Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра ИПМ

**«Операционные системы»**

**Лабораторная работа № 1**

Вариант № 14

Выполнила: Сорокин Ю.Б.

Группа: Р3317

Преподаватель: Лаздин А.В.

Санкт-Петербург

2016 год

**Цель работы**

Изучить основные алгоритмы планирования процессов.

**Задание**

В соответствии с вариантом задания реализовать программную имитацию реализации алгоритма планирования (2 шт.).

Программно подготовить исходные данные, представляющие собой пары чисел, где первое число – время поступления процесса, а второе – время его выполнения. Очевидно, что последовательность первых значений – возрастающая (временные парадоксы запрещены).

Диапазон времен выполнения, и среднее время между поступающими процессами взять из варианта задания.

Число пар <время поступления> – <длительность выполнения>не менее двадцати. Для полученных данных определить среднее время оборота процессов и среднее время ожидания процессов, в соответствии с реализованными алгоритмами планирования. Сделать вывод о предпочтительном алгоритме планирования, считая критерием эффективности минимальное среднее время оборота процессов для нечетных вариантов, и минимальное среднее время ожидания для четных вариантов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер варианта** | **Алгоритм 1** | **Алгоритм 2** | **Диапазон длительностей выполнения** | **Среднее время между поступающими процессами.** |
| 14 | RR(2) | SRT | 3÷11 | 7±3 |

**Код программы**

**public class** Proc {  
 **private** String **name**;  
 **private int AppearTime**;  
 **private int PerfTime**;  
 **private int WatingTime**;  
 **private int TurnTime**;  
  
 **public void** IncWaitingTime(){  
 **WatingTime**++;  
 }  
  
 **public** Integer getTurnTime(){  
 **return TurnTime**;  
 }  
  
 **public void** UpdateTurnTime(){  
 **TurnTime**+=**WatingTime**;  
 }  
  
 **public** Integer getWaitTime(){  
 **return WatingTime**;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public int** getAppearTime() {  
 **return AppearTime**;  
 }  
  
 **public int** getPerfTime() {  
 **return PerfTime**;  
 }  
  
 **public void** DecrPerfTime(){  
 **PerfTime**--;  
 }  
  
 **public** Proc(**int** i,**int** a, **int** b){  
 **name**=**"Proc"**+String.*valueOf*(i);  
 **AppearTime** = a;  
 **PerfTime** = b;  
 **TurnTime** = **PerfTime**;  
 }  
}

**public class** Main {  
 **private static int** *n* = 22;  
 **private static int** *limit*=2;  
 **private static** ArrayList<Proc> *rr* = **new** ArrayList<>();  
 **private static** ArrayList<Proc> *srt* = **new** ArrayList<>();  
 **private static** Queue<Proc> *q* = **new** LinkedList<>();  
 **public static void** main(String args[]){  
 *BuildMas*();  
 **for** (Proc p:*srt*)  
 System.***out***.println(p.getName()+**"|"**+p.getAppearTime()+**"|"**+p.getPerfTime());  
 *SRT*();  
 *RR*();  
 **double** RRavrTurn=0, SRTavrTurn=0;  
 **double** RRavrWait=0, SRTavrWait=0;  
 **for**(**int** i=0;i<*n*;i++) {  
 RRavrTurn = *rr*.get(i).getTurnTime() + RRavrTurn;  
 RRavrWait = *rr*.get(i).getWaitTime() + RRavrWait;  
 SRTavrTurn = *srt*.get(i).getTurnTime() + SRTavrTurn;  
 SRTavrWait = *srt*.get(i).getWaitTime() + SRTavrWait;  
 }  
 RRavrWait/=*n*;  
 RRavrTurn/=*n*;  
 SRTavrWait/=*n*;  
 SRTavrTurn/=*n*;  
 System.***out***.println(**"Среднее время оборота RR: "**+RRavrTurn+**" Среднее время ожидания RR: "**+RRavrWait);  
 System.***out***.println(**"Среднее время оборота SRT:"**+SRTavrTurn+**" Среднее время ожидания SRT:"**+SRTavrWait);  
 }  
  
  
 **private static void** SRT(){  
 **int** i=0;  
 Proc next;  
 Proc current=**null**;  
 **int** finished = 0;  
 **for**(**int** timer=0;;timer++){  
 **if**(i<*n*) {  
 next = *srt*.get(i);  
  
 **if** (timer == next.getAppearTime()) {  
 **if** (current == **null**)  
 current = next;  
 **else if** (next.getPerfTime() < current.getPerfTime()) {  
 *q*.add(current);  
 current = next;  
 } **else** *q*.add(next);  
 i++;  
 }  
  
  
 **if** (current == **null**) **continue**;  
 }  
  
 current.DecrPerfTime();  
 *//System.out.println(current.getName());* **for**(Proc p:*q*){  
 p.IncWaitingTime();  
 }  
  
 **if**(current.getPerfTime()==0){  
 finished++;  
 **if** (finished==*n*) **break**;  
 current = *FindMin*();  
 *q*.remove(current);  
 }  
 }  
  
 **for**(Proc p: *q*){  
 p.UpdateTurnTime();  
 }  
 }  
  
 **private static** Proc FindMin(){  
 **if**(*q*.isEmpty()) **return null**;  
 Proc min = **new** Proc(-1,0,12);  
 **for**(Proc p:*q*){  
 **if** (p.getPerfTime()< min.getPerfTime())  
 min = p;  
 }  
 **return** min;  
 }  
  
 **private static void** RR(){  
 **int** num=0;  
 **int** timer=-1;  
 **int** inProg = 0;  
 **int** finished = 0;  
 Proc current=**null**;  
 **while**(**true**) {  
 timer++;  
  
 **if**(num<*n* && timer==*rr*.get(num).getAppearTime())  
 *q*.add(*rr*.get(num++));  
  
 **if**(inProg==0)  
 current = *q*.poll();  
  
 **if**(current==**null**) {  
 *//System.out.println("жду");* **continue**;  
 }  
  
 **if**(inProg!=*limit*) {  
 *//System.out.println(current.getName());* current.DecrPerfTime();  
 inProg++;  
 }  
  
 **for**(Proc p:*q*){  
 p.IncWaitingTime();  
 }  
  
 **if** (current.getPerfTime() == 0){  
 inProg=0;  
 current=**null**;  
 **if**(++finished==*n*) **break**;  
 }  
  
 **if**(inProg==*limit*) {  
 inProg=0;  
 *q*.add(current);  
 current=**null**;  
 }  
 }  
  
 **for**(Proc p: *q*){  
 p.UpdateTurnTime();  
 }  
 }  
  
 **private static void** BuildMas(){  
 …  
 }  
}

**Пример результата запуска программы**

Количество процессов 25

Время появления процессов

0 7 11 16 24 30 34 35 40 46 54 60 68 74 80 88 98 108 114 123 128 133 141

Длительность процессов

9 8 11 9 4 10 8 11 7 7 3 9 4 11 9 8 9 5 7 9 5 10

SRT

Среднее время ожидания 13.045

Среднее время оборота 20.9

Максимальная длина очереди 2

RR(2)

Среднее время ожидания 28.09

Среднее время оборота 35.954

Максимальная длина очереди 7

**Вывод**

Время ожидания для алгоритма round-robin с квантом, равным 2 единице времени выполнения процесса, оказалось в среднем больше, чем для алгоритма SRT. Алгоритм SRT, учитывающий время исполнения процесса, в данном случае более эффективен.