Programação Paralela
Assignment 2

Departamento de Informática
Universidade de Évora

2025/2026

The goal of this assignment is to implement a termination detection algorithm for a distributed basic algorithm.

1. You are given code which implements a trivial distributed algorithm. The behaviour of this algorithm is as follows:

- Each process running the algorithm is either ACTIVE or PASSIVE.
- Initially, every process is PASSIVE, except for the initiator, which is ACTIVE.
- A PASSIVE process waits to be sent a message; when it receives one, it becomes ACTIVE.
- While a process is ACTIVE, it:
  - Sends a message to some other random process with a given probability.
  - Becomes PASSIVE with a (different) given probability.
  - Receives messages sent from the other processes.

The messages contain a single sequential integer value.

2. The algorithm is *terminated* when all processes are PASSIVE and all messages sent have been received.

3. Your task is to implement a (centralised) termination detection algorithm, that runs in parallel with the basic algorithm (in a different thread of the same process) and which detects the termination of the basic algorithm.

4. The code provided includes the file `control-skeleton.c`, which contains the skeleton of the implementation of the termination detection algorithm.

That file shows the structure of the implementation of the termination detection algorithm, which consists of the following functions:

`detect_termination`

> The main loop of the termination detection algorithm. When this function returns, it means that the termination of the basic algorithm has been detected.

The remaining functions are *hooks*, called from the code of the basic algorithm, to notify the termination detection algorithm of events that are relevant to it:

`control_basic_start_hook`

> Called when the basic algorithm is starting up; allows initial synchronisation between the basic algorithm and the termination detection algorithm.

`control_become_passive_hook`

> Called when the process becomes PASSIVE.

`control_become_active_hook`

Called when the process becomes ACTIVE.

`control_basic_send_hook`

Called when the process sends a (basic) message to another process.

`control_basic_receive_hook`

Called when the process receives a (basic) message from another process.

5. You may define any other function you deem necessary.

6. Apart from the `control-skeleton.c` file, it should not be necessary to change any of the code provided.

However, if you find that the basic algorithm is taking too long to terminate, or is terminating too fast, you may wish to change the value of the `SEND_BASIC_MESSAGE_PROBABILITY` and `BECOME_PASSIVE_PROBABILITY` constants, in file `basic.c`.

7. To compile and run the program, you need to install an implementation of the MPI standard. Open MPI is recommended.

It is also recommended to have the `make` utility installed.

8. To compile the code provided, you must unpack the archive provided into some directory (folder) on your file system and then run the command `make` in that directory.

9. The compiled program is run with the following command, where n stands for the number of processes that will execute the algorithm ($n \geq 2$):

`mpirun -np` n `./termination`

10. As provided, the code runs the basic algorithm for 10 seconds, and then acts as if termination has been detected (the `detect_termination` function returns).

11. When run with the `--quiet` option, debugging messages, produced by the `trace` function, are not displayed.

The `-r` option causes the re-seeding of the random number generator.

12. The assignment may be completed individually or by groups of two students. In the latter case, the group has the choice to either implement two different centralised algorithms or to implement one decentralised termination detection algorithm.

13. Examples of centralised termination detection algorithms are the Dijkstra-Scholten and Safra's algorithms. Examples of decentralised termination detection algorithms are the Shavit-Francez and Rana's algorithms.

14. To complete the assignment, you must submit the code of your program(s) and a report containing:

- A description of the algorithm(s) implemented.
- The analysis of the behaviour of the algorithm(s), including the problems encountered during its(their) implementation and the solutions found for those problems, or suggestions of possible solutions.
- Recipes to compile and run your program(s). (You may opt to adapt and use the provided Makefile.)

15. The assignment should be submitted, via `moodle`, by **June 30, 2026**.