Министерство образования и науки Украины
Севастопольский государственный технический университет

Методические указания

к лабораторной работе №3 по курсу
“Параллельные и распределенные вычисления” на тему
«Решение задачи «читатели и писатели»
для студентов направления
“Компьютерная инженерия ”

Севастополь
2009

Цель работы:

Исследовать возможности, предоставляемые системой программирования JAVA для решения задачи синхронизации «читатели и писатели».

Задание на лабораторную работу

1. Разработать в соответствии с вариантом задания программу, создающую подпроцессы двух видов, осуществляющих обмен данными
2. Исследовать поведение программы при использовании различных методов синхронизации (событийные переменные, семафоры)
3. Проанализировать полученные результаты.

Методика выполнения работы

Задача “читатели и писатели” относится к базовым задачам синхронизации. В общем (простейшем) случае взаимодействуют процессы двух видов. Процессы-читатели считывают информацию из общей области памяти, процессы-писатели модифицируют информацию в общей области памяти.
Для корректного решения задачи “читатели и писатели” необходимо обеспечить режим взаимного исключения по отношению к общей области для процессов-писателей, как относительно читателей, так и относительно других писателей, в то же время допускается одновременная работа нескольких процессов-читателей.

Рассмотрим пример программы, реализующий простейший вариант задачи “читатели и писатели”.

Основной поток создает несколько подпроцессов каждого вида, которые будут выполняться в течение 5-ти секунд. Процесс Writer модифицирует общую переменную; процес Reader – считывает общую переменную без ее модификации.

```
class PC
{
  public static void main(String args[]) {
    Q q = new Q();
    new Writer(q,1);
    new Writer(q,2);
    new Reader(q,1);
    new Reader(q,2);
    new Reader(q,3);
    new Reader(q,4);
    try {Thread.sleep(5000);}     catch(InterruptedException e) {} ;
    q.work = false;
  }
}
```

```
class Q
{
  int n;
  boolean work = true;
  int read()
  {
    Thread t = Thread.currentThread();
    System.out.println(t.getName()+" try...");
    System.out.println(t.getName()+" reading...");
    try { Thread.sleep(50); } catch (InterruptedException e) {}
    System.out.println(t.getName()+":"+n);
    return n;
  }
  void write(int n)
  {
    Thread t = Thread.currentThread();
    System.out.println(t.getName()+" try...");
    System.out.println(t.getName()+" writing...");
    try { Thread.sleep(100); } catch (InterruptedException e) {}
    this.n = n;
    System.out.println(t.getName()+":"+n);
  }
}
class Writer implements Runnable
{
  Q q;
  Writer(Q q,int num)
  {
    this.q = q;
    new Thread(this, "Writer "+num).start();
  }
  public void run()
  {
    int i = 0;
    while (q.work) { q.write(i++); }
  }
}
class Reader implements Runnable
{
  Q q;
  Reader(Q q, int num)
  {
    this.q = q;
    new Thread(this, "Reader "+num).start();
  }
  public void run()
  {
    while (q.work) { q.read(); }
  }
}
```

Анализ полученных результатов позволяет сделать вывод о некорректном решении поставленной задачи.

Разработаем дополнительные процедуры, обеспечивающие проверку возможности продолжения работы для процесса каждого вида (StartRead и StartWrite, соответственно, для читателей и писателей).
Процесс-читатель может продолжать работу в том случае, если нет работающих процессов-писателей, независимо от количества работающих процессов-читателей.
Процесс-писатель может продолжать работу, если нет работающих процессов, как писателей, так и читателей.
Будем использовать режим пассивного ожидания (процесс, которые не может продолжать работу, будет помещаться в очередь). Соответственно, потребуются средства, освобождающие ожидающие процессы. Установим, что процесс при завершении работы с общим ресурсом должен будет выполнить эти действия (процедуры StopRead и StopWrite).
При разработке вышеуказанных процедур необходимо определить, каким процессам будет отдаваться предпочтение (какой процесс из очереди, читатель или писатель, в первую очередь получит управление).

В качестве базового механизма используем событийные переменные из пакета java.util.concurrent.locks (описания помещаем в класс Q):

```
final Lock lock = new ReentrantLock();
final Condition ReaderCanStart = lock.newCondition();
final Condition WriterCanStart = lock.newCondition();
```

Четыре служебные переменные будут содержать количество читателей в очереди, количество активных читателей, количество писателей в очереди и количество активныых писателей:

```
int WaitingReader = 0;
int RunningReader = 0;
int WaitingWriter = 0;
int RunningWriter = 0;
```

Процедура запуска процесса чтения:

```
void StartRead()   {
    lock.lock();
    try {
    if ((RunningWriter>0) | (WaitingWriter>0)) {
      WaitingReader++;
      try { ReaderCanStart.await(); } catch (InterruptedException e) {}
    } else {
      RunningReader++;
    }
   } finally { lock.unlock(); }
 }
```

Отметим, что очередной читатель продолжит работу, если нет работающих и ожидающих писателей.

Процедура завершения процесса чтения:

```
void StopRead() {
  lock.lock();
  try {
  RunningReader--;
  if ((RunningReader==0) & (WaitingWriter>0)) {
    WaitingWriter--;
    RunningWriter++;
    WriterCanStart.signal();
   }
  } finally {
     lock.unlock();
 }
}
```

Процедура освобождает ожидающего писателя, если все читатели неактивны.

Процедура запуска процесса записи:

```
void StartWrite()
{
  lock.lock();
  try {
  if ((RunningWriter>0) | (RunningReader>0)) {
    WaitingWriter++;
    try { WriterCanStart.await(); } catch (InterruptedException e) {}
  } else RunningWriter++;
 }
  finally {
    lock.unlock();
  }
}
```

Очередной писатель продолжит работу, если нет работающих писателей и работающих читателей.

Процедура завершения процесса записи:

```
  void StopWrite()
  {
    lock.lock();
    try {
    RunningWriter--;
    if (WaitingReader>0) {
      while (WaitingReader>0) {
        RunningReader++;
        WaitingReader--;
        ReaderCanStart.signal();
      }
    } else
    { if (WaitingWriter>0) {
        WaitingWriter--;
        RunningWriter++;
        WriterCanStart.signal();
      }
    }
    } finally {
      lock.unlock();
    }
  }
}
```

Процедура в первую очередь освобождает всех ожидающих читателей. Если таких нет, освобождается очередной ожидающий писатель.

Анализ результатов подтверждает, что получено правильное решение.

В индивидуальном задании требуется синхронизировать работу нескольких (минимум 4-х) писателей и, как минимум, двух читателей. Каждый писатель помещает сформированное им значение в свою ячейку общей области. Эти действия выполняются несколько раз в секунду. Каждый читатель, также несколько раз в секунду, обрабатывает сформированные к этому моменту значения. Необходимо проверить работоспособность программы при различных интенсивностях потоков запросов на чтение и запись.

Варианты заданий

№	Тип процесса - писателя	Тип процесса - читателя	Приоритет (Ч-читатели, П-писатели)
1	1	3	П
2	1	3	Ч
3	1	4	П
4	1	4	Ч
5	2	3	П
6	2	3	Ч
7	2	4	П
8	2	4	Ч
9	1	5	П
10	1	5	Ч
11	1	6	П
12	1	6	Ч
13	2	5	П
14	2	5	Ч
15	2	6	П
16	2	6	Ч

Типы процессов:

1. Генерация ряда натуральных чисел
2. Генерация последовательности целых случайных чисел
3. Среднее арифметическое значение
4. Минимальное значение
5. Максимальное значение
6. «Проголосованное» (самое часто встречающееся) значение

Содержание отчета

1. Постановка задачи
2. Текст программы с комментариями
3. Протоколы испытаний
4. Выводы

Контрольные вопросы

1. Средства синхронизации процессов в пакетах java.