南通大学计算机学院

**操作系统 课程设计**

**报 告 书**

**设计题目 优先级调度算法**

**可变分区管理算法**

专业班级 网工 151

学生姓名 王世家

学 号 1513042032

指导教师 周建美

日 期 2017年6月30日

**实验报告正文内容**

1. **课程设计的目的**

1、模拟实现计算机的处理机调度，帮助学生理解处理机调度的原理和过程。

2、模拟实现计算机主存空间的分配和回收，帮助学生理解主存空间分配和回收的原理和过程。

1. **课程设计内容**

1、设计一个按优先级调度的程序。

2、设计一个最先适应算法实现主存空间的分配和回收的程序。

1. **算法设计**

进程调度采用以优先级为基础的调度算法，可变分区采用First-Fit、Best-Fit、Worst-First算法。 **4、测试结果**

1）优先级调度算法

图1：开始时间相同，优先级不同

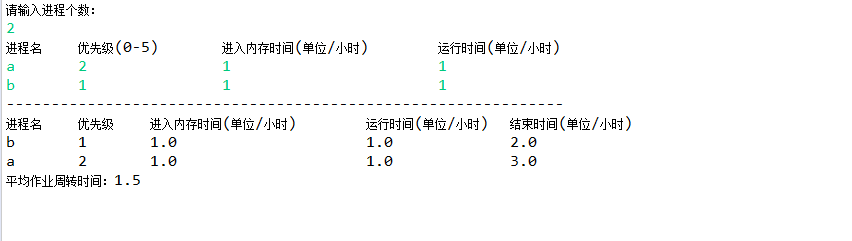


图2：开始时间不相同，优先级相同

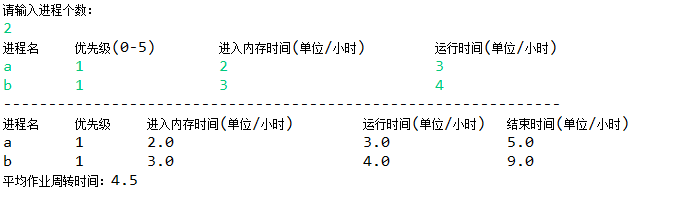


图3：开始时间不相同，优先级不相同

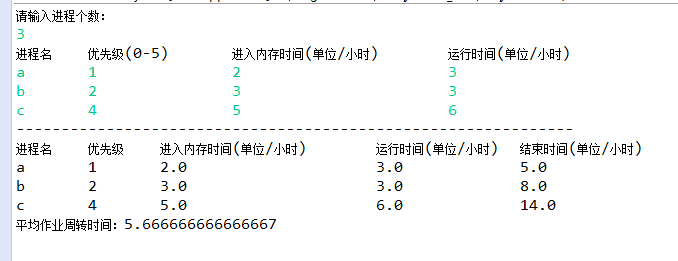
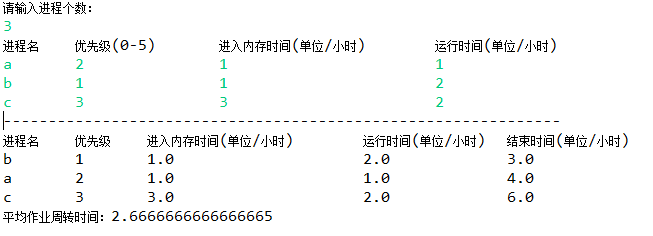
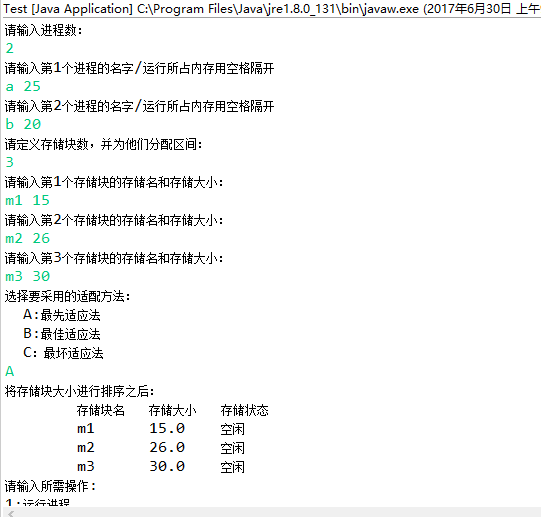
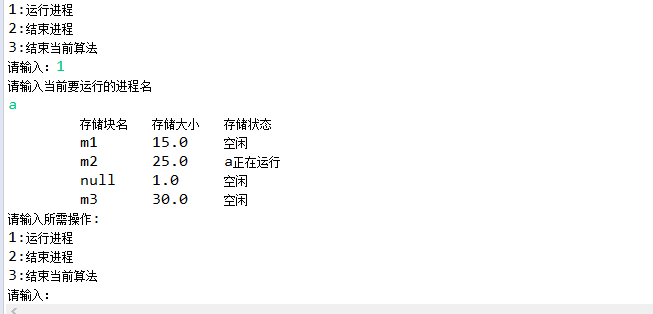
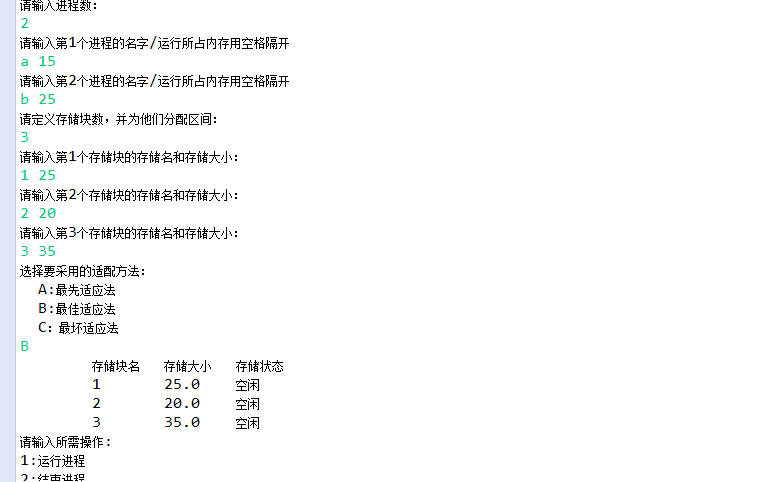


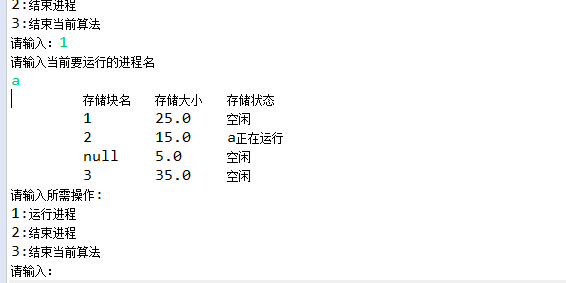
图4：综上



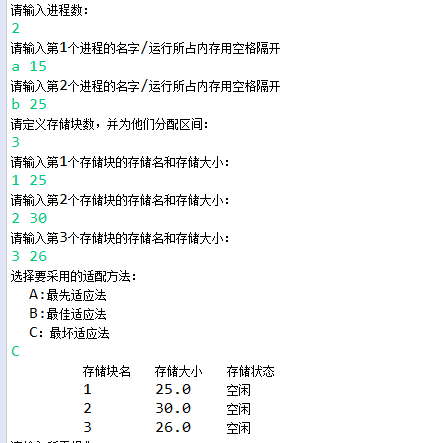
1. 可变分区管理算法

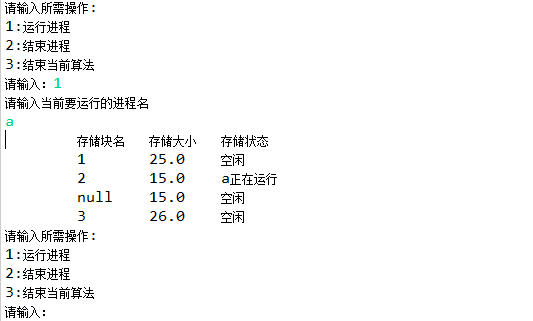
1>First-Fit

2>Best-Fit 



3>Worst-Fit





**5、实验小结**

本次实验有利于加深对优先级调度算法、可变分区管理算法的理解，动手能力较强。编程时间较短，排错误的时间较长。

附：程序源代码

//进程定义

package cn.edu.ntu;

public class Procedure {

String procedureName; // 定义进程优先级

int priority; // 进程优先级

double startTime; //开始时间

double runTime; // 运行时间

double terminalTime; //结束时间

public Procedure() {}

public Procedure(String procedureName, int priority, double startTime, double runtime, double terminalTime) {

super();

this.procedureName = procedureName;

this.priority = priority;

this.startTime = startTime;

this.runTime = runtime;

this.terminalTime = terminalTime;

}

}

//优先级调度算法

package cn.edu.ntu; //包名：公司域名反转

import java.util.HashMap;

import java.util.Scanner;

public class PriorityTest {

public static void function() {

Scanner sc = new Scanner(System.in);

HashMap<String,Procedure> hash = new HashMap<>();

Procedure temp1 = new Procedure();

Procedure temp2 = new Procedure();

System.out.println("请输入进程个数：");

int num = sc.nextInt();

Procedure p[] = new Procedure[num]; //创建含有num 个进程的数组

double startTime[] = new double[num]; //创建优先级数组，存放进程优先级

double average = 0; //平均作业周转时间

double sum = 0;

String str[] = new String[num];

String str2;

System.out.println("进程名\t" + "优先级(0-5)\t" + "进入内存时间(单位/小时)\t" + "运行时间(单位/小时)\t");

int sign = 0;

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

p[i] = new Procedure();

p[i].procedureName = sc.next();

str[i] = p[i].procedureName;

hash.put(p[i].procedureName, p[i]);

p[i].priority = sc.nextInt();

p[i].startTime = sc.nextDouble();

startTime[i] = p[i].startTime;

p[i].runTime = sc.nextDouble();

}

for(int i = 0;i < startTime.length - 1; i++) {

sign ++;

if(i < startTime.length - 1) {

if(startTime[i] < startTime[i + 1]) {

if(i == 0) {

p[i].terminalTime = p[i].startTime + p[i].runTime;

} else if(p[i - 1].terminalTime < p[i].startTime) {

p[i].terminalTime = p[i].startTime + p[i].runTime;

} else if(p[i - 1].terminalTime >= p[i].startTime)

{

p[i].terminalTime = p[i - 1].terminalTime + p[i].runTime;

}

} else if(startTime[i] == startTime[i + 1])

{

if(p[i].priority > p[i + 1].priority) {

temp1 = p[i + 1];

p[i + 1] = p[i];

p[i] = temp1;

str2 = str[i + 1];

str[i + 1] = str[i];

str[i] = str2;

}

if(i == 0) {

p[i].terminalTime = p[i].startTime + p[i].runTime;

} else if(sign < startTime.length - 1)

{

if(p[i - 1].terminalTime < p[i].startTime) {

p[i].terminalTime = p[i].startTime + p[i].runTime;

} else if(p[i - 1].terminalTime >= p[i].startTime)

{

p[i].terminalTime = p[i - 1].terminalTime + p[i].runTime;

}

} else if(sign == startTime.length - 1) {

if(p[sign - 1].terminalTime < p[sign].startTime) {

p[sign].terminalTime = p[sign].startTime + p[sign].runTime;

} else if(p[sign - 1].terminalTime >= p[sign].startTime) {

p[sign].terminalTime = p[sign - 1].terminalTime + p[sign].runTime;

}

}

}

}

}

if(sign == 0) {

p[sign].terminalTime = p[sign].startTime + p[sign].runTime;

} else if(sign == startTime.length - 1)

{

if(p[sign - 1].terminalTime > p[sign].startTime) {

p[sign].terminalTime = p[sign - 1].terminalTime + p[sign].runTime;

} else

{

p[sign].terminalTime = p[sign].startTime + p[sign].runTime;

}

}

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

sum += p[i].terminalTime - p[i].startTime;

}

average = sum / num;

System.out.println("--------------------------------------------------------------");

System.out.println("进程名\t" + "优先级\t" + "进入内存时间(单位/小时)\t" + "运行时间(单位/小时)\t" + "结束时间(单位/小时)\t");

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

temp2 = hash.get(str[i]);

System.out.println(temp2.procedureName + "\t" + temp2.priority + "\t" +

temp2.startTime + "\t" + "\t" + "\t" + temp2.runTime +

"\t"+ "\t" + temp2.terminalTime);

}

System.out.println("平均作业周转时间：" + average);

sc.close();

}

public static void main(String[] args) {

PriorityTest test = new PriorityTest();

test.function();

}

}

// 可变分区进程定义

package cn.edu.ntu;

public class ProcedureTest2 {

String procedureName;

double procedureSize;

public ProcedureTest2() {

}

public ProcedureTest2(String procedureName, double procedureSize) {

super();

this.procedureName = procedureName;

this.procedureSize = procedureSize;

}

}

//存储块定义

package cn.edu.ntu;

public class MemoryTest {

String memoryName;

double memorySize;

String station = "空闲";

public String getName() {

return memoryName;

}

public void setName(String name) {

this.memoryName = memoryName;

}

public String getMemoryName() {

return memoryName;

}

public void setMemoryName(String memoryName) {

this.memoryName = memoryName;

}

public double getMemorySize() {

return memorySize;

}

public void setMemorySize(double memorySize) {

this.memorySize = memorySize;

}

public String getStation() {

return station;

}

public void setStation(String station) {

this.station = station;

}

}

//排序算法

package cn.edu.ntu;

import java.util.ArrayList;

public class MethodTest {

public void BubbleSort(ArrayList<MemoryTest> array) {

System.*out*.println("将存储块大小进行排序之后：");

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

for (int j = i; j < array.size(); j++) {

if (array.get(i).memorySize > array.get(j).memorySize) {

Object obj = array.get(i);

array.set(i, array.get(j));

array.set(j, (MemoryTest) obj);

}

}

}

}

public static void Output(ArrayList<MemoryTest> array2) { // 输出

System.*out*.println("\t存储块名\t存储大小\t存储状态");

for (int i = 0; i < array2.size(); i++) {

System.*out*.println(

"\t" + array2.get(i).memoryName + "\t" + array2.get(i).memorySize + "\t" + array2.get(i).station);

}

}

}

//主程序

package cn.edu.ntu;

import java.util.ArrayList;

import java.util.Scanner;

import java.util.Vector;

class Access {

Scanner input = new Scanner(System.in);

MethodTest method = new MethodTest();

public void mainMethod(String str, Vector<ProcedureTest2> pcbs, ArrayList<MemoryTest> list) {

while (true) {

System.out.println("请输入所需操作:");

System.out.println("1:运行进程");

System.out.println("2:结束进程");

System.out.println("3:结束当前算法");

System.out.print("请输入：");

int p = input.nextInt();

if (p == 1) {

System.out.println("请输入当前要运行的进程名");

String name = input.next();

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

if (name.equals(pcbs.get(i).procedureName)) {

if (str.equals("A")) {

for (int j = 0; j < list.size(); j++) {

if (list.get(j).memorySize >= pcbs.get(i).procedureSize

&& list.get(j).station.equals("空闲")) {

MemoryTest memorys = new MemoryTest();

list.get(j).setStation(name + "正在运行");

double remainder = 0;

remainder = list.get(j).memorySize - pcbs.get(i).procedureSize;

if (remainder != 0) {

list.get(j).setMemorySize(pcbs.get(i).procedureSize);

list.add(j + 1, memorys);

list.get(j + 1).setName("剩余存储块大小");

list.get(j + 1).setMemorySize(remainder);

}

MethodTest.Output(list);

break;

}

}

}

/\*最佳适应算法是从全部空闲区中找出能满足作业要求的、且大小最小的空闲分区的一种计算方法，

使碎片尽量小。\*/

else if (str.equals("B")) {

MemoryTest memorys = new MemoryTest();

double remainder = 100;

int ss = -1;

for (int j = 0; j < list.size(); j++) {

if (list.get(j).memorySize >= pcbs.get(i).procedureSize

&& list.get(j).station.equals("空闲")) {

if ((list.get(j).memorySize - pcbs.get(i).procedureSize) < remainder) {

remainder = (list.get(j).memorySize - pcbs.get(i).procedureSize);

ss = j;

}

}

}

if (remainder != -1) {

list.get(ss).setStation(name + "正在运行");

list.get(ss).setMemorySize(pcbs.get(i).procedureSize);

if (remainder != 0) {

list.add(ss + 1, memorys);

list.get(ss + 1).setName("剩余存储块大小");

list.get(ss + 1).setMemorySize(remainder);

}

}

MethodTest.Output(list);

break;

}

//最坏适应分配算法要扫描整个空闲分区或链表，总是挑选一个最大的空闲分区分割给作业使用。

else if (str.equals("C")) {

MemoryTest memorys = new MemoryTest();

double remainder = -1;

int ss = -1;

for (int j = 0; j < list.size(); j++) {

if (list.get(j).memorySize > pcbs.get(i).procedureSize

&& list.get(j).station.equals("空闲")) {

if ((list.get(j).memorySize - pcbs.get(i).procedureSize) > remainder) {

remainder = list.get(j).memorySize - pcbs.get(i).procedureSize;

ss = j;

}

}

}

if (ss != -1) {

list.get(ss).setStation(name + "正在运行");

list.get(ss).setMemorySize(pcbs.get(i).procedureSize);

if (remainder != 0) {

list.add(ss + 1, memorys);

list.get(ss + 1).setName("剩余存储块大小");

list.get(ss + 1).setMemorySize(remainder);

}

}

MethodTest.Output(list);

break;

}

}

}

} else if (p == 2) {

System.out.println("请输入要结束的进程名");

String name = input.next();

String names = name + "正在运行";

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

if (names.equals(list.get(i).station)) {

list.get(i).setStation("空闲");

if (list.get(i + 1).memoryName.equals("字存储块")) {

list.get(i).memorySize = list.get(i).memorySize + list.get(i + 1).memorySize;

list.remove(i + 1);

}

MethodTest.Output(list);

break;

}

}

} else if (p == 0) {

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

if (list.get(i).getStation() != "空闲") {

list.get(i).setStation("空闲");

if (list.get(i + 1).memoryName.equals("字存储块")) {

list.get(i).memorySize = list.get(i).memorySize + list.get(i + 1).memorySize;

list.remove(i + 1);

}

}

}

break;

}

}

}

}

public class Test {

private static Scanner input;

public static void main(String args[]) {

Vector<ProcedureTest2> pcbs = new Vector<ProcedureTest2>();

System.out.println("请输入进程数：");

input = new Scanner(System.in);

int n = input.nextInt();

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

ProcedureTest2 pcb = new ProcedureTest2();

System.out.println("请输入第" + (i + 1) + "个进程的名字/运行所占内存用空格隔开");

pcb.procedureName = input.next();

pcb.procedureSize = input.nextDouble();

pcbs.add(pcb);

}

System.out.println("请定义存储块数，并为他们分配区间：");

int m = input.nextInt();

ArrayList<MemoryTest> list = new ArrayList<MemoryTest>();

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

MemoryTest nck = new MemoryTest();

System.out.println("请输入第" + (i + 1) + "个存储块的存储名和存储大小：");

nck.memoryName = input.next();

nck.memorySize = input.nextInt();

list.add(nck);

}

while (true) {

System.out.println("选择要采用的适配方法：");

System.out.println(" A:最先适应法");

System.out.println(" B:最佳适应法");

System.out.println(" C：最坏适应法");

String str = input.next();

MethodTest methid = new MethodTest();

if (str.equals("A")) {

methid.BubbleSort(list);

MethodTest.Output(list);

Access access = new Access();

access.mainMethod(str, pcbs, list);

} else if (str.equals("B")) {

MethodTest.Output(list);

Access access = new Access();

access.mainMethod(str, pcbs, list);

} else if (str.equals("C")) {

MethodTest.Output(list);

Access access = new Access();

access.mainMethod(str, pcbs, list);

}

}

}

}