Lab Session 3, Concurrency and Parallelism

Jean-Viven Millo
INRIA Sophia-Antipolis
jean-vivien.millo@inria.fr

Andrea Tettamanzi
UNSA
andrea.tettamanzi@unice.fr

November 6, 2012

Abstract

The goal of this session is to discover the basic mechanisms of semaphores and priorities. In order to realize the exercises, you will require the following documents: i) this lab session subject ii) the course iii) the JAVADOC.

Exercise 1: Semaphore

Taken from *http://en.wikipedia.org/wiki/Dining_philosophers_problem*
Five silent philosophers sit at a table around a bowl of spaghetti. A fork is placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when he has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it's not being used by another philosopher. After he finishes eating, he needs to put down both the forks so they become available to others. A philosopher can grab the fork on his right or the one on his left as they become available, but can't start eating before getting both of them.

Eating is not limited by the amount of spaghetti left: assume an infinite supply. An alternative problem formulation uses rice and chopsticks instead of spaghetti and forks.

The problem is how to design a discipline of behavior (a concurrent algorithm) such that each philosopher won't starve, i.e. can forever continue to alternate between eating and thinking, assuming that any philosopher can not know when others may want to eat or think.
1/Define the *fork* to be a empty class (We only need to instantiate it).
2/Implements the dining philosophers problem in a way that avoid deadlock. The accesses to the forks have to be *synchronized*.
3/Change the implementation in order to use the class **java.util.concurrent.Semaphore**.

Exercise 2: Readers writers with priority

Let us consider a table of 1000 bytes that will simulate the behavior of a memory. The access to this memory are the following:

1. There are three writers that are reading data from text files and writing them in memory
2. There are three readers that read data in the memory and write them in (other) text files.
3. The writers have priority over the readers. Of course, when the memory is full, writers cannot write anymore.
4. The three readers as well as the three writers have there own hierarchy of priorities.
5. The three readers as well as the three writers alternate between sleeping for a random time and writing 100 bytes in the memory.

1/ Implement the system as described
2/ Let the readers and writers work on a random size (≤ 1000) packet of data instead of 100.
3/ Consider the following extra requirement: Reader i reads only data from Writer i (with $i \in \{1, 2, 3\}$).