Concurrent and Distributed Systems

Practical 1 – Processes communicating via Files

This lab contains a checkpoint at the end. Please contact any lecturing staff when you reach that point to receive your credit.

It is important that you use the lab session to ask staff any questions you might have on the material. There are no tutorials! Please do not sit and be stuck!

This lab is concerned with experimenting with communicating processes and understanding the implications, and it is based on the Ornamental Garden Problem from Lab 0.

From Lab 0 you should have an implementation of the problem. It is recommended that you finish Lab 0 before you attempt Lab 1.

Please, **get the Lab 0 checkpoint before proceeding**: show the working program that you have developed for Lab 0 and explain how it works and why the gate counter does not count properly people entering in the garden.

For Lab 1, a fresh implementation of the Ornamental Garden Problem is available on the canvas page of CSCU9V5. This implementation contains two classes: the garden_gate_problem class and the gate class. The former only serves to perform some initialisation tasks and to create an instance of the gate class. The gate class symbolises the garden gates, or rather the counters at the gates.

Remember that the program takes one command line parameter signalling which of the two gates it implements (‘gate_bottom’ or ‘gate_top’). Thus the program needs to be started twice (in Eclipse, using configurations) to implement both gates. The bottom gate needs to be started first as this triggers some general initialisation.

The program simulates 50 people entering the garden by each of the two gates. Thus after the program finishes 100 people should be in the garden. We do not cover the case of some people leaving in the meantime – the garden offers many attractions! Both gates count the number of people passing through and write the result to a file. A gate reads the value from the file, increments it, and writes it back to the file. So, by reading the file, the total number of people currently in the garden is known.

Even though Java offers threads as means to implement concurrency, this first lab concentrates on **processes**. The program has intentionally been designed **not** to make use of threads. It needs to be run twice to get two completely separate processes. Threads will be dealt with in later labs.

Task 0: If necessary, complete Lab 0 and demonstrate it to a demonstrator, and get the checkpoint for Lab 0.

Task 1: Back to the results of the program form Lab 0. Try running the program as the lower gate and as the upper gate sequentially (people only passing through at one gate at a time) and concurrently (people passing through both gates). Why the total is not always 100? You should be able to explain the results in both cases. Try to play with the delay times: How do the relative speeds of the two gates affect the results? - try to identify one or two interesting examples.

Task 2: Design a solution to this problem. The solution should **not** employ any synchronisation mechanism between the two processes. We will cover these options later in the course. Try and redesign the solution, so the problem does not occur in the first place. The solution should be clear if you fully understand the problem encountered in Task 1.

Checkpoint: explain the developed solution and your reflection/examples on timings.