

中國地質大學



题 目:	操作系统原理上机报告
姓 名:	常文瀚
院 系:	计算机学院
班 级:	191181
学 号:	20181001095

2020 年 6 月 9 日

1. 实验一 多级队列调度算法

1.1 实验题目：设 RQ 分为 RQ1 和 RQ2，RQ1 采用轮转法，时间 $q=7$ 。

$RQ1 > RQ2$ ，RQ2 采用短进程优先调度算法。

测试数据如下：RQ1: P1-P5, RQ2: P6-P10

进程	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
运行时间	16	11	14	13	15	21	18	10	7	14
已等待时间	6	5	4	3	2	1	2	3	4	5

1.2 程序功能及设计思路

根据先来先服务的原则，将需要执行的所有进程按照到达时间的大小排成一个升序的序列，每次都给一个进程同样大小的时间片，在这个时间片内如果进程执行结束了，那么把进程从进程队列中删去，如果进程没有结束，那么把该进程停止然后改为等待状态，放到进程队列的尾部，直到所有的进程都已执行完毕。

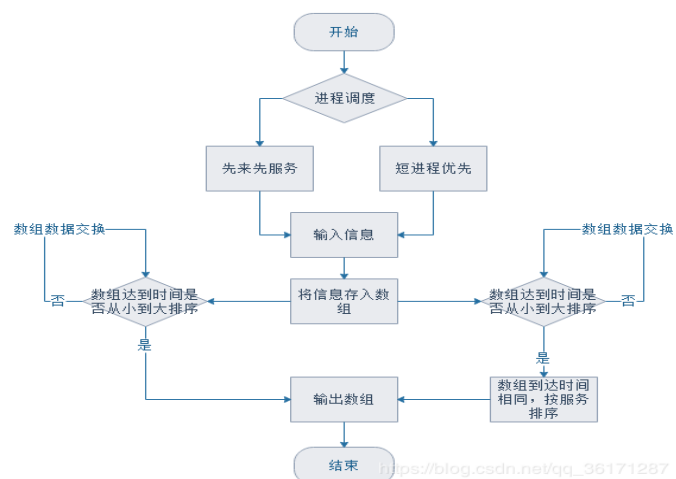
短作业(进程)优先调度算法 SJ(P)F，是指对短作业或短进程优先调度的算法。它们可以分别用于作业调度和进程调度。短作业优先(SJF)的调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。而短进程优先(SPF)调度算法则是从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时再重新调度。

1.3 数据结构及算法设计

(1) 编程语言：Java

(2) 数据结构：单链表

(3) 程序框图：



(4) 源代码:

```
class Node{

    Node next;

    String name;

    int need;

    int turn;

    public Node(String name,int need,int turn){

        this.name = name;

        this.need = need;

        this.turn = turn;

        this.next = null;

    }

}

public class question1 {

    static int clock = 0,piecetime = 7,max=1000;

    public static void test(int clock,int piecetime){

        //设置数组和新建一些节点

        String[] name1 = {"p1","p2","p3","p4","p5"};

        String[] name2 = {"p6","p7","p8","p9","p10"};

        int[] need1 = {16,11,14,13,15};

        int[] wait1 = {6,5,4,3,2};

        int[] need2 = {21,18,10,7,14};

        int[] wait2 = {1,2,3,4,5};

        Node RQ1 = new Node(name1[0], need1[0], wait1[0]);

        Node RQ2 = new Node(name2[0], need2[0], wait2[0]);

        Node p,q,s=null,Finish = null;

        //轮转调度法 RoundRobin
```

```

p = RQ1;
for(int i = 1; i<5; i++){
    q = new Node(name1[i], need1[i], wait1[i]);
    //System.out.print(q.name+"-"+q.need+"-"+q.turn+"; ");
    p.next = q;
    p=p.next;
}

p = RQ1;
while(p!=null) {
    boolean flag = false;
    if(p.need!=0) {
        q=p;
        while(q.next!=null){
            q=q.next;    //q 指针指向最后一个结点
        }
        if(p.need>piecetime){    //如果该节点表示的进程还需的时间大于一个时间片
            clock += piecetime;
            p.need -= piecetime;
            //把刚运行了一个时间片的结点放到队列尾部
            q.next = p;
            p=p.next;
            q.next.next = null;
        }
        else{
            clock += p.need;
            p.turn += clock;
            p.need = 0;
            if(s==null){
                s=p;
            }
        }
    }
    p=p.next;
}

```

```
        Finish = s;

        flag = true;    //Finish 指针指向最后运行后的链表的第一
个节点
```

```
    }

    else{

        s.next = p;

    }

    if(flag==false){

        s=s.next;

    }

}

else{

    p=p.next;

}

}

RQ1 = Finish;

//短进程优先调度, ShortestJobFirst

p=RQ2;

for(int i = 1; i<5; i++){

    q = new Node(name2[i], need2[i], wait2[i]);

    p.next = q;

    p=p.next;

}

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

    q = RQ2;

    p = RQ2;
```

```

        //p.need = max;

        //p.next = null;
        while(q!=null){

            if(q.need!=0 && q.need<p.need)

                p=q;

            q=q.next;

        }

        clock += p.need;

        p.turn += clock;

        p.need = max;

    }

    //输出结果

    System.out.println("轮转调度法：");

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

    {

        System.out.println(RQ1.name+"周转时间为："+RQ1.turn);

        RQ1= RQ1.next;

    }

    System.out.println("短进程优先调度：");

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

        System.out.println(RQ2.name+"周转时间为："+RQ2.turn);

        RQ2 = RQ2.next;

    }

}

public static void main(String[] args){

    test(clock,piectime);

    //System.out.println("短进程优先调度法");

```

```

        //shorestJobFirst(clock, piecetime);
    }
}

```

1.4 程序运行情况

```

//shorestJobFirst(clock, piecetime);
    }
}

```

```

public static void main(String[] args) {
    test(clock, piecetime);
}

```

```

轮转调度法:
p2周转时间为: 51
p3周转时间为: 57
p4周转时间为: 62
p1周转时间为: 74
p5周转时间为: 71
短进程优先调度:
p6周转时间为: 140
p7周转时间为: 120
p8周转时间为: 89
p9周转时间为: 80
p10周转时间为: 105

```

1.5 编程中遇到的困难及解决方法、实习心得

在实验一的实验过程中，为了深入了解轮转调度法与短进程优先调度法，并且同时对本学期学到的 Java 课程复习，我选择了 Java 与单链表的数据结构来实现两种算法。实验一开始因为急于完成实验，导致我对实验算法的理解并不深入，有一些部分的理解比较模糊，例如短进程优先调度算法的排序阶段如何进行、轮转调度法忘记算上等待时间。在反复查阅网上资料，阅读理解一些博客中的 C++代码后，成功实现了两个算法。

从这个实验中，我认为以后的实践课程中应该做好记录，记录自己在理解算法是的错误，记录自己在实现算法时的一些误区，以及应对的方法，可以把他们打包保存在 github 上，同时要注意保证数据结构的正确性与便捷。

2. 实验二 银行家算法

2.1 实习题目：实现银行家算法

2.2 程序功能及设计思路：

银行家算法是一个避免死锁的著名算法，是由艾兹格·迪杰斯特拉在 1965 年为 T. H. E 系统设计的一种避免死锁产生的算法。它以银行借贷系统的分配策略为基础，判断并保证系统的安全运行。在银行中，客户申请贷款的数量是有限的，每个客户在第一次申请贷款时要声明完成该项目所需的最大资金量，在满足所有贷款要求时，客户应及时归还。银行家在客户申请的贷款数量不超过自己拥有的最大值时，都应尽量满足客户的需要。在这样的描述中，银行家就好比操作系统，资金就是资源，客户就相当于要申请资源的进程。

2.3 数据结构及算法设计

2.3.1 数据结构

(1) 编程语言：Java

(2) 可利用资源向量 ava

是个含有 r 个元素的[数组](#)，其中的每一个元素代表一类可利用的资源数目。如果 $Available[j]=K$ ，则表示系统中现有 R_j 类资源 K 个。

(3) 需求矩阵 req

(4) 分配[矩阵](#) $allo$

这也是一个 $p \times r$ 的[矩阵](#)，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果 $Allocation[i, j]=K$ ，则表示进程 i 当前已分得 R_j 类资源的 数目为 K 。

(5) 需求[矩阵](#) $need$ 。

这也是一个 $p \times r$ 的[矩阵](#)，用以表示每一个进程尚需的各类资源数。如果 $Need[i, j]=K$ ，则表示进程 i 还需要 R_j 类资源 K 个，方能完成其任务。

2.3.2 算法设计

银行家算法的基本思想是分配资源之前，判断系统是否是安全的；若是，才分配。它是最具有代表性的避免[死锁](#)的算法。

设进程 $cusneed$ 提出请求 $REQUEST[i]$ ，则银行家算法按如下规则进行判断。

(1) 如果 $REQUEST[cusneed][i] \leq NEED[cusneed][i]$ ，则转(2)；否则，出错。

(2) 如果 $REQUEST[cusneed][i] \leq AVAILABLE[i]$ ，则转(3)；否则，等待。

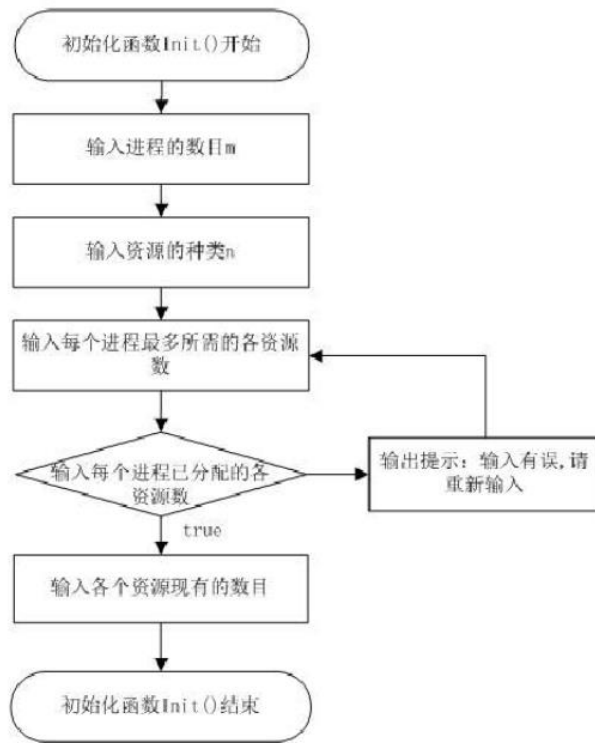
(3) 系统试探分配资源，修改相关数据：

$AVAILABLE[i] -= REQUEST[cusneed][i]$;

ALLOCATION[cusneed][i]+=REQUEST[cusneed][i];

NEED[cusneed][i]-=REQUEST[cusneed][i];

(4) 系统执行安全性检查，如安全，则分配成立；否则试探性分配作废，系统恢复原状，进程等待。



银行家算法流程图：

2.3.3 源代码：

```
import java.util.Scanner;

public class question2 {

    static int p = 5;
    static int r = 3;

    public static boolean com(int[] m, int[] n) {
        for(int i=0;i<m.length;i++) {
            if(m[i]<n[i])
                return false;
        }
    }
}
```

```

        return true;
    }

    public static boolean IsSafe(int allo[][], int need[][], int ava[]) {
        boolean[] finish = new boolean[p];
        for(int i=0;i<p;i++)
            finish[i] = false;
        for(int i=0;i<p;i++) {
            if(finish[i])
                finish[i]=false;
            else {
                int[] temp = new int[r];
                for(int j = 0;j<r;j++)
                    temp[j] = need[i][j];
                if(com(ava, temp)) {
                    finish[i] = true;
                    int a = i+1;
                    System.out.println("第"+ a +"个进程安全");
                    for(int k = 0;k<3;k++)
                        ava[k]+=allo[i][k];
                }
            }
        }
        for(int i=0;i<p;i++) {
            if(!finish[i]) {
                int a = i+1;
                System.out.println("第"+ a +"个进程不安全");
                return false;
            }
        }
    }
}

```

```

        return true;
    }

    public static void Apply(int allo[][], int need[][], int ava[], int req[],
int n){
        n=n-1;
        int[] temp = new int[r];
        for(int i=0;i<r;i++)
            temp[i] = need[n][i];
        if(!com(temp, req)) {
            System.out.println("请求违法");
            return;
        }
        else if(!com(ava, req)) {
            System.out.println("发生阻塞");
            return;
        }
        else {
            for(int j=0;j<r;j++) {
                allo[n][j] = allo[n][j] + req[j];
                need[n][j] = need[n][j] - req[j];
                ava[j] = ava[j] - req[j];
            }
            if (IsSafe(allo, need, ava)) {
                int a = n+1;
                System.out.println("同意"+a+"个进程申请资源");
            }
            else {
                int a=n+1;
                System.out.println("不同意"+a+"个进程申请资源");
            }
        }
    }
}

```

```

        System.out.println("恢复之前的状态");
        for (int j = 0; j < r; j++)
        {
            allo[n][j] = allo[n][j] - req[j];
            need[n][j] = need[n][j] + req[j];
            ava[j] = ava[j] + req[j];
        }
    }
}

```

```

public static void main(String[] args) {
    int[][] allo = new int[p][r];
    int[][] need = new int[p][r];
    int[] ava = new int[r];
    int[] req= new int[r]; int n;
    Scanner sc = new Scanner(System.in);

    System.out.println(" 输入每个进程已分配的资源数据");
    for(int i=0;i<p;i++)
        for(int j=0;j<r;j++)
            allo[i][j] = sc.nextInt();
    System.out.println(" 输入每个进程还需要的资源数据");
    for (int i = 0; i < p; i++)
        for (int j = 0; j < r; j++)
            need[i][j] = sc.nextInt();
    System.out.println("输入可用的资源数据");
    for (int j = 0; j < r; j++)
        ava[j] = sc.nextInt();
}

```

```

        if (IsSafe(allo, need, ava))

            System.out.println("初始状态安全");

        else

            System.out.println("初始状态不安全");

        while (true) {

            System.out.println("输入申请的资源");

            for (int i = 0; i < r; i++)

                req[i] = sc.nextInt();

            System.out.println("第 n 个进程申请资源——n 的值");

            n=sc.nextInt();

            Apply(allo, need, ava, req, n);

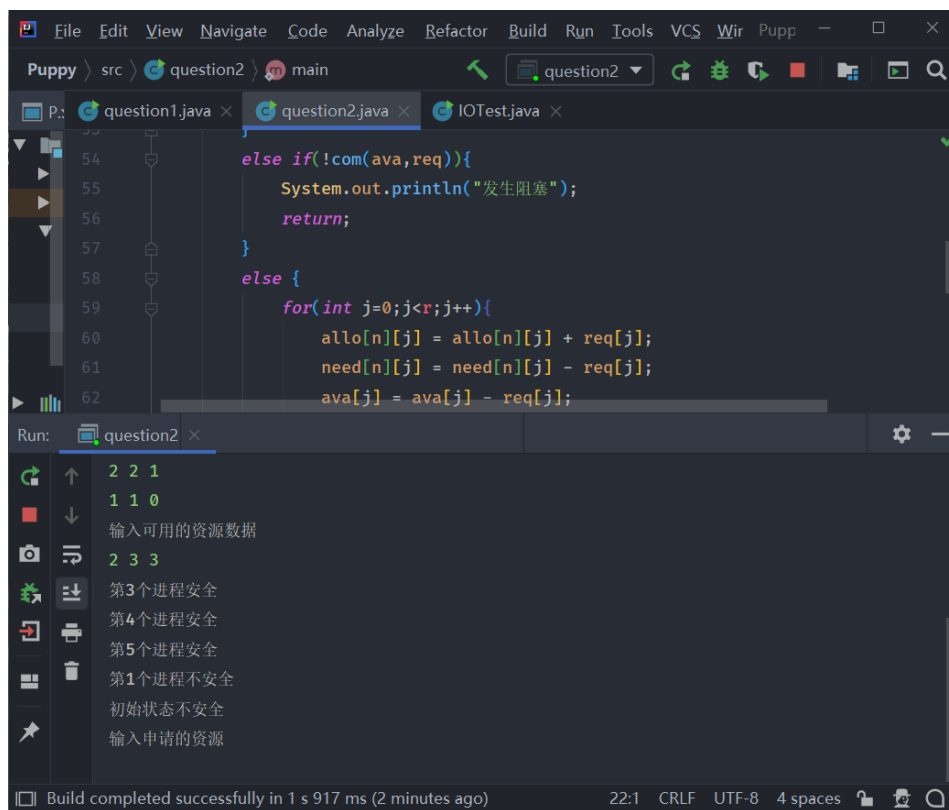
        }

    }

}

```

2.4 程序运行情况



The screenshot shows an IDE window with the following components:

- Editor:** Displays the Java code for `question2.java`. The visible code includes:


```

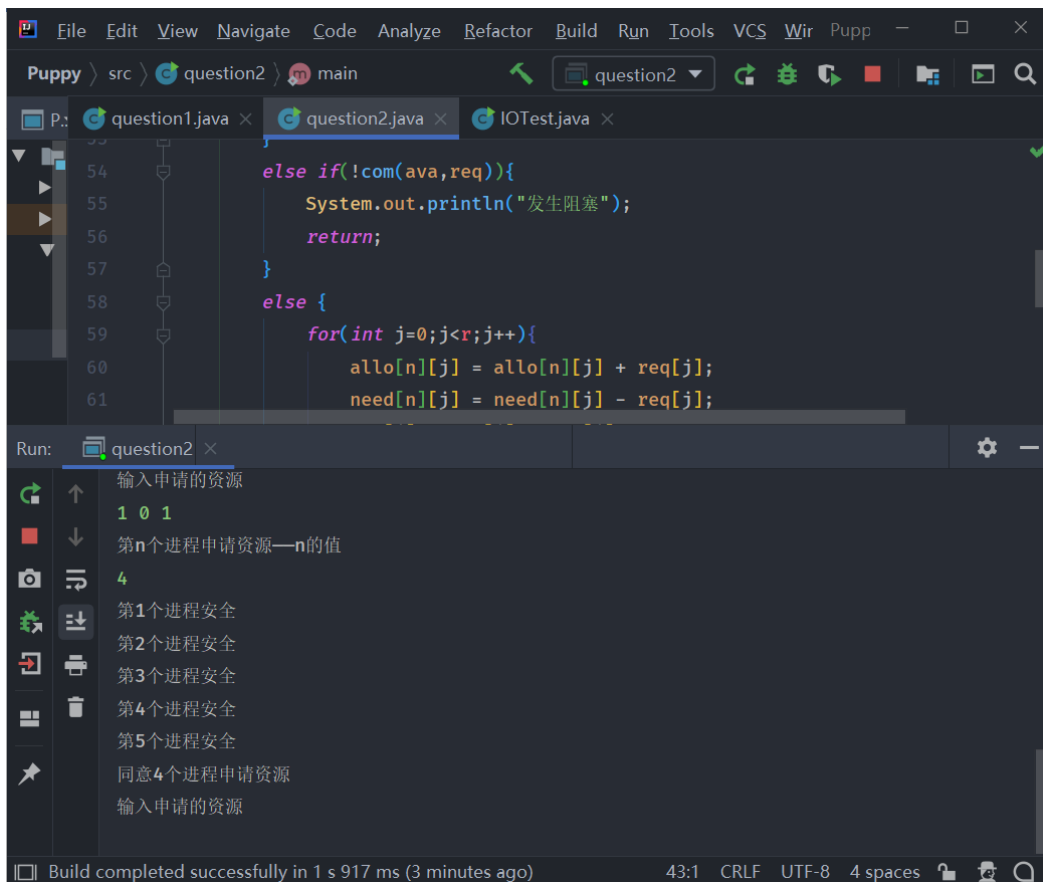
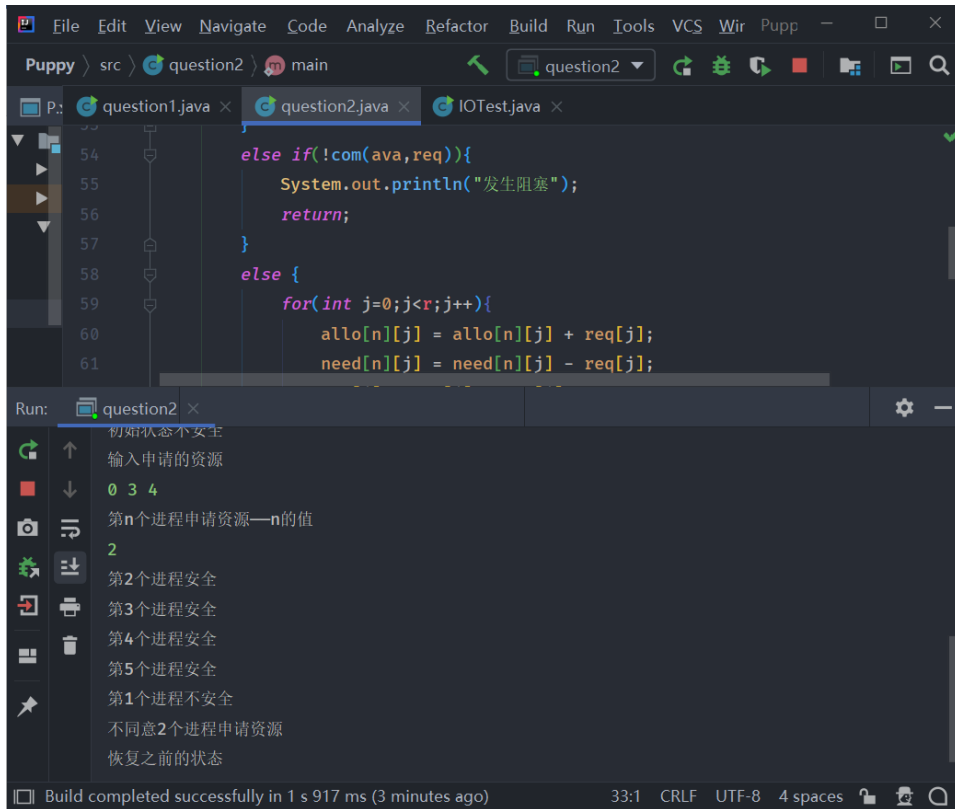
else if(!com(ava, req)){
    System.out.println("发生阻塞");
    return;
}
else {
    for(int j=0; j<r; j++){
        allo[n][j] = allo[n][j] + req[j];
        need[n][j] = need[n][j] - req[j];
        ava[j] = ava[j] - req[j];
    }
}

```
- Run Console:** Shows the output of the program execution:


```

2 2 1
1 1 0
输入可用的资源数据
2 3 3
第3个进程安全
第4个进程安全
第5个进程安全
第1个进程不安全
初始状态不安全
输入申请的资源

```
- Status Bar:** Indicates "Build completed successfully in 1 s 917 ms (2 minutes ago)".



File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Wir Pupp - □ ×

Puppy > src > question2 > main

question1.java × question2.java × IOTest.java ×

```
54 else if(!com(ava,req)){
55     System.out.println("发生阻塞");
56     return;
57 }
58 else {
59     for(int j=0;j<r;j++){
60         allo[n][j] = allo[n][j] + req[j];
61         need[n][j] = need[n][j] - req[j];
```

Run: question2 ×

↑ 输入申请的资源
2 0 1
↓ 第n个进程申请资源—n的值
1
↕ 第1个进程安全
第2个进程安全
第3个进程安全
第4个进程安全
第5个进程安全
同意1个进程申请资源
输入申请的资源

Build completed successfully in 1 s 917 ms (4 minutes ago) 53:1 CRLF UTF-8 4 spaces

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Wir Pupp - □ ×

Puppy > src > question2 > main

question1.java × question2.java × IOTest.java ×

```
84 int[][] need = new int[p][r];
85 int[] ava = new int[r];
86 int[] req= new int[r]; int n;
87 Scanner sc = new Scanner(System.in);
88
89 System.out.println(" 输入每个进程已分配的资源数据");
90 for(int i=0;i<p;i++)
91     for(int j=0;j<r;j++)
92         allo[i][j] = sc.nextInt();
```

Run: question2 ×

↑ 输入申请的资源
0 0 2
↓ 第n个进程申请资源—n的值
3
↕ 第1个进程安全
第2个进程安全
第3个进程安全
第4个进程安全
第5个进程安全
同意3个进程申请资源
输入申请的资源

All files are up-to-date (a minute ago) 63:1 CRLF UTF-8 4 spaces

2.5 编程中遇到的困难及解决方法、实习心得

这个代码基本实现银行家算法的所有要求，资源分配错误会提示“进程不安全，不同意分配资源”，每次进程请求后会提示是否继续，而且也会影响安全序列的排列顺序，当进程请求至所有进程执行完毕也会提示。在有了第一次实验的经验和基础，这次我对算法的理解速度更快，同时注意的点更多，使得我对计算机调度算法从自己的思考到实践有了更多的认识，在我自己思考设计银行算法时有一些死锁的机制不能很好理解，在从网站上查询了一些教学视频后，问题得以解决。

3. 实验三 动态分区式存贮区管理

3.1 实习题目：

设计一个动态分区式存贮区管理程序，要求支持不同的放置策略。如首次、最佳、最坏。

3.2 程序功能及设计思路：

3.2.1 程序功能：模拟动态分区式存贮区管理

3.2.2 设计思路：

(1) 设计一个动态分区式存贮区管理程序，支持不同的放置策略。如首次、最佳、最坏。

(2) 分区描述器中为 `flag/size/next`;

(3) 自由主存队列按链表组织，主存大小假设为 `maxsize` (单位为节=`rd` 的大小)。

(4) 作业申请 `n` 节, 实际分配的分区大小应为 `n+1` 节。 其中一节作为分区描述器，其他 `n` 节提供给作业。

(5) 已分配区放在高地址处。

(6) 合并时应考虑四种情况：

假设回收区为 `r`, 上邻为 `f1` (`f1` 需搜索自由主存队列), 下邻为 `f2` (`f2` 可直接计算)

A) `f1` 空闲, `f2` 已分配;

B) `f1` 已分配, `f2` 空闲;

C) `f1` 空闲, `f2` 空闲;

D) `f1` 已分配, `f2` 已分配;

3.3 数据结构与算法设计

3.3.1 编程语言：C++

3.3.2 数据结构：单链表

参考节点：

```
struct distrabuteArea {
    int qishidizhi;
    int daxiao;
    int zuoyehao;
    ZHUANGTAI state;           // 分区状态
    distrabuteArea* pre;       // 分区前向指针
    distrabuteArea* nxt;       // 分区后向指针
}disHead;
```

3.3.3 源代码

```
#include<iostream>
#include<iomanip>
using namespace std;
enum ZHUANGTAI { kongxian, zhanyong };
struct distrabuteArea {
    int qishidizhi;
    int daxiao;
    int zuoyehao;
    ZHUANGTAI state;          // 分区状态
    distrabuteArea* pre;      // 分区前向指针
    distrabuteArea* nxt;      // 分区后向指针
}disHead;

void intdisArea()
{
    //分配初始分区内存
    distrabuteArea* fir = (distrabuteArea*)malloc(sizeof(distrabuteArea));
    // 给首个分区赋值
    fir->qishidizhi = 0;
    fir->daxiao = 512;
    fir->state = kongxian;
    fir->zuoyehao = -1;
    fir->pre = &disHead;
    fir->nxt = NULL;
    // 初始化分区头部信息
    disHead.pre = NULL;
    disHead.nxt = fir;
}

int shoucishiying(int zuoyehao, int daxiao)
{
    distrabuteArea* p = disHead.nxt;
    while (p != NULL)
    {
        if (p->state == kongxian && p->daxiao >= daxiao) {
            // 找到要分配的空闲分区
            {
                // 分配大小为daxiao的区间
                distrabuteArea* node = (distrabuteArea*)malloc(sizeof(distrabuteArea));
                node->qishidizhi = p->qishidizhi + daxiao;
                node->daxiao = p->daxiao - daxiao;
                node->state = kongxian;
```

```

        node->zuoyehao = -1;
        // 修改分区链节点指针
        node->pre = p;
        node->nxt = p->nxt;
        if (p->nxt != NULL) {
            p->nxt->pre = node;
        }
        p->nxt = node;
        // 分配空闲区间
        p->daxiao = daxiao;
        p->state = zhanyong;
        p->zuoyehao = zuoyehao;

    }
    cout << "分配成功" << endl;
    return 1;
}

p = p->nxt;
}

cout << "找不到合适的内存分区，分配失败" << endl;
return 0;
}

int zuijia(int zuoyehao, int daxiao)
{
    distrabuteArea* tar = NULL;
    int tarSize = 512 + 1;
    distrabuteArea* p = disHead.nxt;
    while (p != NULL)
    {
        // 寻找最佳空闲区间
        if (p->state == kongxian && p->daxiao >= daxiao && p->daxiao < tarSize) {
            tar = p;
            tarSize = p->daxiao;
        }
        p = p->nxt;
    }

    if (tar != NULL) {
        // 找到要分配的空闲分区
        {
            // 分配大小为daxiao的区间
            distrabuteArea* node = (distrabuteArea*)malloc(sizeof(distrabuteArea));
            node->qishidizhi = tar->qishidizhi + daxiao;

```

```

        node->daxiao = tar->daxiao - daxiao;
        node->state = kongxian;
        node->zuoyehao = -1;
        // 修改分区链节点指针
        node->pre = tar;
        node->nxt = tar->nxt;
        if (tar->nxt != NULL) {
            tar->nxt->pre = node;
        }
        tar->nxt = node;
        // 分配空闲区间
        tar->daxiao = daxiao;
        tar->state = zhanyong;
        tar->zuoyehao = zuoyehao;
    }
    cout << "分配成功" << endl;
    return 1;
}
else {
    cout << "找不到合适的内存分区，分配失败" << endl;
    return 0;
}
}

int zuihuai(int zuoyehao, int daxiao)
{
    distrabuteArea* tar = NULL;
    int tarSize = 512 + 1;
    distrabuteArea* p = disHead.nxt;
    while (p != NULL)
    {
        // 寻找最大空闲区间
        int tempsize=0;
        if (p->state == kongxian && p->daxiao >= tempsize) {
            tar = p;
            tarSize = p->daxiao;
            tempsize = p->daxiao;
        }
        p = p->nxt;
    }
    if (tar != NULL && tarSize >= daxiao) {
        // 找到要分配的空闲分区
        {
            // 分配大小为daxiao的区间

```

```

        distrabuteArea* node = (distrabuteArea*)malloc(sizeof(distrabuteArea));
        node->qishidizhi = tar->qishidizhi + daxiao;
        node->daxiao = tar->daxiao - daxiao;
        node->state = kongxian;
        node->zuoyehao = -1;
        // 修改分区链节点指针
        node->pre = tar;
        node->nxt = tar->nxt;
        if (tar->nxt != NULL) {
            tar->nxt->pre = node;
        }
        tar->nxt = node;
        // 分配空闲区间
        tar->daxiao = daxiao;
        tar->state = zhanyong;
        tar->zuoyehao = zuoyehao;
    }
    cout << "分配成功" << endl;
    return 1;
}
else {
    cout << "找不到合适的内存分区，分配失败" << endl;
    return 0;
}
}

```

```

int neicunhuishou(int zuoyehao)
{
    int flag = 0;
    distrabuteArea* p = disHead.nxt, * pp;
    while (p != NULL)
    {
        if (p->state == zhanyong && p->zuoyehao == zuoyehao) {
            flag = 1;
            if ((p->pre != &disHead && p->pre->state == kongxian)
                && (p->nxt != NULL && p->nxt->state == kongxian)) {
                // 上下都是空闲分区
                // 先合并上区间
                pp = p;
                p = p->pre;
                p->daxiao = p->daxiao + pp->daxiao;
                p->nxt = pp->nxt;
                pp->nxt->pre = p;
                free(pp);
            }
        }
    }
}

```

```

        // 后合并下区间
        pp = p->nxt;
        p->daxiao = p->daxiao + pp->daxiao;
        p->nxt = pp->nxt;
        if (pp->nxt != NULL) {
            pp->nxt->pre = p;
        }
        free(pp);
    }
    else if ((p->pre == &disHead || p->pre->state == zhanyong)
        && (p->nxt != NULL && p->nxt->state == kongxian)) {
        // 下面是空闲分区
        pp = p->nxt;
        p->daxiao = p->daxiao + pp->daxiao;
        p->state = kongxian;
        p->zuoyehao = -1;
        p->nxt = pp->nxt;
        if (pp->nxt != NULL) {
            pp->nxt->pre = p;
        }
        free(pp);
    }
    else if ((p->pre != &disHead && p->pre->state == kongxian)
        && (p->nxt == NULL || p->nxt->state == zhanyong)) {
        // 上面是空闲分区
        pp = p;
        p = p->pre;
        p->daxiao = p->daxiao + pp->daxiao;
        p->nxt = pp->nxt;
        if (pp->nxt != NULL) {
            pp->nxt->pre = p;
        }
        free(pp);
    }
    else {
        // 上下都是占用分区
        p->state = kongxian;
        p->zuoyehao = -1;
    }
}
p = p->nxt;
}
if (flag == 1) {
    cout << "回收成功" << endl;
}

```

```

        return 1;
    }
    else {
        cout << "没有目标作业" << endl;
        return 0;
    }
}

void xianshi()
{
    cout << "当前的内存分配情况如下：" << endl;
    cout << endl;
    cout << "起始地址" << " " << "空间大小" << " " << "工作状态" << " " << "作业号" << endl;
    distrabuteArea* p = disHead.nxt;
    while (p != NULL)
    {
        cout << endl;
        cout << p->qishidizhi << '\t';
        cout << p->daxiao << '\t';
        printf("%s \t", p->state == kongxian ? "空闲" : "占用");
        if (p->zuoyehao > 0) {
            cout << p->zuoyehao;
        }
        else {
            cout << " ";
        }
        cout << endl;
        p = p->nxt;
    }
}

int main()
{
    int option, ope, zuoyehao, daxiao;
    intdisArea();
    while (1)
    {
        cout << "0. 首次适应算法" << endl;
        cout << "1. 最佳适应算法" << endl;
        cout << "2. 最坏适应算法" << endl;
        cin >> option;
        if (option == 0) {

```

```

        cout << "首次适应算法" << endl;
        break;
    }
    else if (option == 1) {
        cout << "最佳适应算法" << endl;
        break;
    }
    else if (option == 2) {
        cout << "最坏适应算法" << endl;
        break;
    }
    else {
        cout << "错误: 请输入 0/1" << endl;
    }
}
while (1)
{
    cout << endl;
    cout << " 1: 申请分区      2: 释放分区      0: 退出" << endl;
    cin >> ope;
    if (ope == 0) break;
    if (ope == 1) {
        cout << "请输入作业号: " << endl;
        cin >> zuoyehao;
        cout << "请输入作业申请的内存大小: " << endl;
        int size0;
        cin >> size0;
        daxiao = size0 + 1;
        if (daxiao <= 0) {
            cout << "错误: 分配内存大小必须为正值" << endl;
            continue;
        }
        if (option == 0) {
            shoucishiying(zuoyehao, daxiao);
        }
        else if (option == 1) {
            zuijia(zuoyehao, daxiao);
        }
        else {
            zuihuai(zuoyehao, daxiao);
        }
        xianshi();
    }
    else if (ope == 2) {

```



```

        cout << "请输入要回收的作业号：" << endl;
        cin >> zuoyehao;
        neicunhuishou(zuoyehao);
        xianshi();
    }
    else {
        cout << "输入错误" << endl;
    }
}

cout << "结束" << endl;
return 0;
}

```

3.4 程序运行情况

```

C:\Users\christan\source\repos\ConsoleApplication1\D...
1: 申请分区      2: 释放分区      0: 退出
1
请输入作业号:
10
请输入作业申请的内存大小:
42
分配成功
当前的内存分配情况如下:
起始地址 空间大小 工作状态      作业号
0         73      zhanyong        5
73        37      zhanyong        7
110       61      zhanyong        8
171       43      zhanyong        10
214       14      kongxian
228       101     zhanyong        6
329       111     zhanyong        9
440       72      kongxian
1: 申请分区      2: 释放分区      0: 退出

```

```

C:\Users\christan\source\repos\Console...
1
请输入作业号:
10
请输入作业申请的内存大小:
42
分配成功
当前的内存分配情况如下:
起始地址 空间大小 工作状态      作业号
0         37      占用          7
37        61      占用          8
98        111     占用          9
209       19      空闲
228       101     占用          6
329       43      占用          10
372       64      空闲
436       73      占用          5
509       3       空闲
1: 申请分区      2: 释放分区      0: 退出

```

```
C:\Users\christan\source\repos\Cons...
找不到合适的内存分区，分配失败
当前的内存分配情况如下：

起始地址 空间大小 工作状态      作业号
0          436     空闲
436        73     占用          5
509        3     空闲

1: 申请分区      2: 释放分区      0: 退出
1
请输入作业号：
10
请输入作业申请的内存大小：
42
找不到合适的内存分区，分配失败
当前的内存分配情况如下：

起始地址 空间大小 工作状态      作业号
0          436     空闲
436        73     占用          5
509        3     空闲

1: 申请分区      2: 释放分区      0: 退出
```

3.5 编程中遇到的困难及解决方法、实习心得

存储数据是操作系统很重要的功能之一，而要保证数据存储的高效以及提高磁盘资源的利用率，对存储区的管理就显得十分的重要。文件总是数量巨大并且各个文件大小不一，所占用的硬盘空间也各不相同，不加管理的任其存放最终就会导致磁盘空间的碎片化严重，极其浪费硬件资源。在对存储区管理的学习中，了解到了各种管理算法，也让我对操作系统的磁盘管理有了更深刻的理解。一个好的分区管理算法，不仅可以提高磁盘率，并且也可以明显的提升磁盘资源的利用率。