Sistemas Multinúcleo e Distribuídos
Mestrado em Engenharia Informática
Instituto Superior de Engenharia do Porto

TheyChat! A reliable chat system!

Introduction

The goal of this exercise is to develop a simple text-based messaging system called *TheyChat!*. The implementation will use Erlang and will be strongly based on Erlang's processes and message passing features focusing on reliability and fault-tolerance.

System behavior

TheyChat! uses a client/server architecture and a supervisor/worker design pattern. Clients connect to servers and can chat with each other (using the server). The focus of this exercise will not be the chat features but the reliability and fault-tolerance of the architecture. There will be three main nodes to implement:

Client : Able to join/leave servers and send/receive messages.

Router : Responsible to indicate to a client the PId of a server the client wishes to join and to monitor and solve server failures.

Server : This node is where clients can chat by sharing messages and also should implement fail-safe features.

In figure 1 we can see a proposal of the organization of nodes to accomplish the goal of this project. In more detail, note that processes *r*outer and *r*outer_monitor both supervise each other meaning that if one of them fails the other will restart it. The process *s*erver_monitor supervises (and restarts in case of failure) the *s*erver_chat process. The router is also responsible to restart the *s*erver_monitor in case of failure. It is very important to note that one of the features that the monitoring processes must provide is state backups, meaning that when some process fails, its supervisor must restart it with the state data it had when failed. For example, if a server process fails it must restart with the list of clients it had before crashing in such a way it can resume operation gracefully. It is sufficient to have all the processes running in the same machine (but in different Erlang Virtual Machines) since it is easy to escalate that solution to one based on different networked machines. The number and functionally of processes in the system should be the adequate to accomplish the goal and not necessarily the ones described in figure 1.
The communication protocol of the client/server iteration is not predefined but should allow the following actions:

- Join a given server.
- Leave a given server.
- Send and receive chat messages.

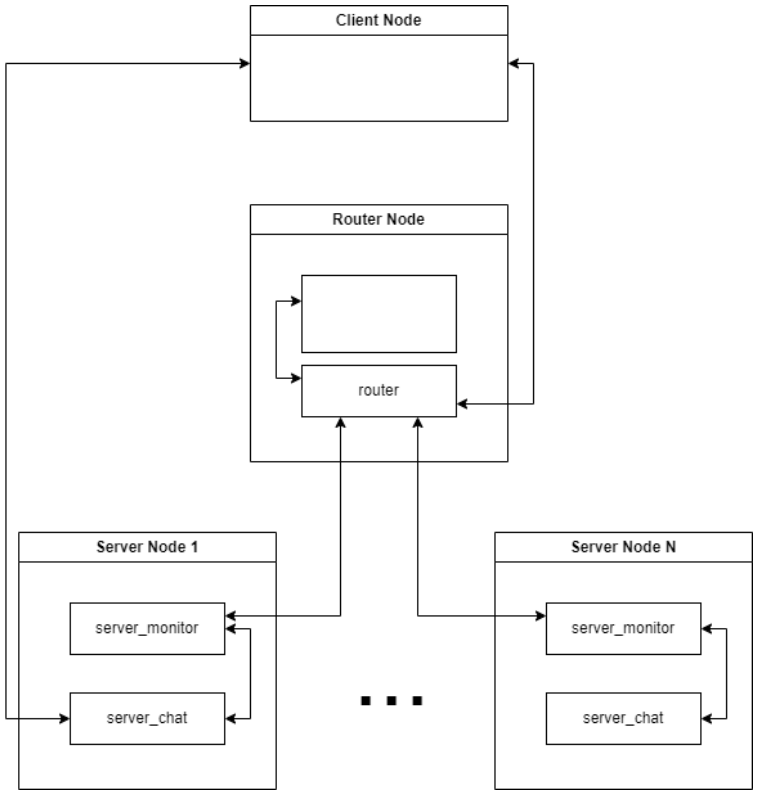

Figure 1: Client, router and server nodes

Error handling

Error handling should be possible when:

- Router fails
- Server fails
- Client fails

Error handling in this project has the broader meaning of restarting the failed processes resuming the system operation with minimal or without impact for the different entities. This will not be the case of clients. When a client fails, the server must notice it and update the client list so it will stop sending messages to that client.

Submission and presentation

No report is needed and only the submission of the client and server programs is necessary (deadline is 26^{th} of May). The presentation occurs from 27^{th} of May to 7^{th} of June.

Grading

This assignment is to be solved individually and will assess the knowledge about distributed programming in the functional language Erlang. Grading criteria is described

next.

Description	Percentage (%)
Multiple clients and at least one server communicating through messages	25
Router basic implementation	10
Router fault-tolerance	10
Router supervises server	10
Server basic functionality	15
Server fault-tolerance	15
Supports different servers (for different topics)	15

Code of honor

In "Código de boas Práticas de Conduta" from 27th of October of 2020, it is stated that students must submit a declaration as described in the Article 8 for each submitted project. This declaration is a signed commitment of the originality of the submitted work. Failure to submit this declaration will remove the work from evaluation. Submitting a work that does not comply with the signed declaration will have legal consequences.