Министерство образования и науки Украины
Севастопольский государственный технический университет

Методические указания

к лабораторной работе №1 по курсу
“Параллельные и распределенные вычисления” на тему
«Исследование средств организации подпроцессов»
для студентов направления
“Компьютерная инженерия ”

Севастополь
2009

Цель работы:

Исследовать возможности, предоставляемые системой программирования JAVA для организации параллельно исполняющихся ветвей программы.

Задание на лабораторную работу

1. Разработать в соответствии с вариантом задания программу, создающую два независимых подпроцесса.
2. Исследовать поведение программы при различных значениях приоритетов подпроцессов, влияние временных задержек и средств явной передачи управления.
3. Проанализировать полученные результаты.

Методика выполнения работы

Для организации параллельного выполнения процессов в однопроцессорных системах применяется метод разделения времени. Другими словами, процессор поочередно обслуживает готовые к исполнению задачи, создавая тем самым виртуальную многопроцессорную систему.
В системе программирования Java используется дисциплина планирования работы процессора на основе приоритетов. Это означает, что в первую очередь должен исполняться подпроцесс с большим приоритетом. Как выполняются потоки с одинаковым приоритетом, в спецификации Java не определено. Кроме этого, необходимо учитывать особенности планировщика ОС. Порядок их выполнения определяется операционной системой.
Например, в реализации Java для Windows используется режим квантования времени (каждому подпроцессу выделяется определенный промежуток времени. В других реализациях (например, Solaris) квантование времени не применяется. Соответственно, в первом случае можно надеяться на «равноправное» обслуживание равноприоритетных подпроцессов, во втором случае – равноприоритетные подпроцессы будут выполняться последовательно.
В любом случае, следует учитывать, что переключение процессора между подпроцессами происходит в результате прерывания, которое по отношению к текущему процессу может быть как внешним, так и внутренним (например, выполнение операции ввода-вывода).

Для организации подпроцессов в системе программирования JAVA используется класс Thread, инкапсулирующий необходимые средства.
При создании нового объекта Thread нужно указать, какой программный код должен выполняться. Соответственно, потребуется определить метод run.

Для исследования работы планировщика подпроцессов составим простую программу, в которой будет создано два подпроцесса. Для анализа результатов работы в методе run предусмотрим вывод сообщения на каждой итерации.

```
class ThreadTest implements Runnable
{
 ThreadTest()
 {
  Thread t1 = new Thread(this, "Thread 1");
  System.out.println("Thread created: " + t1);
  t1.start();
  Thread t2 = new Thread(this, "Thread 2");
  System.out.println("Thread created: " + t2);
  t2.start();
  try
  {
   t1.join();
   t2.join();
  }
  catch (InterruptedException e)
  {
   System.out.println("interrupted");
  }
 System.out.println("exiting main thread");
 }



 public void run()
 {
   Thread t = Thread.currentThread();

    for (int i = 1; i <= 1000; i++)
    {
     System.out.println(t.getName()+" " + i);
    }
  System.out.println("exiting "+t.getName());
 }
 public static void main(String args[])
 {
  new ThreadTest();
 }
}
```

Проверим, каким образом происходит выделение процессорного времени каждому подпроцессу. Для этого запустим программу 2-3 раза, сохраняя выходные данные. Сравним полученные результаты.

Изменим программу, увеличив приоритет второго подпроцесса:

```
…
Thread t2 = new Thread(this, "Thread 2");
t2.setPriority(Thread.NORM_PRIORITY+2);
System.out.println("Thread created: " + t2);
t2.start();
…
```

Проанализируем полученные результаты.

Добавим временную приостановку процесса в метод run.

```
public void run()
  {
    Thread t = Thread.currentThread();
    for (int i = 1; i <= 1000; i++)
      {
        System.out.println(t.getName()+" " + i);
        try
        {
          t.sleep(100);
        }

        catch (InterruptedException e)
        {
          System.out.println("interrupted");
        }
      }
    System.out.println("exiting "+t.getName());
  }
```

Проанализируем результаты при равных и неравных приоритетах процессов.

Модифицируем метод run так, чтобы подпроцесс на каждой итерации явно освобождал процессор:

```
public void run()
{
   Thread t = Thread.currentThread();
   for (int i = 1; i <= 1000; i++)
     {
        System.out.println(t.getName()+" " + i);
        t.yield();
     }
   System.out.println("exiting "+t.getName());
}
```

Также проанализируем результаты при равных и неравных приоритетах процессов.

Варианты заданий

№	Тип первого процесса	Тип второго процесса
1	1	2
2	1	3
3	1	4
4	1	5
5	1	6
6	2	3
7	2	4
8	2	5
9	2	6
10	3	4
11	3	5
12	3	6
13	4	5
14	4	6
15	5	6

Типы процессов:

1. Вычисление суммы ряда натуральных чисел
2. Вычисление среднего арифметического значения последовательности случайных чисел
3. Вычисление среднего арифметического значения ряда натуральных чисел
4. Вычисление чисел Фибоначчи
5. Вычисление среднего арифметического значения ряда Фибоначчи
6. Вычисление факториала заданного числа

Содержание отчета

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

Контрольные вопросы

1. Основные методы класса Thread
2. Возможности системы управления подпроцессами