**武汉大学计算机学院**

**本科生实验报告**

**操作系统设计与实现**

专 业 名 称 ：软件工程

课 程 名 称 ：操作系统设计

指 导 教 师 ：曾平

学 生 学 号 ：2018302110103

学 生 姓 名 ：郑志强

二○二〇年五月

**郑 重 声 明**

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 郑志强 日期： 2020/5/16

**摘 要**

处理器调度实验的实验目的是模拟在单处理器环境下的处理器调度，加深了解处理器调度的工作。

实验内容主要包括：选择一个调度算法，实现处理器调度。

**关键词：**单处理器环境；处理器调度；

主存空间的分配和回收的实验目的是通过本实习帮助理解在不同的存储管理方式下应怎样进行存储空间的分配和回收。

实验内容主要包括：主存储器空间的分配和回收

**关键词：**主存的分配和回收；

磁盘存储空间的分配和回收的实验目的是通过本实习使学生掌握磁盘存储空间的分配和回收算法。

实验内容主要包括：模拟磁盘空闲空间的表示方法，以及模拟实现磁盘空间的分配和回收。

**关键词：**磁盘空间的分配和回收；

进程创建的实验目的是了解进程的创建过程，进一步理解进程的概念，明确进程和程序的区别。

实验内容主要包括：利用fork()系统调用创建进程。

**关键词：**进程的创建；fork()函数；

进程同步的实验目的是模拟P、V操作同步机构的实现，模拟进程的并发执行，了解进程并发执行时同步机构的作用。

实验内容主要包括：模拟实现同步机构，以避免发生进程执行时可能出现的与时间有关的错误。

**关键词：**进程的并发执行；P、V操作同步机构的实现；

**目 录**

**1 处理器调度**

1.1 实习内容及上机实验所用平台………………………………………5

1.2 数据结构及代码段分析 ………………………………………………5

1.3 调试过程…………………………………………………………………6

1.4 实验总结…………………………………………………………………7

**2 主存空间的分配和回收**

2.1 实习内容及上机实验所用平台………………………………………8

2.2 数据结构及代码段分析 ………………………………………………8

2.3 调试过程…………………………………………………………………9

2.4 实验总结…………………………………………………………………10

**3 磁盘存储空间的分配和回收**

3.1 实习内容及上机实验所用平台………………………………………11

3.2 数据结构及代码段分析 ………………………………………………11

3.3 调试过程…………………………………………………………………13

3.4 实验总结…………………………………………………………………17

**4 进程创建**

4.1 实习内容及上机实验所用平台………………………………………18

4.2 数据结构及代码段分析 ………………………………………………18

4.3 调试过程…………………………………………………………………19

4.4 实验总结…………………………………………………………………19

**5 进程同步**

5.1 实习内容及上机实验所用平台………………………………………20

5.2 数据结构及代码段分析 ………………………………………………25

5.3 调试过程…………………………………………………………………29

5.4 实验总结…………………………………………………………………29

**6 附录**…………………………………………………………………………30

1. **处理器调度**
   1. **实习内容及上机实验所用平台**

**1.1.1 实验内容**

第一题：设计一个按优先数调度算法实现处理器调度的程序。

[提示]：

(1)假定系统有5个进程，每个进程用一个PCB来代表。PCB的结构为：

·进程名——如P1~P5。

·指针——按优先数的大小把5个进程连成队列，用指针指出下一个进程PCB的首地址。

·要求运行时间——假设进程需要运行的单位时间数。

·优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。

·状态——假设两种状态：就绪和结束，用R表示就绪，用E表示结束。初始状态都为就绪状态。

(2) 开始运行之前，为每个进程确定它的“优先数”和“要求运行时间”。通过键盘输入这些参数。

(3) 处理器总是选择队首进程运行。采用动态改变优先数的办法，进程每运行1次，优先数减1，要求运行时间减1。

(4) 进程运行一次后，若要求运行时间不等于0，则将它加入就绪队列，否则，将状态改为“结束”，退出就绪队列。

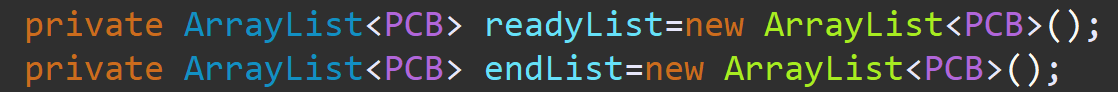
(5) 若就绪队列为空，结束，否则转到(3)重复。

要求能接受键盘输入的进程优先数及要求运行时间，能显示每次进程调度的情况，如哪个进程在运行，哪些进程就绪，就绪进程的排列情况。

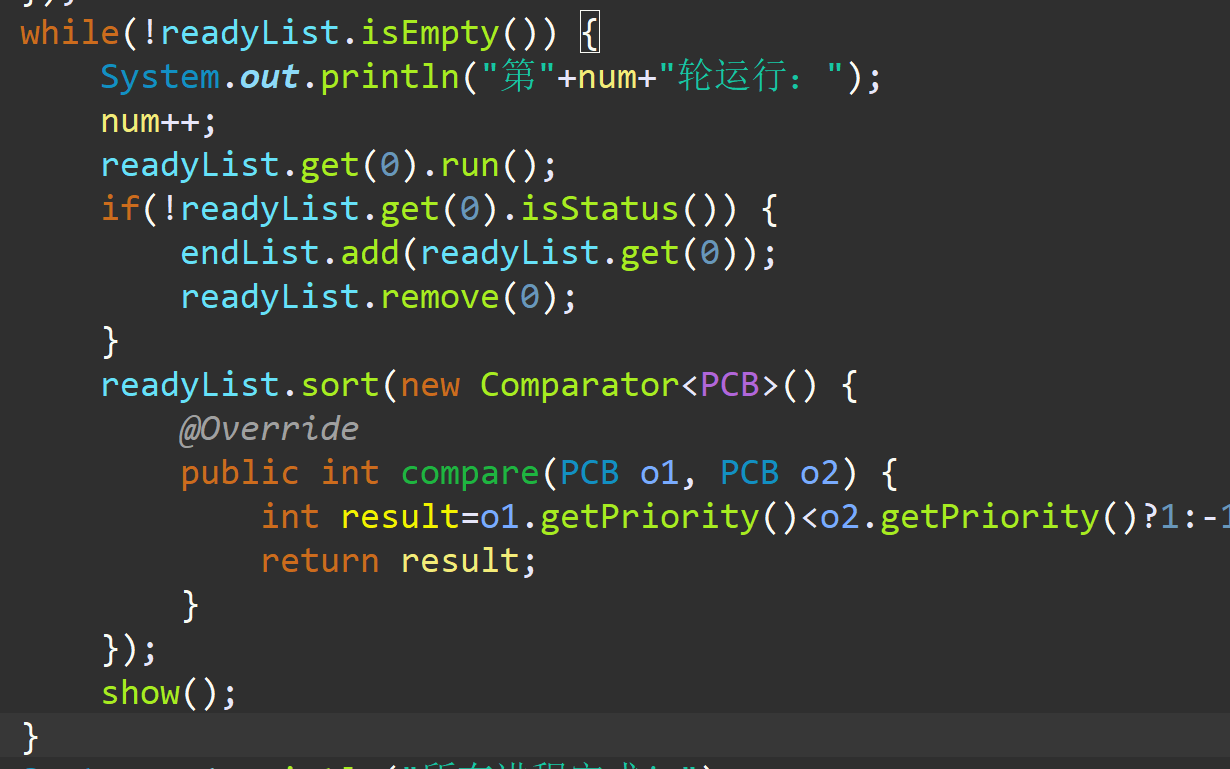
**1.1.2 上机实验所用平台**

Windows 10 Eclipse Jee 2019-09 Java 1.8

**1.2 数据结构及代码段分析**

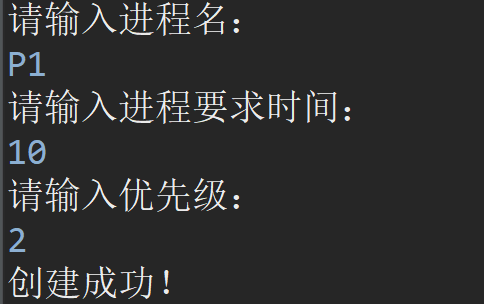


本实验采用两个arraylist来模拟就绪队列和完结队列，就绪进程储存在就绪队列中，完结进程储存在完结队列中。

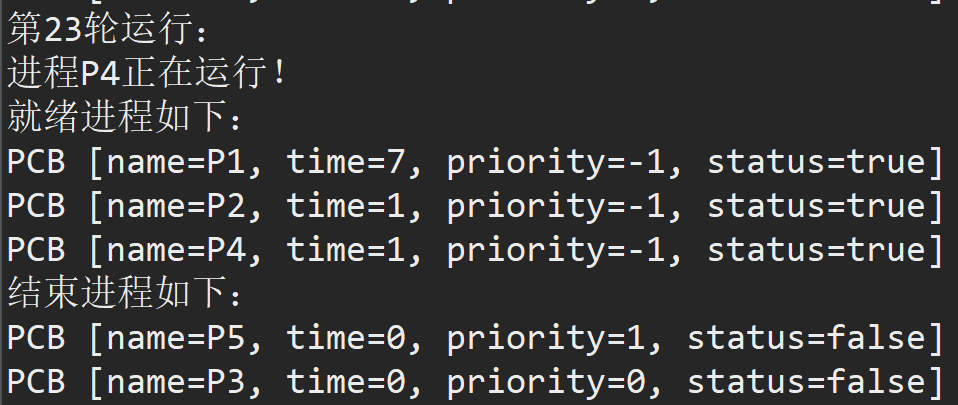


上面为程序运行的主要函数：进程在跑完一次后如果进程时间耗尽，则进入完结队列。每跑完一次后，都会对就绪队列中的进程按优先级从大到小重新排序。直至就绪队列所有进程结束，程序停止。

**1.3 调试过程**



成功创建进程



每一轮运行由正在运行的进程，就绪队列和结束队列组成。

**1.4 实验总结**

经过多组数据测试，实验结果符合预期。本实验说明用arraylist来模拟就绪队列和完结队列是可行的。本实验成功地按优先数调度算法实现了处理器调度。

1. **主存空间的分配和回收**
   1. **实习内容及上机实验所用平台**
      1. **实验内容**

第一题：可变分区管理方式下采用首次适应算法实现主存分配和回收

[提示]：

(1)可变分区方式是按作业需要的主存空间大小来分割分区的。当要装入一个作业时，根据作业需要的主存容量查看是否有足够的空闲空间，若有，则按需分配，否则，作业无法装入。假定内存大小为128K，空闲区说明表格式为：

·分区号——表示是第几个空闲分区；

·起始地址——指出空闲区的起始地址；

·长度——一个连续空闲区的长度；

(2)采用首次适应算法分配回收内存空间。运行时，输入一系列分配请求和回收请求。

要求能接受来自键盘的空间申请及释放请求，能显示分区分配及回收后的内存布局情况。

* + 1. **上机实验所用平台**

Windows 10 Eclipse Jee 2019-09 Java 1.8

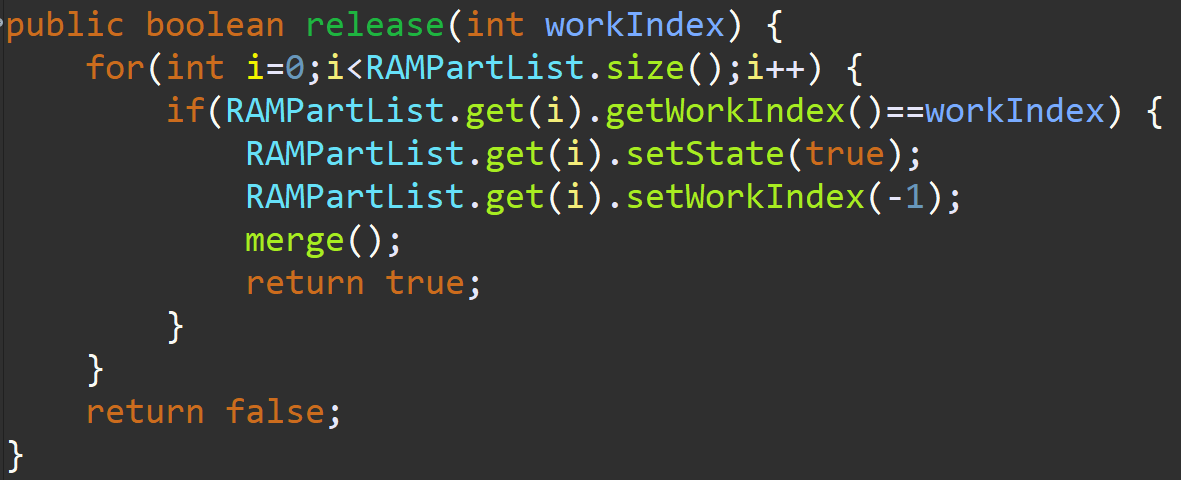
* 1. **数据结构及代码段分析**



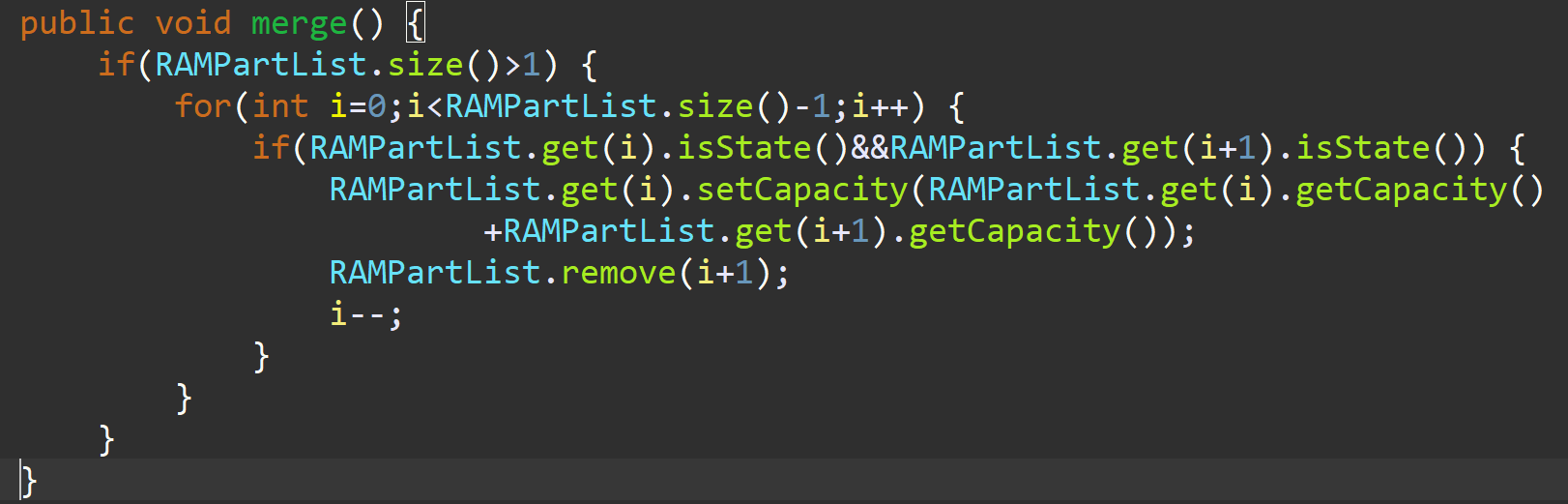
本实验采用linkedlist来模拟内存，RAMPart则是模拟每一个内存块



采用首次适应算法申请内存代码如上

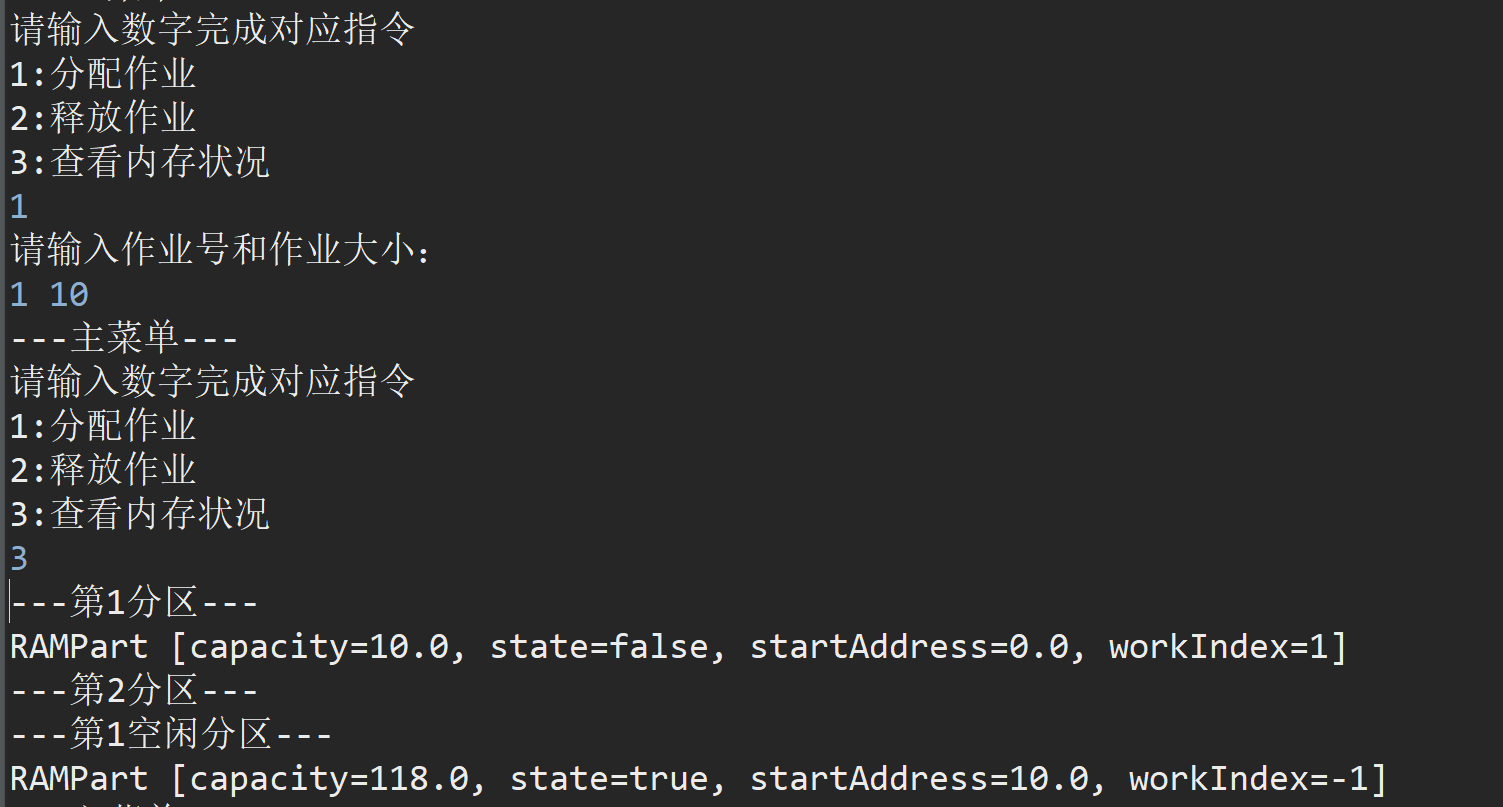


采用首次适应算法释放内存代码如上

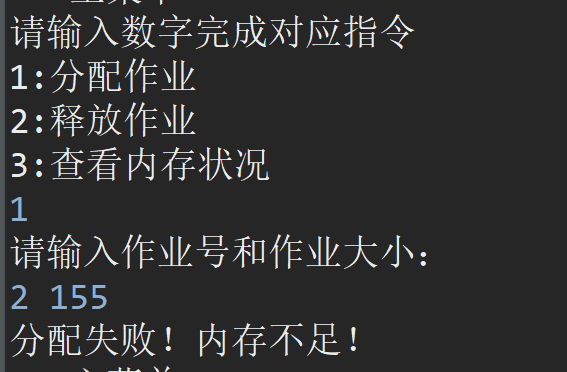


相邻空闲内存块间的合并算法

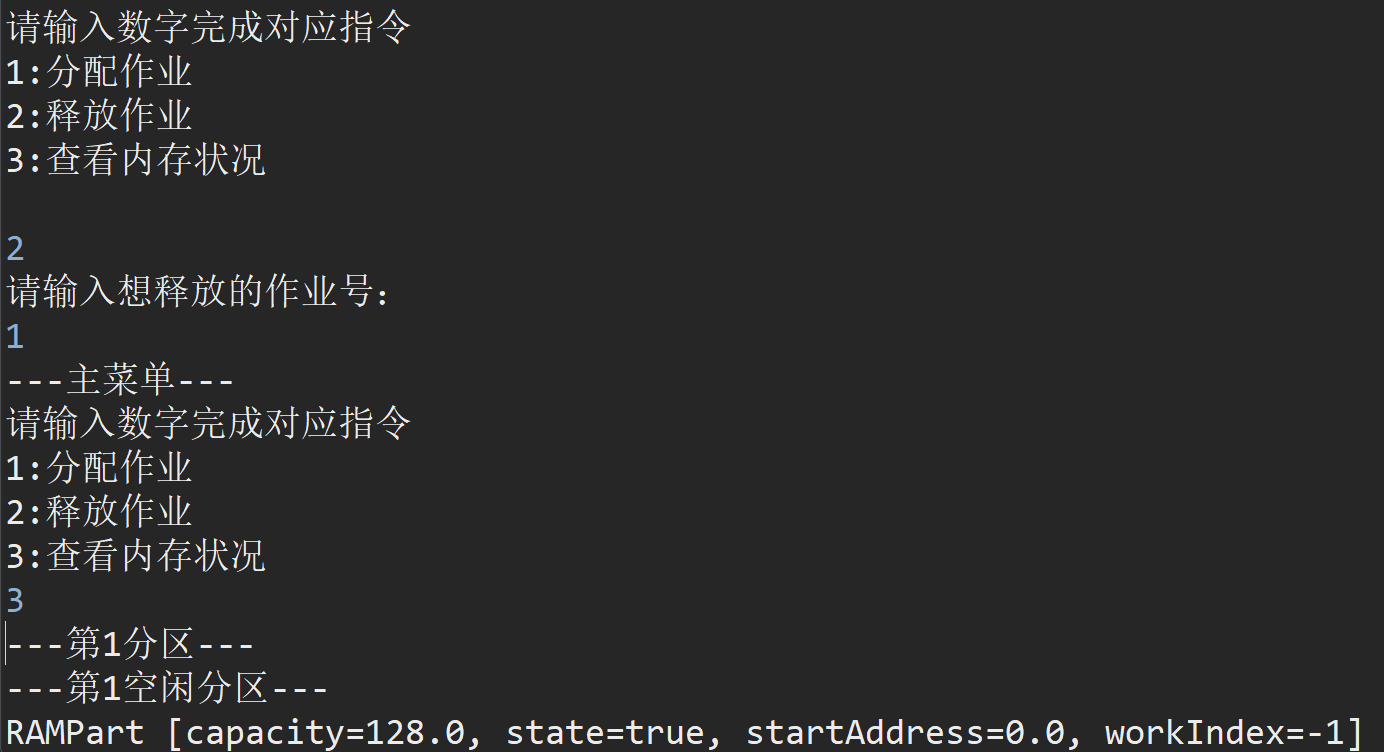
* 1. **调试过程**



成功分配内存



当内存不足时分配失败



成功释放内存

* 1. **实验总结**

本实验采用linkedlist来模拟内存，经过多组数据测试，实验结果均符合预期。本实验成功地在可变分区管理方式下采用首次适应算法实现主存分配和回收。

**3 磁盘存储空间的分配和回收**

3.1 **实习内容及上机实验所用平台**

3.1.1 **实验内容**

第二题：用位示图管理磁盘存储空间

[提示]：

(1) 为了提高磁盘存储空间的利用率，可在磁盘上组织成链接文件、索引文件，这类文件可以把逻辑记录存放在不连续的存储空间。为了表示哪些磁盘空间已被占用，哪些磁盘空间是空闲的，可用位示图来指出。位示图由若干字节构成，每一位与磁盘上的一块对应，“1”状态表示相应块已占用，“0”状态表示该块为空闲。位示图的形式与实习二中的位示图一样，但要注意，对于主存储空间和磁盘存储空间应该用不同的位示图来管理，绝不可混用。

(2) 申请一块磁盘空间时，由分配程序查位示图，找出一个为“0”的位，计算出这一位对应块的磁盘物理地址，且把该位置成占用状态“1”。假设现在有一个盘组共8个柱面，每个柱面有2个磁道（盘面），每个磁道分成4个物理记录。那么，当在位示图中找到某一字节的某一位为“0”时，这个空闲块对应的磁盘物理地址为：

柱面号=字节号

磁道号= 位数 / 4

物理记录号= 位数 % 4

(3) 归还一块磁盘空间时，由回收程序根据归还的磁盘物理地址计算出归还块在位示图中的对应位，把该位置成“0”。按照（2）中假设的盘组，归还块在位示图中的位置计算如下：

字节号=柱面号

位数=磁道号\*4+物理记录号

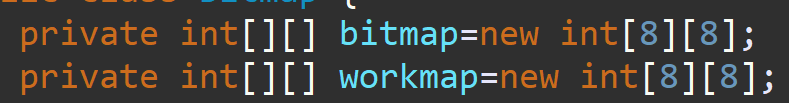
(4) 设计申请磁盘空间和归还磁盘空间的程序。

要求能接受来自键盘的空间申请及释放请求，要求能显示或打印程序运行前和运行后的位示图；分配时把分配到的磁盘空间的物理地址显示或打印出来，归还时把归还块对应于位示图的字节号和位数显示或打印出来。

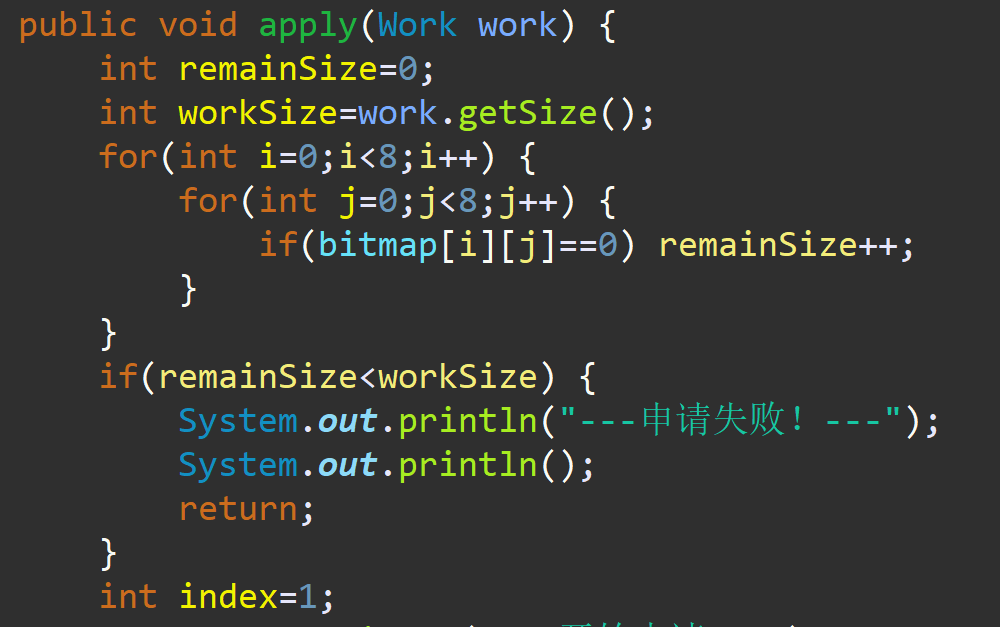
**3.1.2 上机实验所用平台**

Windows 10 Eclipse Jee 2019-09 Java 1.8

**3.2 数据结构及代码段分析**



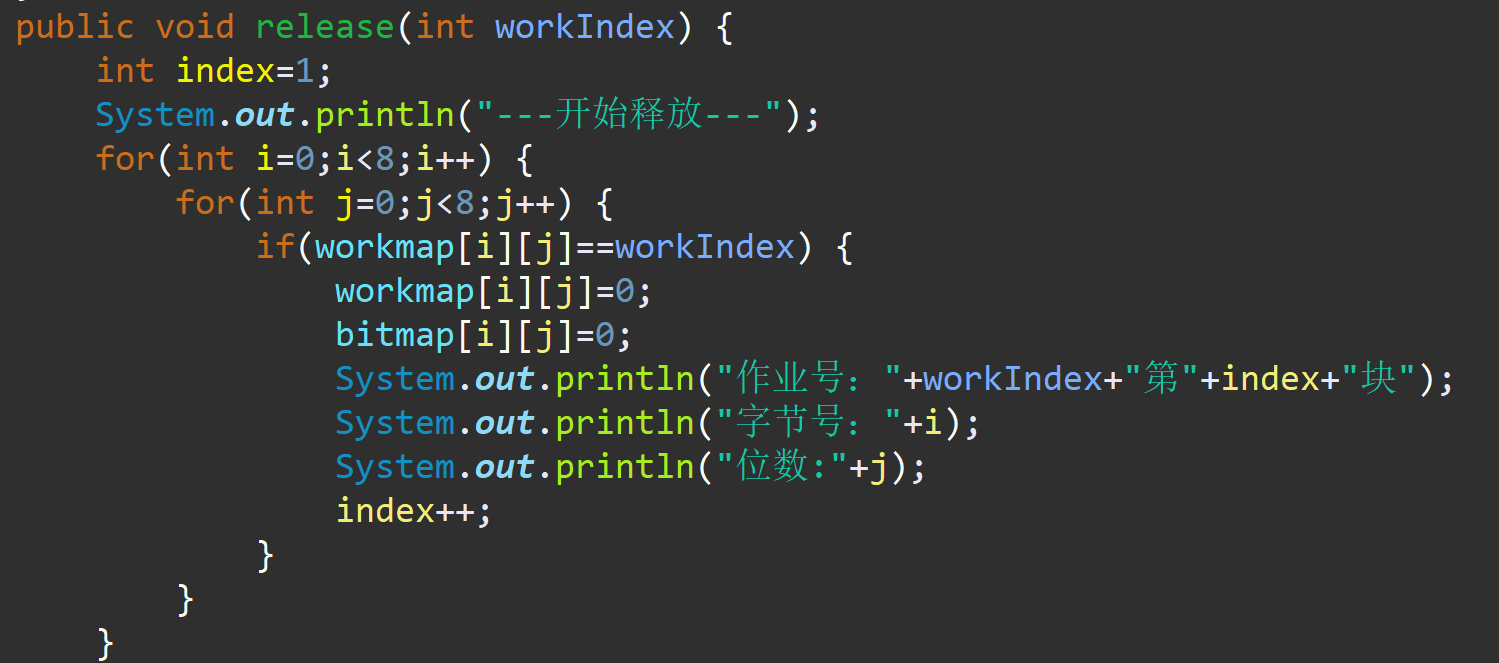
使用两个二维数组来模拟位示图和标记位示图；位示图的1或0来表示对应的磁盘空间块是否被占用，标记位示图的对应位置则保存占用磁盘空间块的作业的作业号，标记位示图中作业号的数量则表示该作业的大小

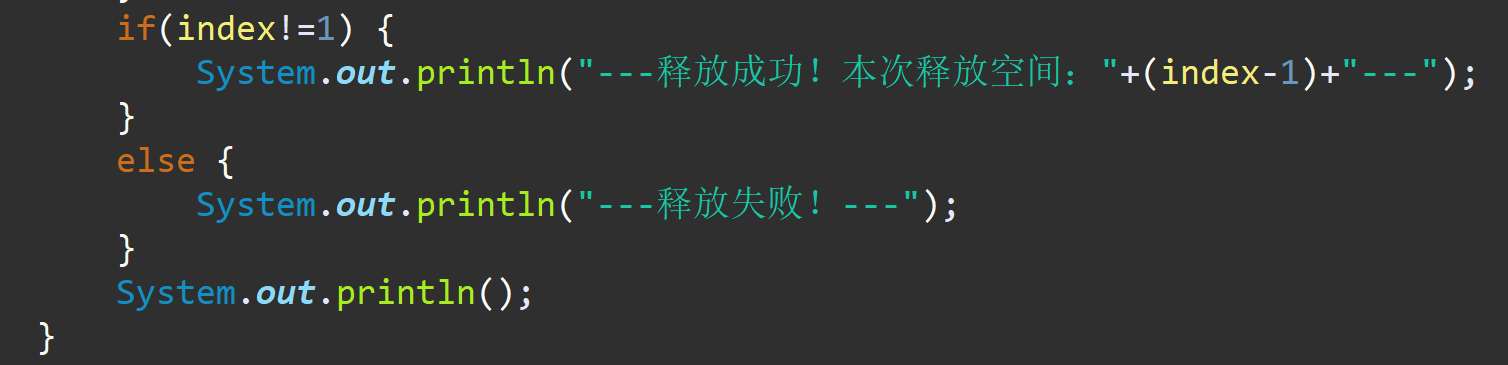


当申请磁盘空间时，若磁盘剩余空间不够，则直接输出申请失败，方法结束



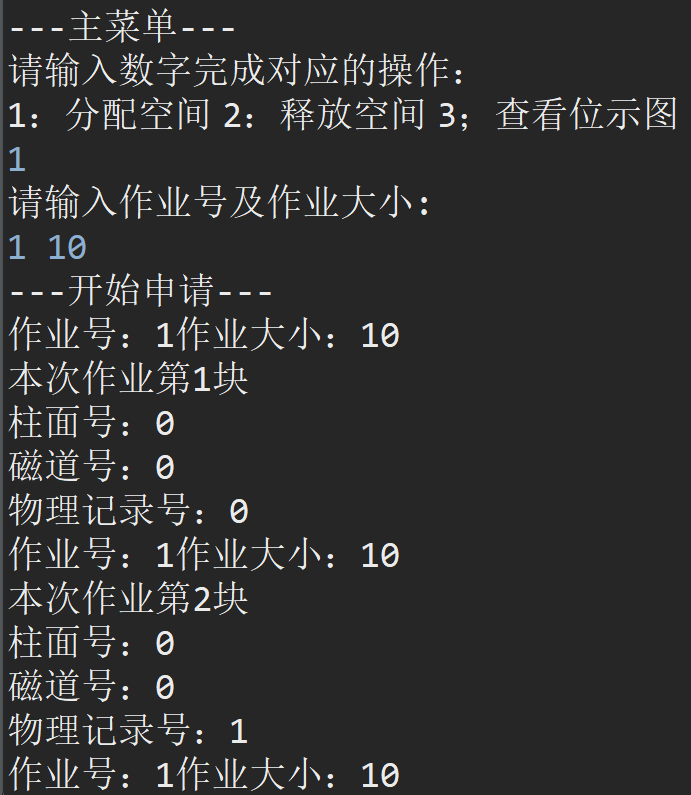
成功申请磁盘空间代码如上

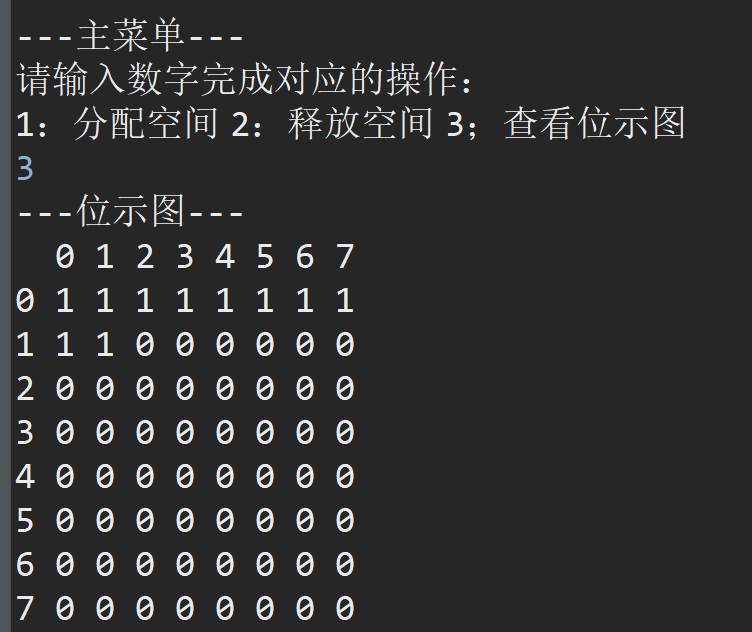




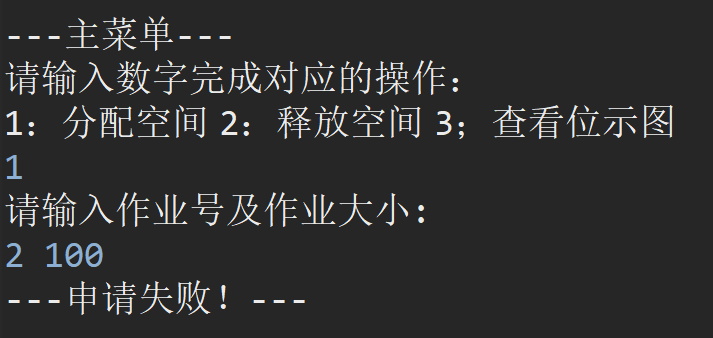
上面两张图为释放磁盘空间代码

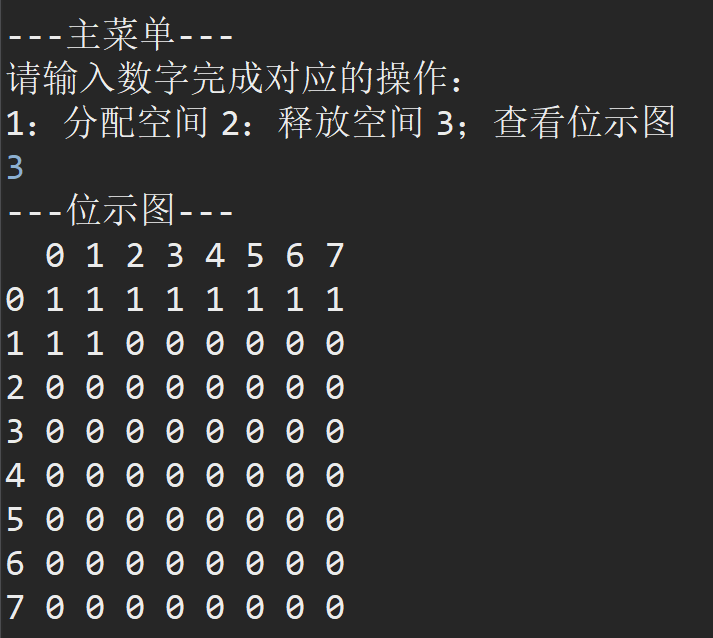
**3.3 调试过程**



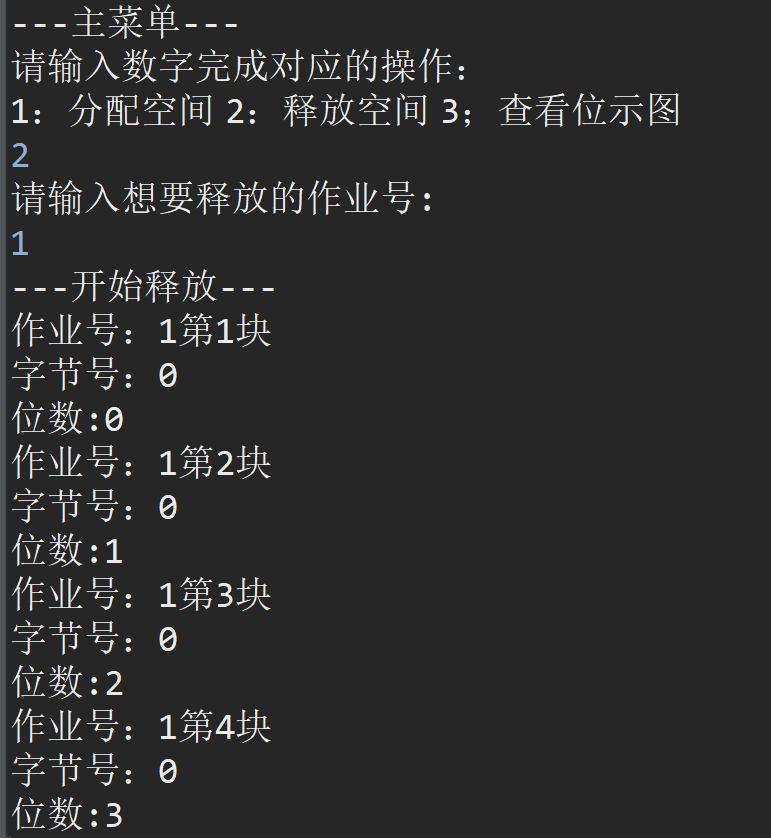


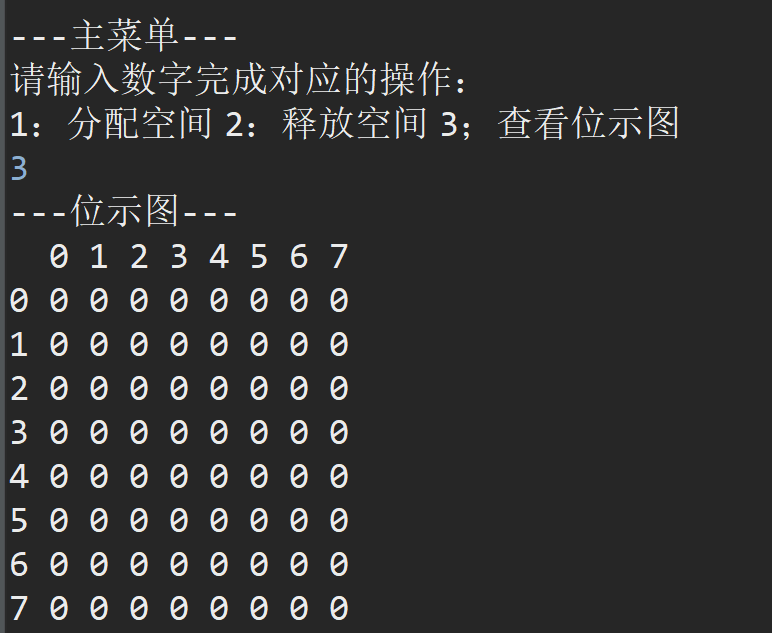
以上两张图为成功申请内存的情况





以上两张图为磁盘空间不足申请失败的情况





以上两张图为成功释放磁盘空间的情况

**3.4 实验总结**

本实验采用两个二维数组来模拟位示图和标记位示图，经过多组数据测试，实验结果均符合预期。本实验成功地用位示图管理了磁盘存储空间。

**4 进程创建**

**4.1 实习内容及上机实验所用平台**

**4.1.1 实验内容**

实习题目：

编制一段程序，使用系统调用fork( )创建两个子进程，这样在此程序运行时，在系统中就有一个父进程和两个子进程在活动。每一个进程在屏幕上显示一个字符，其中父进程显示字符A，子进程分别显示字符 B和字符C。试观察、记录并分析屏幕上进程调度的情况。

[提示]：

（1）可用fork()系统调用来创建一个新进程。

系统调用格式：pid=fork()

fork()返回值意义如下：

=0：若返回值为0，表示当前进程是子进程。

>0：若返回值大于0，表示当前进程是父进程，返回值为子进程的pid值。

-1：若返回值小于0，表示进程创建失败。

如果fork()调用成功，它向父进程返回子进程的pid，并向子进程返回0，即fork()被调用了一次，但返回了两次。此时OS在内存中建立一个新进程，所建的新进程是调用fork()父进程的副本，称为子进程。子进程继承了父进程的许多特性，并具有与父进程完全相同的用户级上下文。父进程与子进程并发执行。

（2）编译和执行的方法：

gcc 源文件名 -o 执行文件名

最后，在shell提示符下输入： ./执行文件名

就可执行该文件。

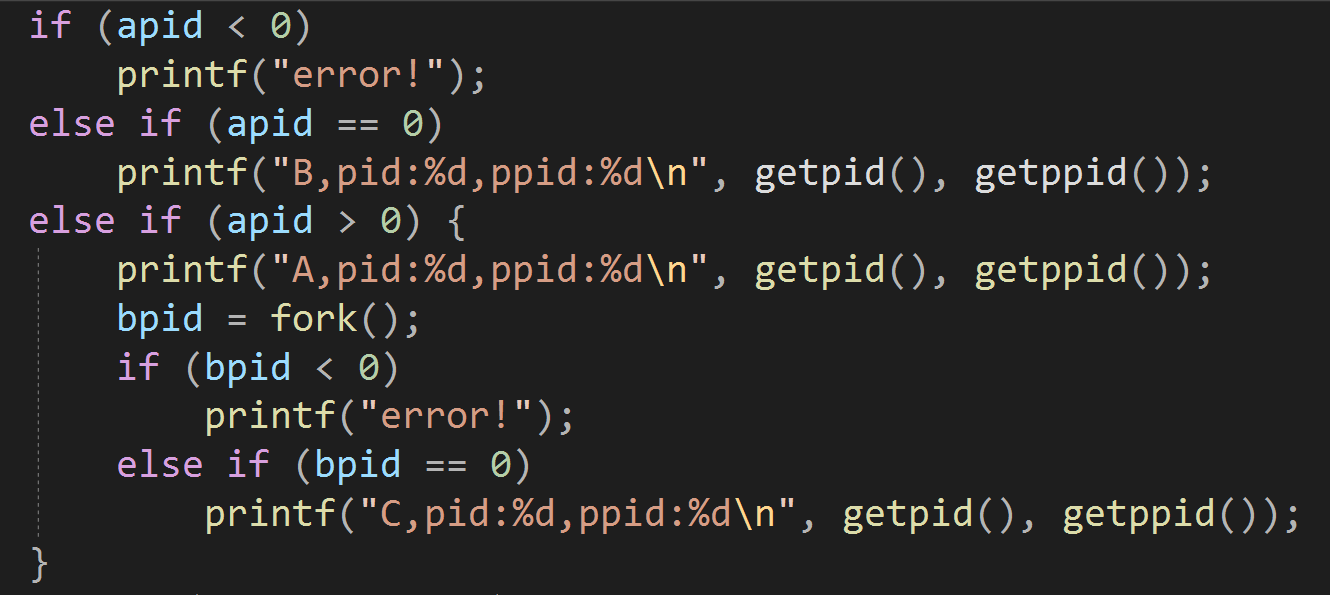
**4.1.2 上机实验所用平台**

Ubuntu 18.04 gcc c语言

**4.2 数据结构及代码段分析**

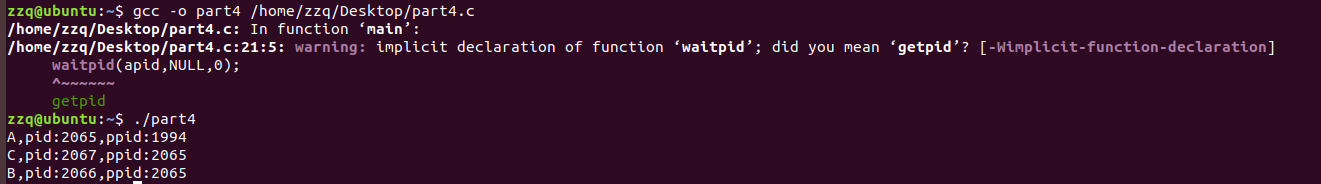


采用apid和bpid来保存两次进程创建的返回值



第一处创建进程时：若是父进程，则apid>0，会输出父进程（A）的pid和ppid；若是子进程，则apid=0，输出子进程（B）的pid和ppid；第二处创建进程时bpid同理。

**4.3 调试过程**



上图为在Ubuntu中编译及运行结果

**4.4 实验总结**

本实验在Ubuntu中运行，采用apid和bpid来保存fork（）函数的返回值，经过多组数据测试，实验结果均符合预期。本实验成功地模拟了多进程创建的过程。

**5 进程同步**

**5.1 实习内容及上机实验所用平台**

**5.1.1 实验内容**

模拟P、V操作实现同步机构，且用P、V操作解决生产者—消费者问题。

[提示]：

(1) P、V操作同步机构，由P操作原语和V操作原语组成，它们的定义如下：

P操作原语P (s)：将信号量s减去1，若结果小于0，则执行原语的进程被置成等待信号量s的状态。

V操作原语V (s)：将信号量s加1，若结果不大于0，则释放一个等待信号量s的进程。

这两条原语是如下的两个过程：

procedure p (var s: semaphore);

begin s: = s-1;

if s<0 then W (s)

end {p}

procedure v (var s: semaphore);

begin s: = s+1;

if s0 then R (s)

end {v}

其中W（s）表示将调用过程的进程置为等待信号量s的状态；R（s）表示释放一个等待信号量s的进程。

在系统初始化时应把semaphore定义为某个类型，为简单起见，在模拟实验中可把上述的semaphore直接改成integer。

(2) 生产者—消费者问题。

假定有一个生产者和一个消费者，生产者每次生产一件产品，并把生产的产品存入共享缓冲器以供消费者取走使用。消费者每次从缓冲器内取出一件产品去消费。禁止生产者将产品放入已满的缓冲器内，禁止消费者从空缓冲器内取产品。假定缓冲器内可同时存放10件产品，用P、V操作来实现生产者和消费者之间的同步，生产者和消费者两个进程的程序如下：

B: array [0..9] of products;

s1, s2; semaphore;

s1: =10, s2: =0;

IN, out: integer;

IN: =0; out: =0;

cobegin

procedure producer;

c: products;

begin

L1:

Produce (c);

P (s1);

B[IN]: =C;

IN: =(IN+1)mod 10;

V (s2);

goto L1

end;

procedure consumer;

x: products;

begin

L2: p (s2);

x: =B[out];

out: =(out+1) mod10;

v (s1);

consume (x);

goto L2

end;

coend.

其中的semaphore和products是预先定义的两个类型，在模拟实现中semaphore用integer代替，products可用integer或char等代替。

(3) 进程控制块PCB。

为了记录进程执行时的情况，以及进程让出处理器后的状态，断点等信息，每个进程都有一个进程控制块PCB。在模拟实验中，假设进程控制块的结构如图1。其中进程的状态有：运行态、就绪态、等待态和完成态。当进程处于等待态时，在进程控制块PCB中要说明进程等待原因（在模拟实验中进程等待原因是为等待信号量s1或s2）；当进程处于等待态或就绪态时，PCB中保留了断点信息，一旦进程再度占有处理器则从断点位置继续运行；当进程处于完成状态，表示进程执行结束。

|  |
| --- |
| 进程名 |
| 状态 |
| 等待原因 |
| 断点 |

图1 进程控制块结构

(4) 处理器的模拟。

计算机硬件提供了一组机器指令，处理器的主要职责是解释执行机器指令。为了模拟生产者和消费者进程的并发执行，我们必须模拟一组指令和处理职能。

模拟的一组指令见图2，其中每条指令的功能由一个过程来实现。用变量PC来模拟“指令计数器”，假设模拟的指令长度为1，每执行一条模拟指令后，PC加1，提取出下一条指令地址。使用模拟的指令，可把生产者和消费者进程的程序表示为图3的形式。

定义两个一维数组PA[0..4]和SA[0..4]，每一个PA[i]存放生产者程序中的一条模拟指令执行的入口地址；每个SA[i]存放消费者程序中的一条模拟指令执行的入口地址。于是模拟处理器执行一条指令的过程为：取出PC之值，按PA[PC]或SA[PC]得模拟指令执行的入口地址，将PC之值加1，转向由入口地址确定的相应的过程执行。

(5) 程序设计

本实验中的程序由三部分组成：初始化程序、处理器调度程序、模拟处理器指令执行程序。各部分程序的功能及相互间的关系由图4至图7给出。

|  |  |
| --- | --- |
| 模拟的指令 | 功 能 |
| p (s) | 执行P操作原语 |
| v (s) | 执行V操作原语 |
| Put | B[IN]: =product; IN: = (IN+1) mod 10 |
| GET | x:=B[out]; out: =(out+1) mod 10 |
| produce | 输入一个字符放入C中 |
| consume | 打印或显示x中的字符 |
| GOTO L | PC: =L |
| NOP | 空操作 |

图2 模拟的处理器指令

|  |  |  |
| --- | --- | --- |
| 序号 | 生产者程序 | 消费者程序 |
| 0 | produce | p (s2) |
| 1 | p (s1) | GET |
| 2 | PUT | v (s1) |
| 3 | v (s2) | consume |
| 4 | goto 0 | goto 0 |

图3 生产者和消费者程序

 ·初始化程序：模拟实验的程序从初始化程序入口启动，初始化工作包括对信号量s1、s2赋初值，对生产者、消费者进程的PCB初始化。初始化后转向处理调度程序，其流程如图4。

6

图4 初始化流程

·处理器调度程序：在计算机系统中，进程并发执行时，任一进程占用处理器执行完一条指令后就有可能被打断而让出处理器由其它进程运行。故在模拟系统中也类似处理，每当执行一条模拟的指令后，保护当前进程的现场，让它成为非运行态，由处理器调度程序按随机数再选择一个就绪进程占用处理器运行。处理器调度程序流程见图5。

5

图5 模拟处理器调度

·模拟处理器指令执行程序：按“指令计数器”PC之值执行指定的指令，且PC加1指向下一条指令。模拟处理器指令执行程序的流程图见图6和图7。7

 图6 模拟处理器指令执行

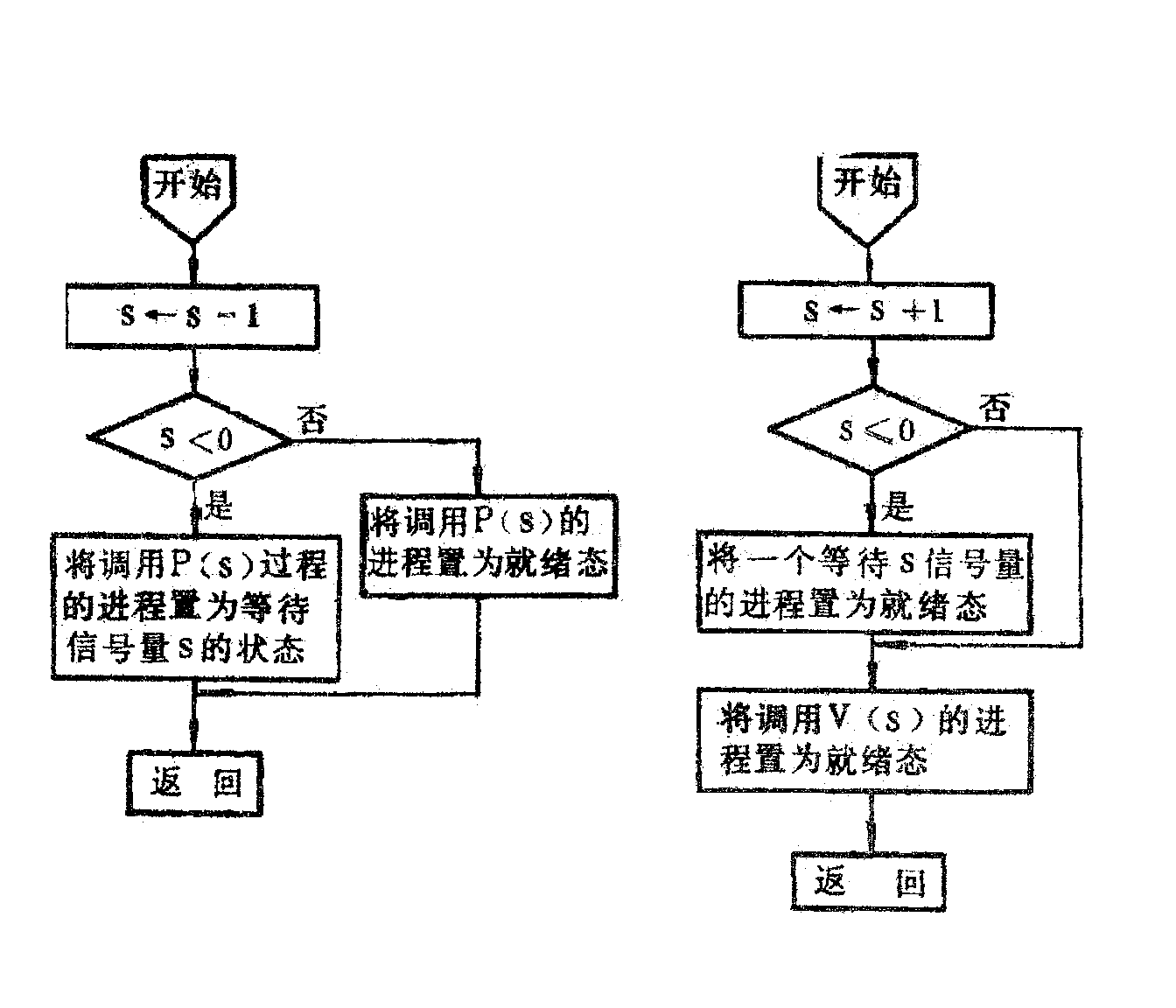
(a) 模拟P (s) (b) 模拟V (s) 

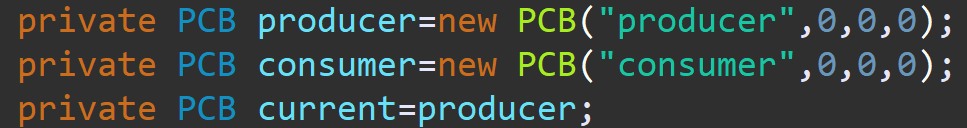
图7 模拟P、V操作的执行

 另外，为了使得模拟程序有一个结束条件，在图6中附加了“生产者运行结束”的条件判断，模拟时可以采用人工选择的方法实现。7给出了P（s）和V（s）模拟指令执行过程的流程。其它模拟指令的执行过程已在图2中指出。

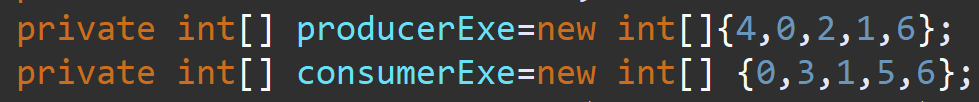
**5.1.2 上机实验所用平台**

Windows 10 Eclipse Jee 2019-09 Java 1.8

**5.2 数据结构及代码段分析**



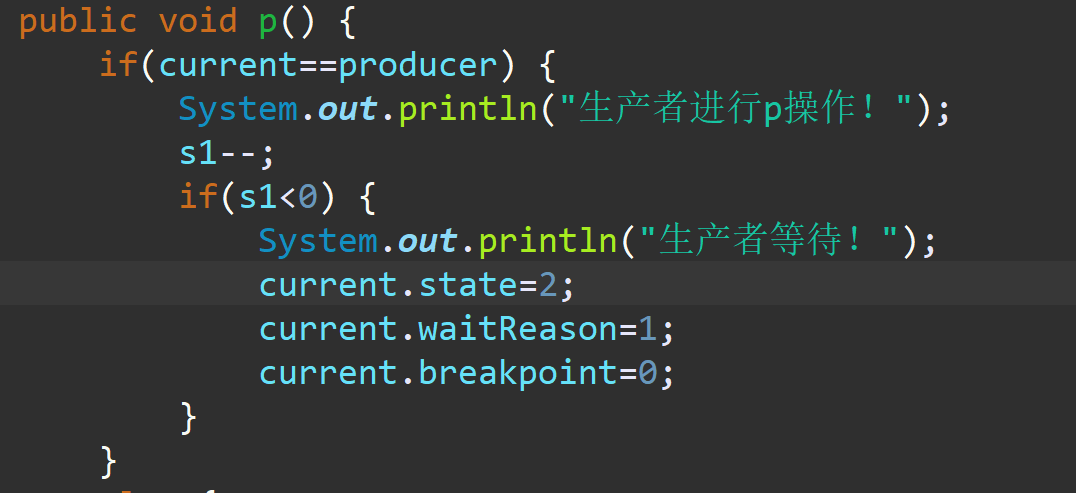
用不同的PCB对象来表示生产者消费者和当前运行对象



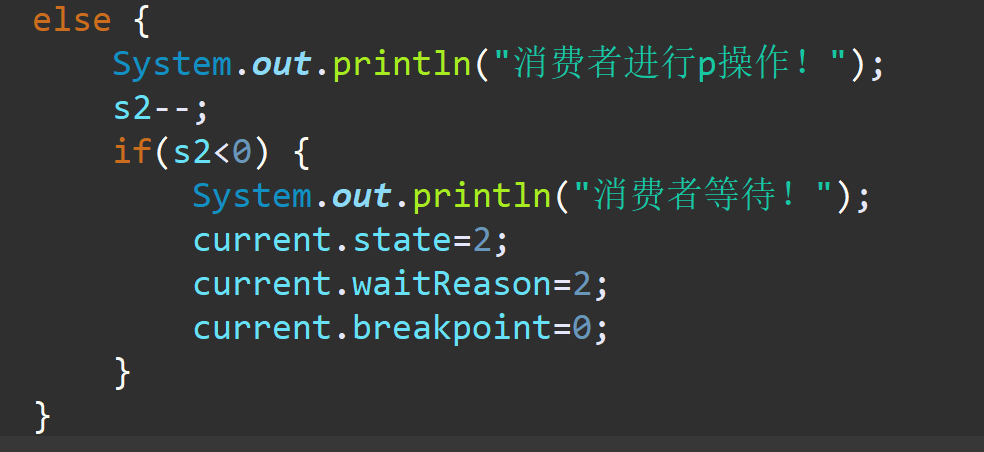
用两个数组来储存对生产者和消费者的各种操作



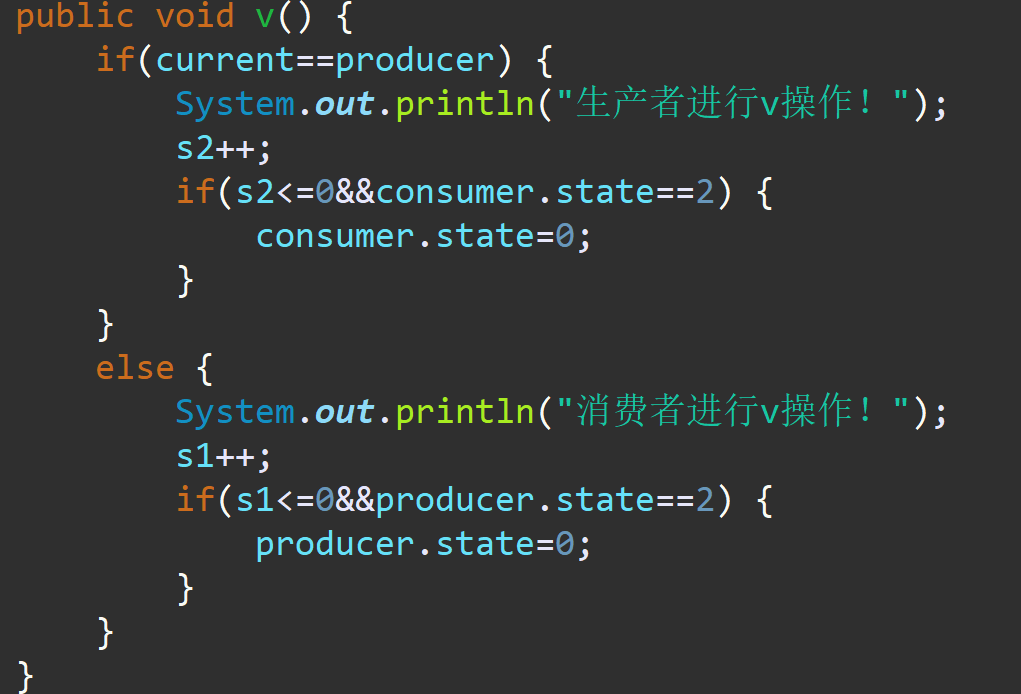
用buffer数组来模拟缓冲区



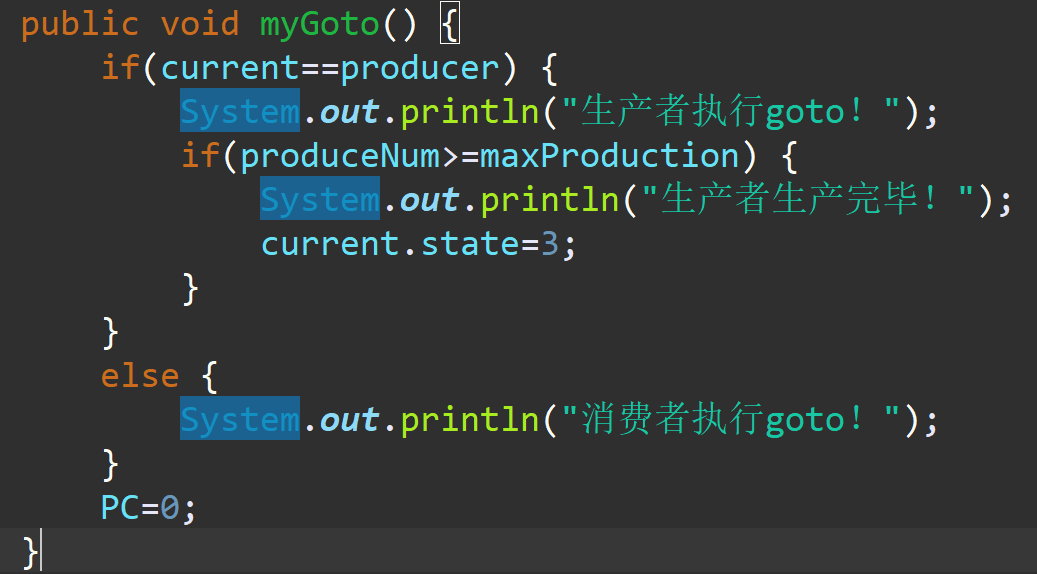
生产者p操作



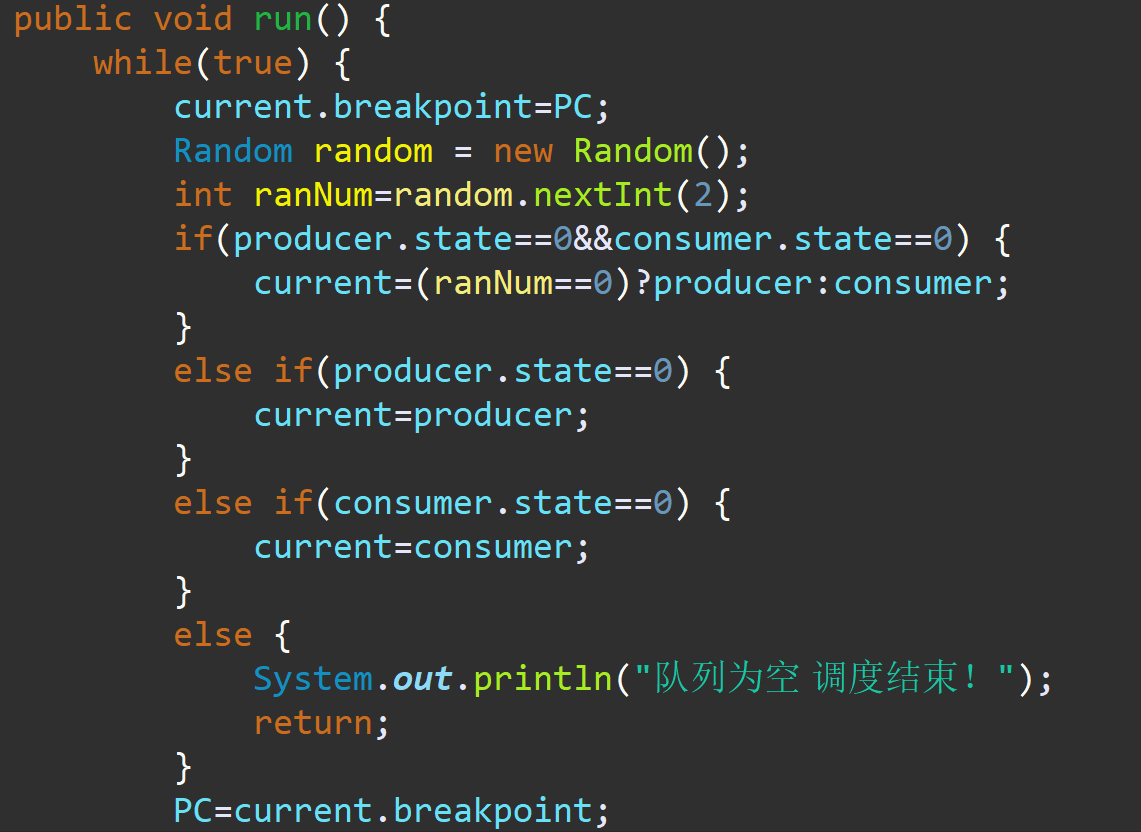
消费者p操作



生产者和消费者的v操作



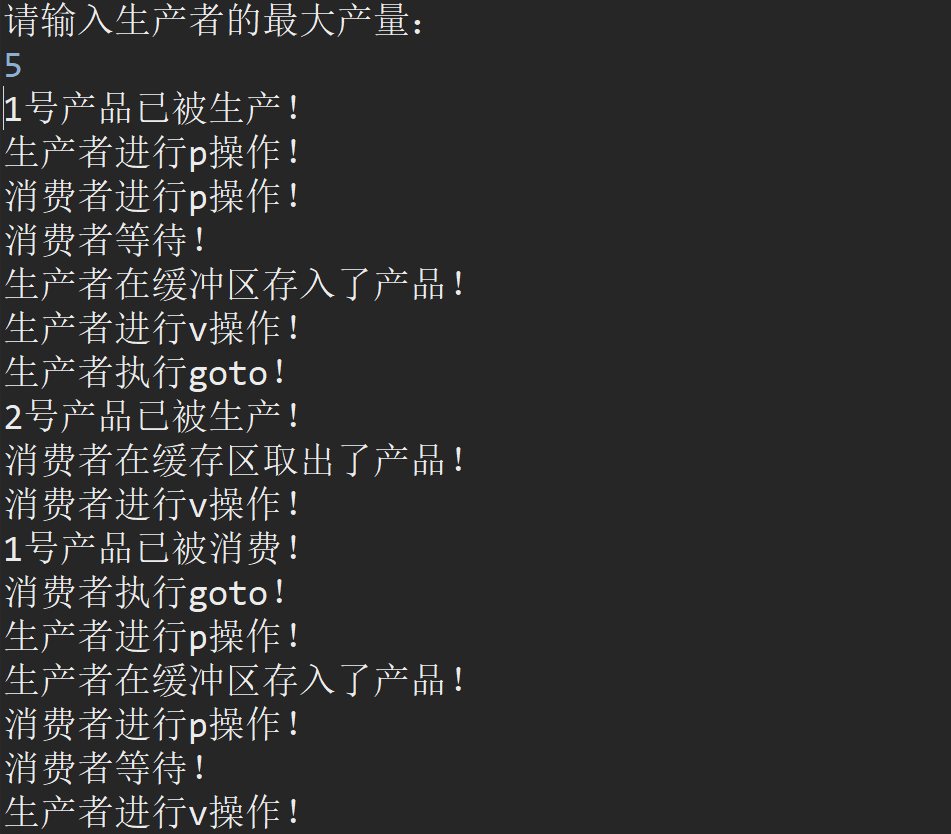
Goto函数：实现程序的跳转





上面两图为程序运行主体函数，生产者和消费者随机运行

**5.3 调试过程**



上图为生产者最大产量为5时部分运行结果截图

**5.4 实验总结**

本实验用不同的PCB对象来表示生产者消费者和当前运行对象，用两个数组来储存对生产者和消费者的各种操作，用buffer数组来模拟缓冲区。经过多次测试，实验结果均符合预期。本实验成功地模拟P、V操作实现同步机构，且用P、V操作解决生产者—消费者问题。

**6 附录**

Task 1：

Main.java

package task1;

public class Main {

public static void main(String[] args) {

OS os=new OS();

os.init();

os.run();

}

}

OS.java

package task1;

import java.util.ArrayList;

import java.util.Comparator;

public class OS {

private ArrayList<PCB> readyList=new ArrayList<PCB>();

private ArrayList<PCB> endList=new ArrayList<PCB>();

public void init() {

for(int i=0;i<5;i++) {

PCB p=new PCB();

readyList.add(p);

}

}

public void show() {

if(!readyList.isEmpty()) {

System.out.println("就绪进程如下：");

for(PCB p:readyList) {

System.out.println(p);

}

}

if(!endList.isEmpty()) {

System.out.println("结束进程如下：");

for(PCB p:endList) {

System.out.println(p);

}

}

}

public void run() {

int num=1;

readyList.sort(new Comparator<PCB>() {

@Override

public int compare(PCB o1, PCB o2) {

int result=o1.getPriority()<o2.getPriority()?1:-1;

return result;

}

});

while(!readyList.isEmpty()) {

System.out.println("第"+num+"轮运行：");

num++;

readyList.get(0).run();

if(!readyList.get(0).isStatus()) {

endList.add(readyList.get(0));

readyList.remove(0);

}

readyList.sort(new Comparator<PCB>() {

@Override

public int compare(PCB o1, PCB o2) {

int result=o1.getPriority()<o2.getPriority()?1:-1;

return result;

}

});

show();

}

System.out.println("所有进程完成！");

}

}

PCB.java

package task1;

import java.util.Scanner;

public class PCB {

private String name;//进程名

private int time;//要求运行时间

private int priority;//优先级

private boolean status;//状态：0为完成 1为就绪

public int getTime() {

return time;

}

public void setTime(int time) {

this.time = time;

}

public int getPriority() {

return priority;

}

public void setPriority(int priority) {

this.priority = priority;

}

public boolean isStatus() {

return status;

}

public void setStatus(boolean status) {

this.status = status;

}

public PCB() {

Scanner scanner = new Scanner(System.in);

System.out.println("请输入进程名：");

name=scanner.nextLine();

System.out.println("请输入进程要求时间：");

time=scanner.nextInt();

System.out.println("请输入优先级：");

priority=scanner.nextInt();

status=true;

System.out.println("创建成功！");

}

@Override

public String toString() {

return "PCB [name=" + name + ", time=" + time + ", priority=" + priority + ", status=" + status + "]";

}

public void run() {

if(status) {

time--;

priority--;

if(time<1) status=false;

System.out.println("进程"+name+"正在运行！");

}

}

}

Task 2:

Main.java

package task2;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

RAMList ramList=new RAMList();

Scanner scanner=new Scanner(System.in);

while(true) {

System.out.println("---主菜单---");

System.out.println("请输入数字完成对应指令");

System.out.println("1:分配作业");

System.out.println("2:释放作业");

System.out.println("3:查看内存状况");

switch(scanner.nextInt()) {

case 1:

System.out.println("请输入作业号和作业大小：");

if(!ramList.apply(new Work(scanner.nextInt(),scanner.nextDouble()))) {

System.out.println("分配失败！内存不足！");

}

break;

case 2:

System.out.println("请输入想释放的作业号：");

if(!ramList.release(scanner.nextInt())) {

System.out.println("释放失败！作业号错误！");

}

break;

case 3:

int ramIndex=1;

for(int i=0;i<ramList.RAMPartList.size();i++) {

System.out.println("---第"+(i+1)+"分区---");

if(ramList.RAMPartList.get(i).isState()) {

System.out.println("---第"+ramIndex+"空闲分区---");

}

System.out.println(ramList.RAMPartList.get(i));

}

break;

default:

break;

}

}

}

}

RAMPart.java

package task2;

public class RAMPart {

private double capacity;

private boolean state;

private double startAddress;

private int workIndex;

public double getCapacity() {

return capacity;

}

public void setCapacity(double capacity) {

this.capacity = capacity;

}

public boolean isState() {

return state;

}

public RAMPart(double capacity, boolean state, double startAddress, int workIndex) {

super();

this.capacity = capacity;

this.state = state;

this.startAddress = startAddress;

this.workIndex = workIndex;

}

public void setState(boolean state) {

this.state = state;

}

@Override

public String toString() {

return "RAMPart [capacity=" + capacity + ", state=" + state + ", startAddress=" + startAddress + ", workIndex="

+ workIndex + "]";

}

public double getStartAddress() {

return startAddress;

}

public void setStartAddress(double startAddress) {

this.startAddress = startAddress;

}

public int getWorkIndex() {

return workIndex;

}

public void setWorkIndex(int workIndex) {

this.workIndex = workIndex;

}

}

Work.java

package task2;

public class Work {

private int workIndex;

private double size;

public int getWorkIndex() {

return workIndex;

}

public void setWorkIndex(int workIndex) {

this.workIndex = workIndex;

}

public double getSize() {

return size;

}

public void setSize(double size) {

this.size = size;

}

public Work(int workIndex, double size) {

this.workIndex = workIndex;

this.size = size;

}

}

RAMList.java

package task2;

import java.util.LinkedList;

public class RAMList {

public LinkedList<RAMPart> RAMPartList=new LinkedList<RAMPart>();

public RAMList() {

RAMPartList.add(new RAMPart(128,true,0,-1));

}

public boolean apply(Work work) {

for(int i=0;i<RAMPartList.size();i++) {

if(RAMPartList.get(i).isState()) {

if(RAMPartList.get(i).getCapacity()>work.getSize()) {

RAMPartList.add(i, new RAMPart(work.getSize(), false,

RAMPartList.get(i).getStartAddress(), work.getWorkIndex()));

RAMPartList.get(i+1).setCapacity(RAMPartList.get(i+1).getCapacity()

-work.getSize());

RAMPartList.get(i+1).setStartAddress(RAMPartList.get(i).getStartAddress()

+work.getSize());

return true;

}

else if(RAMPartList.get(i).getCapacity()==work.getSize()) {

RAMPartList.get(i).setState(false);

RAMPartList.get(i).setWorkIndex(work.getWorkIndex());

return true;

}

}

}

return false;

}

public boolean release(int workIndex) {

for(int i=0;i<RAMPartList.size();i++) {

if(RAMPartList.get(i).getWorkIndex()==workIndex) {

RAMPartList.get(i).setState(true);

RAMPartList.get(i).setWorkIndex(-1);

merge();

return true;

}

}

return false;

}

public void merge() {

if(RAMPartList.size()>1) {

for(int i=0;i<RAMPartList.size()-1;i++) {

if(RAMPartList.get(i).isState()&&RAMPartList.get(i+1).isState()) {

RAMPartList.get(i).setCapacity(RAMPartList.get(i).getCapacity()

+RAMPartList.get(i+1).getCapacity());

RAMPartList.remove(i+1);

i--;

}

}

}

}

}

Task 3:

Main.java

package task3;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Bitmap bitmap = new Bitmap();

Scanner scanner = new Scanner(System.in);

while(true) {

System.out.println("---主菜单---");

System.out.println("请输入数字完成对应的操作：");

System.out.println("1：分配空间 2：释放空间 3；查看位示图");

switch(scanner.nextInt()) {

case 1:

System.out.println("请输入作业号及作业大小:");

bitmap.apply(new Work(scanner.nextInt(),scanner.nextInt()));

break;

case 2:

System.out.println("请输入想要释放的作业号:");

bitmap.release(scanner.nextInt());

break;

case 3:

bitmap.displayBitmap();

break;

default:

System.out.println("输入错误！");

break;

}

}

}

}

Work.java

package task3;

public class Work {

private int index;

private int size;

public int getIndex() {

return index;

}

public void setIndex(int index) {

this.index = index;

}

public int getSize() {

return size;

}

public Work(int index, int size) {

this.index = index;

this.size = size;

}

public void setSize(int size) {

this.size = size;

}

}

Bitmap.java

package task3;

public class Bitmap {

private int[][] bitmap=new int[8][8];

private int[][] workmap=new int[8][8];

public Bitmap() {

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

bitmap[i][j]=0;

workmap[i][j]=0;

}

}

}

public void apply(Work work) {

int remainSize=0;

int workSize=work.getSize();

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

if(bitmap[i][j]==0) remainSize++;

}

}

if(remainSize<workSize) {

System.out.println("---申请失败！---");

System.out.println();

return;

}

int index=1;

System.out.println("---开始申请---");

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

if(workSize>0&&bitmap[i][j]==0) {

bitmap[i][j]=1;

workmap[i][j]=work.getIndex();

System.out.println("作业号："+work.getIndex()+"作业大小："+work.getSize());

System.out.println("本次作业第"+index+"块");

System.out.println("柱面号："+i);

System.out.println("磁道号："+(j/4));

System.out.println("物理记录号："+(j%4));

index++;

workSize--;

}

}

}

System.out.println("---申请成功！本次申请空间："+work.getSize()+"---");

System.out.println();

}

public void release(int workIndex) {

int index=1;

System.out.println("---开始释放---");

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

if(workmap[i][j]==workIndex) {

workmap[i][j]=0;

bitmap[i][j]=0;

System.out.println("作业号："+workIndex+"第"+index+"块");

System.out.println("字节号："+i);

System.out.println("位数:"+j);

index++;

}

}

}

if(index!=1) {

System.out.println("---释放成功！本次释放空间："+(index-1)+"---");

}

else {

System.out.println("---释放失败！---");

}

System.out.println();

}

public void displayBitmap() {

System.out.println("---位示图---");

System.out.println(" 0 1 2 3 4 5 6 7");

for(int i=0;i<8;i++) {

System.out.print(i+" ");

for(int j=0;j<8;j++) {

System.out.print(bitmap[i][j]+" ");

}

System.out.println();

}

System.out.println();

}

}

Task 4:

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main() {

pid\_t apid, bpid;

apid = fork();

if (apid < 0)

printf("error!");

else if (apid == 0)

printf("B,pid:%d,ppid:%d\n", getpid(), getppid());

else if (apid > 0) {

printf("A,pid:%d,ppid:%d\n", getpid(), getppid());

bpid = fork();

if (bpid < 0)

printf("error!");

else if (bpid == 0)

printf("C,pid:%d,ppid:%d\n", getpid(), getppid());

}

waitpid(apid, NULL, 0);

waitpid(bpid, NULL, 0);

return 0;

}

Task 5:

Main.java

package task5;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("请输入生产者的最大产量：");

OS os=new OS(scanner.nextInt());

os.run();

}

}

OS.java

package task5;

import java.util.Random;

public class OS {

private int produceNum=0;

private int consumeNum=0;

private int s1=10;

private int s2=0;

private int in=0;

private int out=0;

private int[] buffer=new int[10];

private int PC=0;

private int maxProduction=20;

private int[] producerExe=new int[]{4,0,2,1,6};

private int[] consumerExe=new int[] {0,3,1,5,6};

private PCB producer=new PCB("producer",0,0,0);

private PCB consumer=new PCB("consumer",0,0,0);

private PCB current=producer;

public OS(int maxProduction) {

this.maxProduction=maxProduction;

}

public void produce() {

produceNum++;

System.out.println(produceNum+"号产品已被生产！");

}

public void consume() {

System.out.println(consumeNum+"号产品已被消费！");

}

public void p() {

if(current==producer) {

System.out.println("生产者进行p操作！");

s1--;

if(s1<0) {

System.out.println("生产者等待！");

current.state=2;

current.waitReason=1;

current.breakpoint=0;

}

}

else {

System.out.println("消费者进行p操作！");

s2--;

if(s2<0) {

System.out.println("消费者等待！");

current.state=2;

current.waitReason=2;

current.breakpoint=0;

}

}

}

public void v() {

if(current==producer) {

System.out.println("生产者进行v操作！");

s2++;

if(s2<=0&&consumer.state==2) {

consumer.state=0;

}

}

else {

System.out.println("消费者进行v操作！");

s1++;

if(s1<=0&&producer.state==2) {

producer.state=0;

}

}

}

public void put() {

buffer[in]=produceNum;

System.out.println("生产者在缓冲区存入了产品！");

in=(in+1)%10;

}

public void get() {

consumeNum=buffer[out];

System.out.println("消费者在缓存区取出了产品！");

out=(out+1)%10;

}

public void myGoto() {

if(current==producer) {

System.out.println("生产者执行goto！");

if(produceNum>=maxProduction) {

System.out.println("生产者生产完毕！");

current.state=3;

}

}

else {

System.out.println("消费者执行goto！");

}

PC=0;

}

public void run() {

while(true) {

current.breakpoint=PC;

Random random = new Random();

int ranNum=random.nextInt(2);

if(producer.state==0&&consumer.state==0) {

current=(ranNum==0)?producer:consumer;

}

else if(producer.state==0) {

current=producer;

}

else if(consumer.state==0) {

current=consumer;

}

else {

System.out.println("队列为空 调度结束！");

return;

}

PC=current.breakpoint;

switch((current==producer)?producerExe[PC++]:consumerExe[PC++]) {

case 0:

p();break;

case 1:

v();break;

case 2:

put();break;

case 3:

get();break;

case 4:

produce();break;

case 5:

consume();break;

case 6:

myGoto();break;

default:

break;

}

}

}

}

PCB.java

package task5;

public class PCB {

public String name;

public int state;//进程状态：0就绪态 1运行态 2等待态 3完成态

public int waitReason;//等待原因：1为等待s1 2为等待s2

public int breakpoint;//断点

public PCB(String name, int state, int waitReason, int breakpoint) {

super();

this.name = name;

this.state = state;

this.waitReason = waitReason;

this.breakpoint = breakpoint;

}

}