Министерство образования и науки Украины
Севастопольский государственный технический университет

Методические указания

к лабораторной работе №4 по курсу
“Параллельные и распределенные вычисления” на тему
«Организация конвейерных вычислений»
для студентов направления
“Компьютерная инженерия ”

Севастополь
2009

Цель работы:

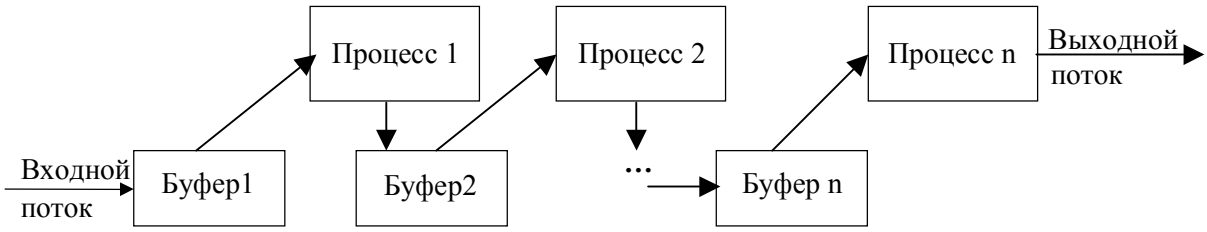
Исследовать методы и средства организации конвейерной обработки данных

Задание на лабораторную работу

1. Разработать в соответствии с вариантом задания программу, выполняющую конвейерную (поэтапную) обработку массива данных
2. Исследовать поведение программы при использовании различных методов синхронизации (семафоры, событийные переменные)
3. Проанализировать полученные результаты.

Методика выполнения работы

Конвейерная обработка данных применяется при решении сложных вычислительных задач, допускающих разбиение процесса решения на относительно независимые части. В этом случае выходные данные (результаты работы) очередного процесса являются входными данными следующего процесса и т.д. Соответственно, за счет параллельной обработки сокращается общее время решения задачи. Типовая схема конвейерной обработки выглядит следующим образом:



Рассмотрим пример программы, реализующий последовательную обработку данных. Основной процесс создает шесть подпроцессов, первый из которых (Generator) формирует очередную порцию данных и передает их на обработку второму подпроцессу (Worker 1). Второй подпроцесс выполняет какие-то действия над этими данными и передает результат на обработку следующему подпроцессу (Worker 2). Далее запускается подпроцесс Worker 3 и т.д. Можно сделать вывод, что процессы попарно взаимодействуют между собой, как “производитель” и “потребитель”.

Для синхронизации подпроцессов по схеме “производитель-потребитель” введем в программу шесть семафоров (s0..s5). Семафор s0 изначально открыт, что дает возможность начать работу первому подпроцессу. Процесс Generator, сформировав очередную порцию данных, открывает семафор s1, что дает возможность стартовать подпроцессу Worker 1. На последнем цикле обработки подпроцесс Worker 5 открывает семафор s0, тем самым повторно запускает процесс Generator.

Рассмотрим программу, реализующую описанную схему взаимодействия подпроцессов. Для простоты картины собственно обработку данных оставим в стороне. Считаем, что процесс Generator очередную порцию данных записывает в буфер 1, процесс Worker 1 выбирает данные из буфера 1 и записывает результат их обработки в буфер 2 и т.д.

```
import java.util.concurrent.*;
class PC {
  public static void main(String args[])
  {
    Q q = new Q();
    new Generator(q,0);
    new Worker(q,1);
    new Worker(q,2);
    new Worker(q,3);
    new Worker(q,4);
    new Worker(q,5);
    try {Thread.sleep(5000);}     catch(InterruptedException e) {} ;
    q.work = false;
  }
}
class Q
{
  int n;
  boolean work = true;
  Semaphore s0 = new Semaphore(1,true);
  Semaphore s1 = new Semaphore(0,true);
  Semaphore s2 = new Semaphore(0,true);
  Semaphore s3 = new Semaphore(0,true);
  Semaphore s4 = new Semaphore(0,true);
  Semaphore s5 = new Semaphore(0,true);
}
class Generator implements Runnable
{
  Q q;
  int num;
  Generator(Q q,int num)
  {
    this.q = q;
    this.num = num;
    new Thread(this, "Generator "+num).start();
  }
  public void run()
  {
    Thread t = Thread.currentThread();
    while (q.work)
    {
      try { q.s0.acquire(); } catch(InterruptedException e) {} ;
      System.out.println(t.getName()+" start");
      try { Thread.sleep(100); } catch(InterruptedException e) {} ;
```

```
      System.out.println(t.getName()+" stop");
      q.s1.release();
    }
  }
}

class Worker implements Runnable
{
  Q q;
  int num;
  Worker(Q q,int num)
  {
    this.q = q;
    this.num = num;
    new Thread(this, "Worker "+num).start();
  }

  public void run()
  {
    Thread t = Thread.currentThread();
    while (q.work)
    {
      switch (num)
      {
        case 1: try { q.s1.acquire(); } catch(InterruptedException e) {} ; break;
        case 2: try { q.s2.acquire(); } catch(InterruptedException e) {} ; break;
        case 3: try { q.s3.acquire(); } catch(InterruptedException e) {} ; break;
        case 4: try { q.s4.acquire(); } catch(InterruptedException e) {} ; break;
        case 5: try { q.s5.acquire(); } catch(InterruptedException e) {} ; break;
      }
      System.out.println(t.getName()+" start");

// здесь должны быть операторы обработки данных
      try { Thread.sleep(100); } catch(InterruptedException e) {} ;

      System.out.println(t.getName()+" stop");

      switch (num)
      {
        case 1: q.s2.release(); break;
        case 2: q.s3.release(); break;
        case 3: q.s4.release(); break;
        case 4: q.s5.release(); break;
        case 5: q.s0.release(); break;
      }
    }
  }
}
```

Проверим правильность решения задачи синхронизации. Из протокола работы программы видно, что процессы работают строго поочередно. Очевидно, что для повышения скорости работы целесообразно обеспечить параллельную работу подпроцессов, непосредственно не взаимодействующих друг с другом. Например, процессы Generator, Worker 2 и Worker 4 или Worker 1, Worker 3 и Worker 5.
Определим условие продолжения работы для каждого из процессов.
Процессу Generator необходим свободный буфер 1.
Процессу Worker 1 необходим заполненный буфер 1 и свободный буфер 2.
Процессу Worker 2 необходим заполненный буфер 2 и свободный буфер 3.
Процессу Worker 3 необходим заполненный буфер 3 и свободный буфер 4.
Процессу Worker 4 необходим заполненный буфер 4 и свободный буфер 5.
Процессу Worker 5 необходим заполненный буфер 5.
В такой интерпретации задача приближается к другой классической – “Обедающие философы”. Для реализации синхронизирущих правил воспользуемся парами семафоров.

```
                                                Semaphore s02 = new Semaphore(1,true);
Semaphore s11 = new Semaphore(0,true); Semaphore s12 = new Semaphore(1,true);
Semaphore s21 = new Semaphore(0,true); Semaphore s22 = new Semaphore(1,true);
Semaphore s31 = new Semaphore(0,true); Semaphore s32 = new Semaphore(1,true);
Semaphore s41 = new Semaphore(0,true); Semaphore s42 = new Semaphore(1,true);
Semaphore s51 = new Semaphore(0,true);
```

Открытый семафор с четным номером сигнализирует о том, что соответствующий буфер свободен для записи информации, открытый семафор с нечетным номером – соответствующий буфер заполнен и доступен для выборки.

Перепишем метод run для процесса типа Worker:

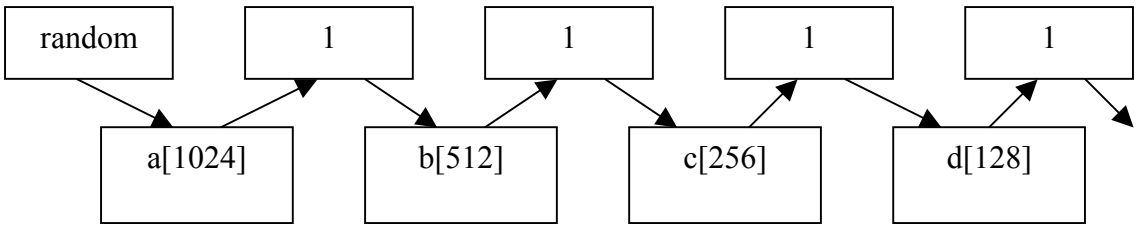
```
public void run()
  {
   Thread t = Thread.currentThread();
   while (q.work)
   {
    switch (num)
    {
     case 1: try { q.s11.acquire(); } catch(InterruptedException e) {} ;
             try { q.s12.acquire(); } catch(InterruptedException e) {} ; break;
     case 2: try { q.s21.acquire(); } catch(InterruptedException e) {} ;
             try { q.s22.acquire(); } catch(InterruptedException e) {} ; break;
     case 3: try { q.s31.acquire(); } catch(InterruptedException e) {} ;
             try { q.s32.acquire(); } catch(InterruptedException e) {} ; break;
     case 4: try { q.s41.acquire(); } catch(InterruptedException e) {} ;
             try { q.s42.acquire(); } catch(InterruptedException e) {} ; break;
     case 5: try { q.s51.acquire(); } catch(InterruptedException e) {} ;
    }
```

```
    System.out.println(t.getName()+" start");

// здесь должны быть операторы обработки данных
    try { Thread.sleep(10); } catch(InterruptedException e) {} ;

    System.out.println(t.getName()+" stop");
    switch (num)
    {
     case 1: q.s21.release(); q.s02.release(); break;
     case 2: q.s31.release(); q.s12.release(); break;
     case 3: q.s41.release(); q.s22.release(); break;
     case 4: q.s51.release(); q.s32.release(); break;
     case 5:                  q.s42.release();
    }
   }
  }
```

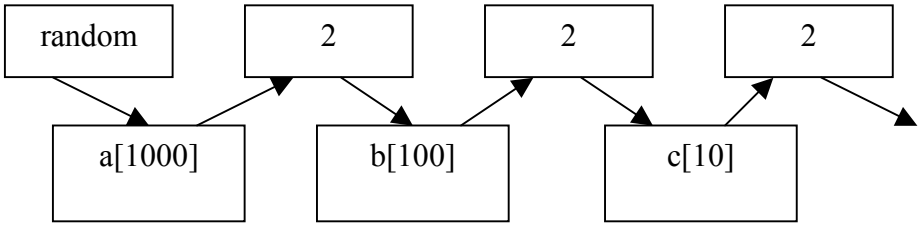
Варианты заданий

N	Схема	Функция - обработчик	Механизм
1	1	Свертка элементов массива: B[j]=(A[i]+A[i+1])/2 (1)	Семафор
2	1	Свертка элементов массива: B[j]=max(A[i],A[i+1]) (1)	Семафор
3	2	Свертка элементов массива: B[j]=(A[i]+…+A[i+9])/10 (2)	Семафор
4	2	Свертка элементов массива: B[j]=min(A[i]…A[i+9]) (2)	Семафор
5	3	Усреднение элементов последовательностей (3)	Семафор
6	3	Выбор максимума из пар элементов (3)	Семафор
7	1	Свертка элементов массива: B[j]=(A[i]+A[i+1])/2 (1)	События
8	1	Свертка элементов массива: B[j]=max(A[i],A[i+1]) (1)	События
9	2	Свертка элементов массива: B[j]=(A[i]+…+A[i+9])/10 (2)	События
10	2	Свертка элементов массива: B[j]=min(A[i]…A[i+9]) (2)	События
11	3	Усреднение элементов последовательностей (3)	События
12	3	Выбор максимума из пар элементов (3)	События

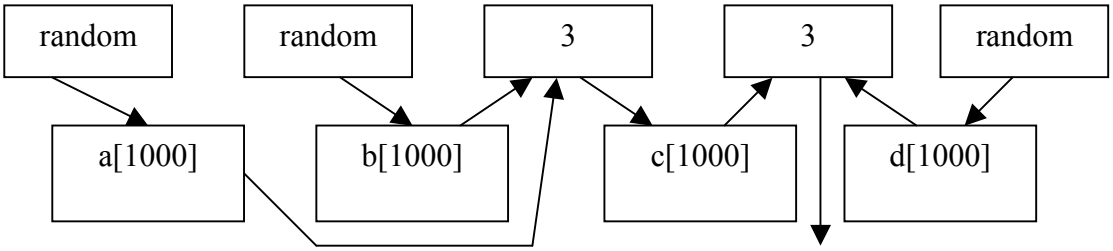
1



2



3



Содержание отчета

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