UNIVERSITY of STIRLING

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 this 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. The experiments are based around the 'Ornamental Garden Problem' known 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. As backup, an implementation of the Ornamental Garden Problem will be made available on request (this will prevent you from getting awarded Checkpoint 0).

The program 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.)

A brief reminder on the problem and the implementation:

To run, the program takes one command line parameter signalling which of the two gates is represented by it (‘gate_bottom’ or ‘gate_top’). Thus the program needs to be started twice to implement both gates. The bottom gate needs to be started first as this triggers initialisation of the file.

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. In practise, 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 in the garden is known.

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

Task 1: What results do you get running the program? 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 then **concurrently** (people passing through both gates). Is the total always 100? Why not? Try to understand the reason for the results! You will need to look at the source code. Think about the print queue example from the lecture.

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.