.getLength() > 0)
                return readyQ.getProcess(random %
readyQ.getLength());
            else
                return null;
        }
    }

    public static void addProcess(Process process){

        ReadyQ readyQ = ReadyQ.getInstance();
        readyQ.addProcess(process);
    }
```

```
}
```

```
// 随机数生成


public class RandomNumber {
    public static int getNumber(){
        return (int)(100000 * Math.random());
    }
}
```

```
import java.util.ArrayList;


public class ReadyQ{
    private final ArrayList<Process> processesReadyQ;


    private static final ReadyQ instance = new ReadyQ();
    private ReadyQ (){
        processesReadyQ = new ArrayList<>();
    }
    public static ReadyQ getInstance() {
        return instance;
    }


    synchronized int getLength(){
        return processesReadyQ.size();
    }


    synchronized Process getProcess(int no){


        Process process = processesReadyQ.get(no);
        processesReadyQ.remove(no);

        System.out.println("readyQ remove out " + process.name);


readyQPanel.getInstance().flashTextArea(processesReadyQ);
        return process;
    }
    synchronized int addProcess(Process process){
        processesReadyQ.add(process);

        System.out.println("readyQ add " + process.name);


readyQPanel.getInstance().flashTextArea(processesReadyQ);
        return getLength();
    }
}
```

```
import java.util.ArrayList;

public class Wait1Q {
    private final ArrayList<Process> processesWait1Q;

    private static final Wait1Q instance = new Wait1Q();
    private Wait1Q (){
        processesWait1Q = new ArrayList<>();
    }
    public static Wait1Q getInstance() {
        return instance;
    }

    int getLength(){
        return processesWait1Q.size();
    }

    void wakeProcess(){
        ReadyQ readyQ = ReadyQ.getInstance();
        for(int i = 0; i < processesWait1Q.size(); i++){
            readyQ.addProcess(processesWait1Q.get(i));
        }
        processesWait1Q.clear();


waitQPanel.getInstance().flashTextArea(processesWait1Q);
        System.out.println("wait1Q clear!!");

    }

    int addProcess(){
        Process process = CPU.getInstance().getNowProcess();
        processesWait1Q.add(process);

        System.out.println("wait1Q add " + process.name);


waitQPanel.getInstance().flashTextArea(processesWait1Q);
        return getLength();
    }
}
```

```
public class Type {
    public final static int WAIT_1 = 1;
    public final static int WAIT_2 = 2;
    public final static int READY = 3;
    public final static int RUNNING = 4;
}
```

```
public class Interrupt extends Exception{
    void content(){}
}

class EmptyOutputInterrupt extends Interrupt{
    public EmptyOutputInterrupt(){
        super();
    }

    void content(){
        SpoolingProcess spoolingProcess =
SpoolingProcess.getInstance();
        spoolingProcess.setType(Type.WAIT_2);

        //spoolingPanel.getInstance().flashJLabel(Type.WAIT_2);
    }
}

class SpoolingEndInterrupt extends Interrupt{
    public SpoolingEndInterrupt(){
        super();
    }

    void content(){
        Wait1Q wait1Q = Wait1Q.getInstance();
        wait1Q.wakeProcess();
    }
}

class SpoolingCacheFullInterrupt extends Interrupt{
    public SpoolingCacheFullInterrupt() {super();}

    void content(){
        Wait1Q.getInstance().addProcess();
    }
}

class UserProcessEndInterrupt extends Interrupt{
    public UserProcessEndInterrupt(){super();}

    void content(){
        SpoolingProcess.getInstance().setType(Type.READY);

        //spoolingPanel.getInstance().flashJLabel(Type.READY);
    }
}
```

```
import java.util.ArrayList;

// 需要大改，把 Arraylist 改成数组，因为需要固定位置
// 不改了，再做一个 spooling 向输出井中的输出程序吧

```

```
public class OutputController {
    ArrayList<OutputBlock> arrayList;

    private static final OutputController instance = new
OutputController();
    private OutputController(){
        arrayList = new ArrayList<>();
    }
    public static OutputController getInstance() {
        return instance;
    }

    static void add(String name, String data) throws Interrupt {
        OutputBlock outputBlock = new OutputBlock(name);
        outputBlock =
Spooling.SpoolingOut2Cache(data,outputBlock);
        instance.arrayList.add(outputBlock);


outCachePanel.getInstance().flashTextArea(instance.arrayList);
    }

    static OutputBlock getTopProcessData() throws
EmptyOutputInterrupt {
        if(instance.arrayList.size() > 0) {
            OutputBlock outputBlock = instance.arrayList.get(0);
            instance.arrayList.remove(0);


outCachePanel.getInstance().flashTextArea(instance.arrayList);
```

```
            return outputBlock;
        }
        else{
            throw new EmptyOutputInterrupt();
        }
    }

}

class OutputBlock{
    private String name;
    private String path;
    private long length;

    OutputBlock(String name){
        this.name = name;
    }

    String getName(){
        return name;
    }
    String getPath(){
        return path;
    }
    long getLength() {return length;}

    void setName(String name){
        this.name = name;
    }
    void setPath(String path){
        this.path = path;
    }
    void setLength(long length) {this.length = length;}

    private OutputBlock(){}
}
```

```
import java.io.*;

/*
* Spooling 程序有两个功能，
* 一个是向缓冲区输出 SpoolingOut2Cache
* 一个是向 io 设备输出 SpoolingOut2Io
* */

class Spooling {

    static final String cachePath = ".\\spoolingCache\\";
    static final String ioPath = ".\\spoolingIo\\";

    private static int fileNum = 0;
    private static final int topNum = 3;

    static OutputBlock SpoolingOut2Cache(String data, OutputBlock
outputBlock) throws SpoolingCacheFullInterrupt {
        if(fileNum > topNum){
            throw new SpoolingCacheFullInterrupt();
        }

        String fileName = outputBlock.getName();
        File file = new File(cachePath + fileName + ".cacche");
        FileWriter fileWriter = null;
        try {
            fileWriter = new FileWriter(file);
            fileWriter.write("");
            fileWriter.write(data);
            fileWriter.flush();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        outputBlock.setPath(cachePath + fileName + ".cacche");
        outputBlock.setLength(file.length());
        fileNum ++;

        System.out.println("Spooling Cache add " + fileName );

        return outputBlock;
    }

    static void SpoolingOut2Io(OutputBlock outputBlock) throws
SpoolingEndInterrupt {
        File file = new File(outputBlock.getPath());
        FileReader fileReader = null;
        try {
            fileReader = new FileReader(file);
            BufferedReader bufferedReader = new
BufferedReader(fileReader);
            String data = bufferedReader.readLine();
            bufferedReader.close();
            fileReader.close();
            file.delete();

            file = new File(ioPath + outputBlock.getName() +
".txt");
            FileWriter fileWriter = new FileWriter(file);
            fileWriter.write("");
            fileWriter.write(data);
            fileWriter.flush();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        fileNum--;

        System.out.println("Spooling io add " + file.getName());

        throw new SpoolingEndInterrupt();

    }
}
```

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;


public class Demo {
    Demo(){
        JFrame jFrame = new JFrame();
        jFrame.setLayout(new FlowLayout());

        jFrame.add(new ProcessInPanel().getjPanel());
        jFrame.add(CPUPanel.getInstance().getjPanel());
        jFrame.add(spoolingPanel.getInstance().getjPanel());
        jFrame.add(readyQPanel.getInstance().getjPanel());
        jFrame.add(waitQPanel.getInstance().getjPanel());
        jFrame.add(outCachePanel.getInstance().getjPanel());

        jFrame.setDefaultCloseOperation(3);
        jFrame.setSize(1800,600);
        jFrame.setVisible(true);
    }
}

class ProcessInPanel{
    private JPanel jPanel;

    ProcessInPanel(){
        this.jPanel = new JPanel();
        jPanel.setBorder(BorderFactory.createTitledBorder("创建
用户进程"));

        jPanel.setLayout(new GridLayout(3,2));

        jPanel.add(new JLabel("name"));
        JTextField name = new JTextField();
        jPanel.add(name);
        jPanel.add(new JLabel("data"));
        JTextField data = new JTextField();
        jPanel.add(data);

        JButton add = new JButton("add Process");
        add.addActionListener(e -> {
            (new Add(name.getText(),data.getText())).start();
        });
        jPanel.add(add);
    }

    JPanel getjPanel(){
        return jPanel;
    }
}

class Add implements Runnable{

    private Thread t;
    private String name;
    private String data;

    Add(String name, String data){
        this.name = name;
        this.data = data;
    }

    @Override
    public void run() {
        UserProcess userProcess = new UserProcess(0,name,data);
        ProcessController.addProcess(userProcess);
    }

    void start(){
        if(t == null) {
            t = new Thread(this, name);
            t.start();
        }
    }
}

class CPUPanel{
    private JPanel jPanel;
    private JTextArea processInfo;

    private static final CPUPanel instance = new CPUPanel();
    public static CPUPanel getInstance(){return instance;}

    private CPUPanel(){
        jPanel = new JPanel();
        jPanel.setBorder(BorderFactory.createTitledBorder("C P
U"));

        processInfo = new JTextArea(5,30);
        processInfo.setEnabled(false);
        jPanel.add(processInfo);
    }

    public void setInfo(String ClassName, String ProcessName){
        processInfo.setText("");
        processInfo.append(ClassName + System.lineSeparator());
        processInfo.append(ProcessName +
System.lineSeparator());
    }

    JPanel getjPanel(){
        return jPanel;
    }
}

class readyQPanel{
    private JPanel jPanel;
    private JTextArea jTextArea;

    private static final readyQPanel instance = new readyQPanel();
    public static readyQPanel getInstance(){return instance;}

    private readyQPanel(){
        jPanel = new JPanel();
        jPanel.setBorder(BorderFactory.createTitledBorder("就 绪
队 列"));

        jTextArea = new JTextArea(30,30);
        jTextArea.setEnabled(false);
        jPanel.add(jTextArea);
    }

    JPanel getjPanel(){
        return jPanel;
    }

    void flashTextArea(ArrayList<Process> arrayList){
        jTextArea.setText("----------" +
System.lineSeparator());

        for (Process process : arrayList) {
            UserProcess userProcess = (UserProcess) process;
            jTextArea.append("---name : " + userProcess.name + "
---data :" + userProcess.data + System.lineSeparator() +
"----------" + System.lineSeparator());
        }
    }
}

class waitQPanel{
    private JPanel jPanel;
    private JTextArea jTextArea;

    private static final waitQPanel instance = new waitQPanel();
    public static waitQPanel getInstance(){return instance;}

    private waitQPanel(){
        jPanel = new JPanel();
        jPanel.setBorder(BorderFactory.createTitledBorder("等 待
1 队 列"));

        jTextArea = new JTextArea(30,30);
        jTextArea.setEnabled(false);
        jPanel.add(jTextArea);
    }

    JPanel getjPanel(){
        return jPanel;
    }

    void flashTextArea(ArrayList<Process> arrayList){
        jTextArea.setText("----------" +
System.lineSeparator());

        for (Process process : arrayList) {
            UserProcess userProcess = (UserProcess) process;
            jTextArea.append("---name : " + userProcess.name + "
---data :" + userProcess.data + System.lineSeparator() +
"----------" + System.lineSeparator());
        }
    }
}

class outCachePanel{
    private JPanel jPanel;
    private JTextArea jTextArea;

    private static final outCachePanel instance = new
outCachePanel();
    public static outCachePanel getInstance(){return instance;}

    private outCachePanel(){
        jPanel = new JPanel();
        jPanel.setBorder(BorderFactory.createTitledBorder("输 出
缓 冲 块 队 列"));

        jTextArea = new JTextArea(30,30);
        jTextArea.setEnabled(false);
        jPanel.add(jTextArea);
    }

    JPanel getjPanel(){
        return jPanel;
    }

    void flashTextArea(ArrayList<OutputBlock> arrayList){
        jTextArea.setText("----------" +
System.lineSeparator());

        for (OutputBlock outputBlock : arrayList) {
            jTextArea.append("---name : " + outputBlock.getName()
+ "     ---path :" + outputBlock.getPath() +
System.lineSeparator()
                    + "     ---length :" + outputBlock.getLength()
                    + System.lineSeparator() + "----------" +
System.lineSeparator());
        }
    }
}

class spoolingPanel{
    private JPanel jPanel;
    private JTextArea jTextArea;
    private static final spoolingPanel instance = new 
spoolingPanel();
    public static spoolingPanel getInstance(){return instance;}

    private spoolingPanel(){
        jPanel = new JPanel();

jPanel.setBorder(BorderFactory.createTitledBorder("spooling 状
态"));

        jPanel.setLayout(new GridLayout(2,1));
        jPanel.add(new JLabel("状 态"));
        jTextArea = new JTextArea(1,10);
        jTextArea.setText("READY");
        jPanel.add(jTextArea);
    }

    JPanel getjPanel(){
        return jPanel;
    }

    void flashJLabel(int type){

        if(type == Type.READY)
            jTextArea.setText("READY");
        else if(type == Type.WAIT_2)
            jTextArea.setText("WAIT_2");
        else  if(type == Type.RUNNING)
            jTextArea.setText("RUNNING");
    }
}
```

```
public class test {
    public static void main(String [] args){

        try {
            for(int i = 0 ; i < 10 ; i++){
                (new Add(i + "" + i + "" + i, "qbctese" + i)).start();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        Thread cpuT = new Thread(new CPUThread(),"cpu");
        cpuT.start();

        Demo demo = new Demo();

    }
}
```

五、使用说明

运行 test.java 中的 main 函数即可