Parallel Programming WS 2014/2015 - Assignment 6

Kastens, Pfahler
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn
Jan 19, 2015

Exercise 1 (Asynchronous Message Passing, Filter)

We want to build a system of collaborating processes that use asynchronous messages to find prime numbers. There are three different kinds of processes implemented as Threads:

- a `PrimeGenerator` process

```
class PrimeGenerator extends Thread {
    private Random rand = new Random(System.currentTimeMillis());

    private int getRandomNumber() {
        return Math.abs(rand.nextInt()) % 1000 + 2;
    }

    public void run() {
    }
}
```

- multiple `PrimeChecker` processes,

```
class PrimeChecker extends Thread {

    private boolean isPrime(int n) {
        if (n % 2 == 0) {
            return false;
        }   // exclude even numbers;
        int div = 3;

        while (div * div <= n) {
            if (n % div == 0) {
                return false;
            }
            div += 2;
        }
        return true;
    }

    public void run() {
    }
}
```

- and a single `PrimeReporter` process

```
class PrimeReporter extends Thread {
    private Set<Integer> oldNumbers = new HashSet<Integer>();

    private void reportNewNumber(Integer num) {
        if (oldNumbers.add(num)) {
            System.out.println("PrimeReporter reports " + num);
        }
    }

    public void run() {
    }
}
```

A `PrimeGenerator` process generates random positive integers that may or may not be prime numbers. Every `PrimeChecker` process continuously retrieves a number and checks it. Only prime numbers are passed on, other numbers are silently discarded. The `PrimeReporter` process accepts numbers that have been identified as prime numbers and outputs them. To prevent duplicate numbers in the output, a history of previously output prime numbers is kept.

Because checking prime numbers is the slowest part of this task, we will use four `PrimeChecker` processes which operate in parallel and share the work load. Generating and reporting numbers will use one process each.

a) Design the communication structure for these processes. The necessary communication channels should belong to processes. They should be accessible from other processes by get methods.

b) LAB: Directory `blatt6/PrimeFilter` contains a framework for the implementation of this idea. `Channel.java` contains the class `Channel` as shown on Slide 61. Complete the process classes in file `PrimeTest.java` and implement a suitable main method.

Exercise 2 (Asynchronous Message Passing: Resource Server)

LAB: Directory `blatt6/LicenseServer` contains an implementation of a client/server application for a license server. Clients request software licenses ("`REQUEST`") from this server. There is limited number of such licenses. The server grants these licenses when available, otherwise the client requests are queued. When a client gets a license it uses the software for some time and then returns the license ("`RELEASE`").

a) Visualize the communication structure of the license server simulation.
b) Compile the supplied Java classes and execute the resulting program. Explain the observed behavior of the program.
c) Design a new communication structure such that the server serves two separate channels, one channel for each kind of message (`REQUEST` and `RELEASE`). Implement your approach. Take care to avoid unnecessary blocking of the server.

Exercise 3 (LAB/HOMEWORK: Broadcast in a net of processors)

Directory `blatt6/Broadcast` contains a Java simulation of the broadcast method, cf. Slide 72. Add code to output the edges of a spanning tree of the network graph. Try different initiator nodes.