

计算机与信息学院实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 操作系统实验 | | | | |
| 实验编号： |  | | | | |
| 实验名称： | 编程实现四种调度算法 | | | | |
| 实验人员： | 年级 | | 2017 | | |
| 专业 | | 软件工程 | | |
| 学号 | | 17111205043 | | |
| 姓名 | | 李志威 | | |
| 实验日期： | 2019.4.9 | | | | |
| 上交日期： | 2019.5.6 | | | | |
| 实 验 室： | 2030201 | | | | |
| 实验评价： |  | | | | |
|  | 实验成绩： |  | | 评定日期： |  |
|  | 指导教师： |  | | | |

一、实验目的

编程实现四种调度算法：

(1) 先来先服务算法

(2) 短作业优先算法

(3) 优先权算法

(4) 基于时间片的多级反馈队列算法

二、实验环境

Java eclipse win10

三、实验内容与步骤

⒈问题描述

编程实现四种调度算法：

(1) 先来先服务算法

(2) 短作业优先算法

(3) 优先权算法

(4) 基于时间片的多级反馈队列算法

⒉基本要求

(1) 通过若干个实例实现各种算法的优劣性对比；

(2) 结果要求可视化展示

⒊实现提示

根据课本所给的四种算法的原理，编程分别实现。在给出相同的进程数、进程名以及相同的到达时间和服务时间的情况下，比较四种算法调度后各进程的完成时间、周转时间和带权周转时间。

**4.** 输入和输出

1）输入

进程数进程名以及各个进程的到达时间和服务时间，基于时间片的多级反馈队列算法确定的时间片的大小。

2）输出（两种形式）

每种算法有如下输出：各个进程名以及对应的完成时间、周转时间和带权周转时间。

四、实验结果与分析

import java.util.Scanner;

/\*\*

\* 模拟PCB调度进程运行

\* (1) 先来先服务算法

\* (2) 短作业优先算法

\* (3) 优先权算法 (非抢占式，静态优先级)

\* (4) 基于时间片的多级反馈队列算法

\* @author LiZhiWei

\* 2019.4.29

\*/

public class main {

public static void main(String[] args) {

PCB pcb1 = new PCB();

PCB pcb2 = new PCB();

PCB pcb3 = new PCB();

PCB pcb4 = new PCB();

Scanner scanner = new Scanner(System.in);

System.out.println("进程模拟系统");

System.out.print("请输入进程的个数:");

int n = scanner.nextInt();

scanner.nextLine();

System.out.println("请依次输入进程的编号、名称、需要的时间片、优先级、到达时间，以“，”分隔，每个进程一行");

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

String s = scanner.nextLine();

String[] split = s.split(",");

pcb1.createProcess(Integer.parseInt(split[0]), split[1], Integer.parseInt(split[2]), Integer.parseInt(split[3]));

pcb2.createProcess(Integer.parseInt(split[0]), split[1], Integer.parseInt(split[2]), Integer.parseInt(split[3]));

pcb3.createProcess(Integer.parseInt(split[0]), split[1], Integer.parseInt(split[2]), Integer.parseInt(split[3]));

pcb4.createProcess(Integer.parseInt(split[0]), split[1], Integer.parseInt(split[2]), Integer.parseInt(split[3]));

}

System.out.println("先来先服务");

pcb1.FCFS();//先来先服务

System.out.println("");System.out.println("短作业优先");

pcb2.SJF();//短作业优先

System.out.println("");System.out.println("优先权算法");

pcb3.PSA();//优先权算法

System.out.println("");System.out.println("基于时间片的多级反馈队列算法");

pcb4.MFQ();//基于时间片的多级反馈队列算法

}

}

public class Process {

int id;//标识符

String name;//名称

enum State{READY,RUNNING,BLOCK,DIE}

State state;//状态

int time;//总共需要的时间片

int RTime;//还需执行的时间片

int priority;//优先级

/\*\*

\* 创建进程

\*/

public Process(int id, String name, int time, int priority) {

this.id = id;

this.name = name;

this.time = time;

this.priority = priority;

state = State.READY;

RTime=time;

}

public Process() {

}

/\*\*

\* 删除进程

\*/

public void delete() {

id = 0;

name = null;

state = null;

time = 0;

priority = 0;

}

/\*\*

\* 调度进程

\*/

public void call() {

state = State.RUNNING;

RTime--;

}

/\*\*

\* 阻塞进程

\*/

public void block() {

state = State.BLOCK;

}

/\*\*

\* 时间到

\*/

public void timeOut() {

state = State.DIE;

}

/\*\*

\* 激活

\*/

public void active() {

state = State.READY;

}

}

import java.util.Queue;

import java.util.Random;

import java.util.concurrent.ArrayBlockingQueue;

public class PCB {

Queue<Process> readyQueue = new ArrayBlockingQueue(100);

Queue<Process> blockQueue = new ArrayBlockingQueue(100);

Queue<Process> dieQueue = new ArrayBlockingQueue(100);

Queue<Process> timeQueue = new ArrayBlockingQueue(100);

/\*\*

\* 创建进程

\*/

public void createProcess(int id, String name, int time, int priority) {

Process process = new Process(id, name, time, priority);

readyQueue.add(process);

}

/\*\*

\* 加入就绪队列

\*/

public void addReadyQueue(Process p) {

readyQueue.add(p);

}

/\*\*

\* 从就绪队列取出

\*/

public Process removeReadyQueue() {

return readyQueue.remove();

}

/\*\*

\* 加入阻塞队列

\*/

public void addBlockQueue(Process p) {

p.block();

blockQueue.add(p);

}

/\*\*

\* 从阻塞队列唤醒，并加入就绪队列

\*/

public Process removeBlockQueue() {

Process p = blockQueue.remove();

p.active();

return p;

}

/\*\*

\* 加入消亡队列

\*/

public void addDieQueue(Process p) {

p.timeOut();;

dieQueue.add(p);

}

public void removeDieQueue() {

for (int i = dieQueue.size(); i >0 ; i--) {

dieQueue.remove();

}

}

/\*\*

\* 显示各队列状态

\*/

public void printState() {

System.out.print("就绪队列：");

for (Object i : readyQueue) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.print("\n阻塞队列：");

for (Object i : blockQueue) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.print("\n死亡队列：");

for (Object i : dieQueue) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.println("");

}

/\*

\* 显示多级队列状态

\*/

public void printStateMFQ(Queue<Process> readyQueue1,Queue<Process> readyQueue2,Queue<Process> readyQueue3) {

System.out.print("就绪队列1：");

for (Object i : readyQueue1) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.print("\n就绪队列2：");

for (Object i : readyQueue2) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.print("\n就绪队列3：");

for (Object i : readyQueue3) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.print("\n阻塞队列：");

for (Object i : blockQueue) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.print("\n死亡队列：");

for (Object i : dieQueue) {

Process p = (Process) i;

System.out.print(p.name);

System.out.print(" ");

}

System.out.println("");

}

/\*\*

\* 冒泡排序对队列进行优先级排序(降序)

\* \* @param q

\*/

public void sortP(Queue<Process> q) {

if (!q.isEmpty()) {

Process[] obj = new Process[q.size()];

int readyNumber = q.size();

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

obj[i] = q.remove();

}

while (readyNumber > 0) {

Process temp = new Process();

int i = 0, j = 1;

while (j < readyNumber) {

if (obj[i].priority > obj[j].priority) {

temp = obj[i];

obj[i] = obj[j];

obj[j] = temp;

}

i++;

j++;

}

q.add(obj[i]);

readyNumber--;

}

}

}

/\*\*

\* 冒泡排序对队列进行时间片排序(升序)

\* \* @param q

\*/

public void sortT(Queue<Process> q) {

if (!q.isEmpty()) {

Process[] obj = new Process[q.size()];

int readyNumber = q.size();

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

obj[i] = q.remove();

}

while (readyNumber > 0) {

Process temp = new Process();

int i = 0, j = 1;

while (j < readyNumber) {

if (obj[i].time < obj[j].time) {

temp = obj[i];

obj[i] = obj[j];

obj[j] = temp;

}

i++;

j++;

}

q.add(obj[i]);

readyNumber--;

}

}

}

public void addRQ(int timeFlag,Queue<Process> q) {

for (int i = timeQueue.size(); i >0 ; i--) {

Process p =timeQueue.remove();

if(p.time<=timeFlag) {

q.add(p);

}else {

timeQueue.add(p);

}

}

}

/\*

\* 先来先服务

\*/

public void FCFS() {

int timeFlag = 0;

while (!readyQueue.isEmpty() || !blockQueue.isEmpty()) {

int time = 0;

Process p = removeReadyQueue();

while(true) {

p.call();

time++;

if (p.RTime <= 0) {

p.timeOut();

addDieQueue(p);

break;

}

}

int n=timeFlag+time;

System.out.println(timeFlag+"--进程 "+p.name+" --> "+n);

timeFlag += time;

}

System.out.println("运行结束，总时间片: "+timeFlag);

}

/\*

\* 短作业优先

\*/

public void SJF() {

sortT(readyQueue);

int timeFlag = 0;

while (!readyQueue.isEmpty() || !blockQueue.isEmpty()) {

int time = 0;

Process p = removeReadyQueue();

while(true) {

p.call();

time++;

if (p.RTime <= 0) {

p.timeOut();

addDieQueue(p);

break;

}

}

int n=timeFlag+time;

System.out.println(timeFlag+"--进程 "+p.name+" --> "+n);

timeFlag += time;

sortT(readyQueue);

sortT(blockQueue);

}

System.out.println("运行结束，总时间片: "+timeFlag);

}

/\*

\* 优先权算法

\*/

public void PSA() {

sortP(readyQueue);

int timeFlag = 0;

while (!readyQueue.isEmpty() || !blockQueue.isEmpty()) {

int time = 0;

Process p = removeReadyQueue();

while(true) {

p.call();

time++;

if (p.RTime <= 0) {

p.timeOut();

addDieQueue(p);

break;

}

}

int n=timeFlag+time;

System.out.println(timeFlag+"--进程 "+p.name+" --> "+n);

timeFlag += time;

sortP(readyQueue);

sortP(blockQueue);

// printState();

}

System.out.println("运行结束，总时间片: "+timeFlag);

}

/\*

\* 基于时间片的多级反馈队列算法

\*/

public void MFQ() {

Queue<Process> readyQueue1 = new ArrayBlockingQueue(100);

Queue<Process> readyQueue2 = new ArrayBlockingQueue(100);

Queue<Process> readyQueue3 = new ArrayBlockingQueue(100);

int timeFlag=0;

while(!readyQueue.isEmpty()) {

readyQueue1.add(readyQueue.remove());

}

while(!readyQueue1.isEmpty()) {

int time=0;

Process p = readyQueue1.remove();

for(int j=0;j<2;j++) {

p.call();

time++;

addRQ(time+timeFlag,readyQueue1);

if (p.RTime <= 0) {

p.timeOut();

addDieQueue(p);

break;

}

}

if(p.RTime>0 && p.state==Process.State.RUNNING) {

readyQueue2.add(p);

}

int n=timeFlag+time;

System.out.println(timeFlag+"--进程 "+p.name+" --> "+n);

timeFlag+=time;

}

printStateMFQ(readyQueue1,readyQueue2,readyQueue3);

System.out.println("");

while(!readyQueue2.isEmpty()) {

int time =0;

Process p = readyQueue2.remove();

for(int j=0;j<4;j++) {

p.call();

time++;

addRQ(time+timeFlag,readyQueue1);

if (p.RTime <= 0) {

p.timeOut();

addDieQueue(p);

break;

}

}

if(p.RTime>0 && p.state==Process.State.RUNNING) {

readyQueue3.add(p);

}

int n=timeFlag+time;

System.out.println(timeFlag+"--进程 "+p.name+" --> "+n);

timeFlag+=time;

}

printStateMFQ(readyQueue1,readyQueue2,readyQueue3);

System.out.println("");

while(!readyQueue3.isEmpty()) {

int time=0;

Process p;

p = readyQueue3.remove();

for(int j=0;j<6;j++) {

p.call();

time++;

addRQ(time+timeFlag,readyQueue1);

if (p.RTime <= 0) {

p.timeOut();

addDieQueue(p);

break;

}

}

if(p.RTime>0) {

readyQueue3.add(p);

}

int n=timeFlag+time;

System.out.println(timeFlag+"--进程 "+p.name+" --> "+n);

timeFlag+=time;

}

printStateMFQ(readyQueue1,readyQueue2,readyQueue3);

System.out.println("");

System.out.println("运行结束，总时间片: "+timeFlag);

}

}



