Message Queues

Large Scale Distributed Systems

Message queuing systems support asynchronous communication allowing for temporal decoupling between communication peers, and therefore contributing to improved scalability. In addition to proper queues, they typically support publish/subscribe message dissemination.

ZeroMQ (ØMQ) is not a standard message queuing library. As Pieter Hintjens, one of its designers, writes in the Preface of the ØMQ Guide:

> ZeroMQ (also known as ØMQ, 0MQ, or zmq) looks like an embeddable networking library but acts like a concurrency framework.

In addition to simple message queuing and publish/subscribe, it supports other interaction patterns such as request/reply and task distribution. All this on top of different transports and handling buffering, re-connection and other properties.

The goal of these lab classes is to become familiar with the basic functionality of ØMQ and some of its supported patterns.

Suggested tasks for first class:

- Choose a language binding, e.g. C, and install the 0MQ system. Refer to `https://zeromq.org`.
- Using the guide at `https://zguide.zeromq.org` and the information in the *Basics* chapter, code and try the REQ/REP pattern.
  - Try running the client first and then the server
  - Try adding another server and see the distribution of requests.
  - Try killing and re-starting the server
  - Try making the communicating parties unreachable
- If using C, download *zhelpers.h* and try to simplify the REQ/REP code used earlier.
- You can also try the PUB/SUB example and play a bit with them.

Suggested tasks for second class:

- Study the 0MQ's polling (e.g. `zmq_poll()`) mechanism. It allows efficient access to multiple sockets from a single thread.
- Adapt the publisher in the first chapter to broadcast weather updates for PT zipcodes (4 digits). Adapt the client subscriber, using the poll mechanism, to fetch data from two publishers, one US and the other PT.
- Implement and test the *Shared Queue* pattern (with DEALER and ROUTER sockets).
  - Try running a single client and worker, with the broker.
  - Try adding another worker and see the distribution of requests.
  - Try killing and re-starting the broker.
  - Try to emulate network partitions (e.g. by running the different processes in different laptops and switching off the appropriate network interfaces)

Suggested tasks for third class:

- Implement and test a proxy to allow a client to subscribe to topics that are published by an unknown number of subscribers (dynamic discovery problem) using the XSUB and the XPUB sockets.
  - Try to implement the proxy by adapting the broker of the *shared queue* pattern
  - Adapt the publisher from the broadcast weather updates example.
  - Adapt the subscriber from the broadcast weather updates example (if needed).
  - Try killing and re-starting the proxy.
  - Try to emulate network partitions (e.g. by running the different processes in different laptops and switching off the appropriate network interfaces).
- Explore PUSH/PULL ZMQ-Sockets with the Divide and Conquer example.