Distributed Algorithms 2014/15
Bonus project

Implementation of *Uniform Reliable Broadcast*

1 Overview

The goal of this practical project is to implement *Uniform Reliable Broadcast* in a system of 5 processes using message passing. Each process broadcasts messages to other processes, making sure that the properties of *Uniform Reliable Broadcast* are satisfied. Messages exchanged between processes may be dropped, delayed or reordered by the network. The execution of processes may be paused for an arbitrary amount of time and resumed later. Processes may also fail by crashing at arbitrary points of their execution.

2 Technical specification

2.1 Processes

One process is represented by one Linux process. Process `n` is started by executing

```
da_proc n addr_1 port_1 addr_2 port_2 addr_3 port_3 addr_4 port_4 addr_5 port_5 m
```

where `n` $\in \{1, 2, 3, 4, 5\}$ is the ID of the process, `addr_k` is the IP address of process `k` and `port_k` is the port on which process `k` listens for incoming network packets. The last parameter, `m`, is the number of messages to be broadcast. A value of `-1` for the parameter `m` makes the process constantly broadcast messages until it is stopped. For example, running all processes locally, each broadcasting 10000 messages, they could be started using the following commands

```
da_proc 1 127.0.0.1 11001 127.0.0.1 11002 127.0.0.1 11003 127.0.0.1 11004 127.0.0.1 11005 10000
da_proc 2 127.0.0.1 11001 127.0.0.1 11002 127.0.0.1 11003 127.0.0.1 11004 127.0.0.1 11005 10000
. . . etc. . . .
```

A process performs all necessary initialization tasks on startup, but it does

not automatically start broadcasting messages. This enables all processes to start and initialize. A process only starts broadcasting messages after receiving the USR1 signal.

A process that receives a TERM or INT signal must immediately stop its execution with the exception of writing to an output log file (see Section 2.3). In particular, it must not send or handle any received network packets. This is used to simulate process crashes. You can assume that only a minority (i.e. at most two) processes crash in one execution.

2.2 Messages

Inter-process point-to-point messages (at the low level) must be carried exclusively by UDP packets in their most basic form, not utilizing any additional features (e.g. any form of feedback about packet delivery) provided by the network stack, the operating system or external libraries. Everything must be implemented on top of these low-level point to point messages.

The application messages (i.e. those broadcast by processes) are numbered sequentially at each process, starting from 1. Thus, each process broadcasts messages 1 to `m`. The data carried by an application message is only the sequence number of that message.

2.3 Output format

The output of each process is a text file. For process `n`, the text file is named `da_proc_n.out` and contains a log of events. Each event is represented by one line of the output file, terminated by a Unix-style line break (`'\n'`). There are two types of events to be logged:

- broadcast of application message, using the format
  `b` *`seq_nr`*
  where *`seq_nr`* is the sequence number of the message
- delivery of application message, using the format
  `d` *`sender seq_nr`*
  where *`sender`* is the number of the process that broadcast the message and *`seq_nr`* is the sequence number of the message.

An example of the content of an output file:

```
b 1
b 2
b 3
d 2 1
d 4 2
b 4
```

Note: The most straight-forward way of logging the output is to append a line to the output file on every broadcast or delivery event. However, this may harm the performance of the application. You might consider more sophisticated logging approaches. Also note that even a crashed process needs to output the sequence of events that occurred before the crash. You can assume that a

process crash will be simulated only by the TERM or INT signals. Remember that writing to files is the only action we allow a process to do after receiving a TERM or INT signal.

3 Compilation

All submitted applications will be tested using Ubuntu 14.04 running on a 64-bit architecture. The submission has to contain all sources of the application. All submitted files are to be placed in one folder, such that executing `make` in that folder produces all necessary executables (at least `da_proc`, which will be called by the testing scripts).

If a language other than C/C++ or Java is used, please indicate packages that need to be installed on the system in a file README.TXT.

4 Template

A simple C template is provided that shows a possible high-level structure of the application. The file `da_proc.c` contains a simple code skeleton that may serve as a starting point for developing the application.

Two shell scripts are provided: `test_correctness` and `test_performance`. They demonstrate how approximately the correctness and performance, respectively, will be tested. You may extend or modify these scripts to test your own application. In particular, no code is yet provided to evaluate the correctness of the output files.

5 Testing and grading

The submitted application will be first tested for correctness under various conditions (packet loss, reordering, delays, simulated asynchrony of processes, crashes, etc...).

If it passes all correctness tests (i.e. if the resulting output logs are consistent with the definition of Uniform Reliable Broadcast), it will be tested for throughput, i.e. the total number of messages delivered by all processes per second. When testing throughput, no artificial network artifacts (such as packet loss or reordering) or process delays/crashes will be simulated.

The authors of the five fastest (in terms of throughput) applications will be rewarded with an increase of their final course grade[1] by 0.5.

6 Cooperation

This project is meant to be completed individually. Copying of other students' solutions is prohibited. You are free to discuss the projects with others, but the submitted source codes must be your own work. Multiple copies of the same code will be disregarded without investigating which is the "original" and which is the "copy".

[1] Does not apply for a final course grade of 6.