CS4103: Distributed Systems

Assignment: P1 - Programming practical 1 - A distributed system for the prisoner dilemma

Deadline: 5th March 2019

Weighting: 30% of coursework

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

---

1 Objective

This practical is aimed at designing and implementing a simple distributed system with the help of existing communication middleware.

2 Competencies

- Develop experience in implementation of distributed systems
- Develop and consolidate knowledge of distributed system communication
- Develop familiarity with existing high level communication middleware

3 Practical Requirements

This is a programming practical in which you are required to produce a program and an associated report. Your implementation must compile and run on the School lab Machines and it should be possible to run your program from the console/terminal without importing it into an IDE. This document includes a general overview and a set of incremental steps. Please ensure that you complete fully one step before moving to the next step. This practical refers in particular to topics covered in week 2 and 3.

3.1 Introduction

The n-person prisoner's dilemma is a well-known example of a game representing a social dilemma involving two or more participants. Each participant represents a prisoner in a cell with no communication with other prisoners. They are all suspect of a crime and the prosecutor needs to establish the sentence for each of the prisoners. The prosecutor gives the following choice to each of the prisoners:

- Betray the others by testifying that the others committed the crime
- Cooperate with the others by remaining silent

On the basis of the answers received, the prosecutor decides to discount the sentence according to a payoff table. We assume that our games are composed by 2 prisoners, and therefore the payoff table is formed as follows:

- If the two prisoners cooperate, each player gets 5 years reduction

- If the two prisoners betray each other, each player gets 1 year reduction
- If a prisoner cooperates and the other one betrays, the first gets 2 years reduction and the other one gets 3 years reduction.

Your task is to implement a distributed application using a high-level communication middleware that allows prisoners to report their decision to a 'prosecutor' service, which will decide on the final sentence and communicate it to the prisoners. Such a system may be used for social experiments to study human behaviour versus an expected ideal behaviour[1]. The Wikipedia page presents some general details of the game[2], however, please note that while there is much more theory behind this game, the important focus of this practical is the development of the distributed system. All information about the game required for this practical is listed in this spec.

3.2 Part 1: Communication set up

This is a centralised architecture with a server and a single client. You are required to make use of one type of RPC, RMI or MOM communication middleware of your choice for the client/server communication. Some examples that we covered in class are: Apache Thrift, Java RMI, SOAP or REST Web Services, and Zero-MQ, but you may also choose an API of your choice. The chosen middleware should take care of the communication, while you will be required to specify the functionalities. Please do not use Sockets APIs directly but experiment with one of the available higher level approaches.

As starting functionality, provide an infrastructure for the following components:

- The game server
- The 'prosecutor' service made available by the game server
- A 'prisoner' client that is able to communicate with the service

When the infrastructure has been set up, ensure that the following operation can be performed for this first step:

- A single testing request: the prisoner client sends a simple request to the prosecutor service (e.g., "Test Connection") and prints out the results (e.g., "Hello from Server").

We assume that the 'prosecutor' service only deals with one case regarding two prisoners (P1, P2).

3.3 Part 2: Prosecutor Service 1 – Single Client Game

This part extends the infrastructure in part 1 for a simple single client game. We assume that there is a single 'prisoner' client which provides a text-interface to the user to gather the choices of prisoner P1. Subsequently the client sends the choice to the 'prosecutor' that determines whether there is any sentence discount. We assume that the second prisoner's choice P2 is already logged by the server (as a simple variable or input parameter). The operations for 'prisoner' client and 'prosecutor' service to be implemented are described below.

1. Prosecutor: initialise service with P2's choice and starts listening for requests.
2. Prisoner: introduce the task to the user (very brief)
3. Prisoner: ask the user to choose an option for the prisoner P1:
    - Cooperate [C]

[1] Grujic, J., Fosco, C., Araujo, L., Cuesta, J. A., & Sanchez, A. (2010). Social experiments in the mesoscale: Humans playing a spatial prisoner's dilemma. PloS one, 5(11), e13749.

[2] `https://en.wikipedia.org/wiki/Prisoner\%27s_dilemma`

- Betray [B]

4. Prisoner: contact the prosecutor and ask to process the prisoner's choice
5. Prosecutor: take P1(sent user choice) and P2(logged in server) choices and compare
6. Prosecutor: Decide automatically on the sentence according to the table below

Choice P1, P2	Reduction (years) P1, P2
C, C	5, 5
C, B	2, 3
B, C	3, 2
B, B	1, 1

7. Prosecutor: respond to prisoner with a decision for P1
8. Prisoner: print out the response from the prosecutor
9. Prosecutor: continue to listen for additional requests

3.4 Additional Extended Functionalities

It is strongly recommended that you ensure you have completed the Requirements in part 1 and 2 before attempting any of these additional requirements. You may wish to consider **one** of the following additional tasks, describing well the purpose of your extension in the report. Please ensure that the code for extended functionalities is separated from the previous parts.

1. **Many cases:** In part 2, we have assumed that only a single case is handled at a time. Develop a solution in which the prosecutor has many cases with associated responses from P2 to be dealt with at the same time. The prisoner P1 should sign up to a particular case, and you should assume that many clients could potentially contact the prosecutor with their requests.
2. **Additional service:** In part 2, we have assumed to communicate only with the 'prosecutor' service. In this task, you may consider an additional service that a prisoner may consult. For example, after receiving their updated sentence the prisoner may appeal to a court to have their discount increased. The new service may arbitrarily decide on whether to grant this or not.
3. **Two client players:** In part 2, we have assumed that a single prisoner client communicates with the service. Develop a solution in which we have two 'prisoner' clients that register with the 'prosecutor'. After receiving the choices from both clients, the prosecutor can communicate the sentence. For this task, you may use a simple polling mechanism, in which the prisoner repeatedly contacts the client to know whether there is an available sentence. Note that if you use a stateless service (e.g., REST service) you would need to provide some simple means to record participants' choices.
4. **Other options:** Suggest and develop your own additional functionality. New requirements should consider ways to extend the communication functionalities of the server or the client. If you are unsure about your proposed extension, please check with the lecturer.

4 Code and Report Requirements

4.1 Code and Libraries

Please develop your program selecting and testing a communication middleware of your choice. You may use a programming language of your choice but that is compatible with the middleware. Please include clear instructions on how to compile (if required) and run your system. You are free to use other libraries as long as they are used for functionalities that are not core to this assignment.

The solution submitted should contain all source code that you have developed. For certain types of middleware, you might need some external libraries or components, these may be excluded from the submission but please include clear instructions on how to set up your system including requirements. You may use automatic build scripts – eg. Maven.

Additionally, you may submit a single solution for both parts, 1 and 2, depending on what is attempted. If you attempt any of the additional requirements, please separate the extended solution from the main solution.

4.2 Report

You are required to submit a report describing your submission. The report should include:

- A brief list of the functionalities implemented and any extension attempted
- If any of the functionalities is only partially working, ensure that this is discussed in your report
- Design: A description and justification for the mechanisms you have implemented as part of your solution
- Examples and Testing: Some selected examples of the main functionalities (screenshots are encouraged) and your approach to testing
- Evaluation: A critical evaluation of the functionalities of your system, with particular attention to reflect on the type of middleware used
- Running: Include clear instructions on how to compile (if required) and run your system (this could also be on a readme file)

A suggested length for the report is around 800-1000 words.

5 Deliverables

You should submit a ZIP file via MMS by the deadline containing:

- A PDF report as discussed in Section 4.2.
- Your implementation of the solution including any source code.
- Any additional script required to run your system.

6 Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in the next section). The following issues will be considered:

- Adherence to the requirements

- Quality of the solution provided
- Example runs
- Insights demonstrated in the report
- Extension activities completed, if any

Some guideline descriptors for this assignment are given below:

8-10	A coding practical containing code that achieves some of the functionalities of part 1, adequately documented or reported.
11-13	A coding practical containing code that fully achieves the functionalities of part 1. The code submitted is of an acceptable standard, and the report describes clearly what was done, with good style.
14-16	A coding practical containing clear and well-structured code that fully achieves the functionalities of part 1 with some attempts to part 2. The submission includes a clear report showing a good level of understanding.
17-18	A coding practical containing clear, well-designed code that fully achieves the functionalities of part 1 and part 2. The submission includes code written in good style, together with a clear and well-written report showing real insight into the subject matter.
19-20	A coding practical containing code that fully achieves the functionalities of part 1 and part 2, including one additional extended functionality. The submission demonstrates unusual clarity of design and implementation, together with an outstandingly well-written report showing evidence of extensive background reading, a full knowledge of the subject and insight into the problem.

7 Policies and Guidelines

7.1 Marking

See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

7.2 Lateness Penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

7.3 Good Academic Practice

The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Alice Toniolo
(a.toniolo@st-andrews.ac.uk)
February 13, 2019