Università degli Studi di Pisa

Laurea Magistrale in Informatica

Advanced Programming

Middle Term Paper

Start Date: 13/11/2013
Submission deadline: 20/12/2013 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone's code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary. **Do not include in an exercise code only needed for a later exercise**.
4. cite references to literature or Web pages from where information was taken.

Exercise 1

Consider a set of tasks among which there is a partial ordering expressing the constraint that a task must precede another. Define a set of classes to represent such tasks and their precedencies (without storing information outside the task objects).

Implement an iterator that lists all possible total orderings that satisfy the precedencies among tasks. An iterator is a class providing an interface like this:

```
interface Iterator<T> {
    bool  HasNext();
    T Next();
}
```

Exercise 2

Extend the previous classes, in order to represent the duration of a task. Implement a method that finds an assignment of tasks to *n* processors which minimizes the overall execution time.

Exercise 3

Extend the previous methods and/or classes so that the iterator will return no solution if there are cycles in the dependencies.

Exercise 4

Discuss and propose how to implement (as pseudocode) a generic iterator class using threads that provides a `yield(x)` method for returning `x` as a result of a call to `Next()`.

Exercise 5

Illustrate the constructs of threads in a programming language of your choice. What is the relation between a `yield` method on threads and the `yield` operator on iterators?