Distributed Algorithms 2020-2021

Project description

Teaching Assistants:
Arsany Guirguis, Athanasios Xygkis

Extra support:
Ali El Abridi

Goal of the project

Implement certain building blocks necessary for distributed systems:

- Focus on broadcast algorithms

- 1st deliverable: FIFO broadcast

- 2nd deliverable: Causal broadcast

Notes:

- Be modular: work from the 1st deliverable should be used in 2nd,
- Delivery dates will be announced!

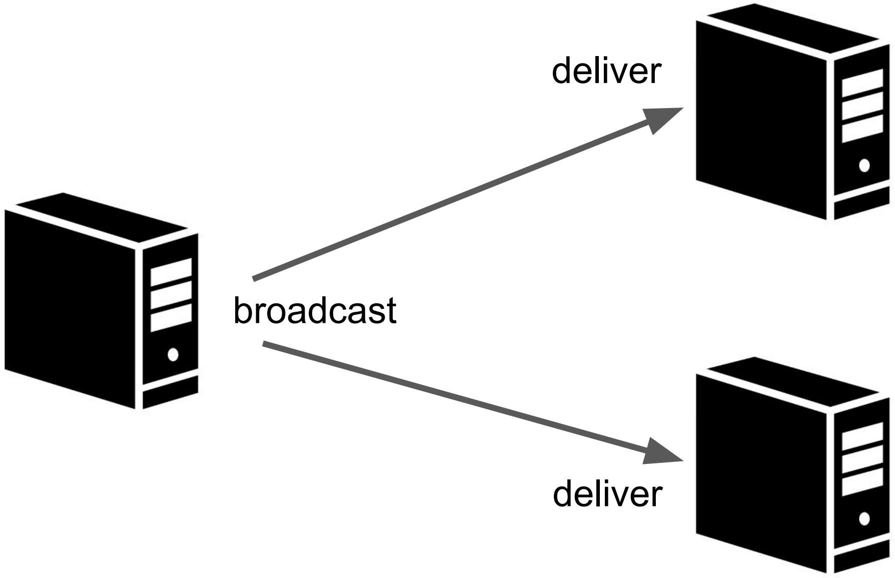
Grading scheme

- The project accounts for 30% of the final grade.
- You work alone:
  - You can discuss problems and solutions with your colleagues,
  - There is an oral exam where you explain what you did.
- We prioritize correctness over performance.
- We have zero tolerance to cheating!
  - Always give credit to stuff you find online (e.g. in stackoverflow)

Introduction to broadcast

Goal: Ensure reliable message delivery to all participants.

Informally:



Basic assumptions

- *Asynchrony*:
  No bounds on messages and process execution delays

- *Failures*:
  Processes fail by crashing.

  They stop executing instructions/actions after the crash.

- *Communication*:
  Messages are not lost, only delayed.

  Messages are not created unless a process sent them.

  Messages are delivered only once.

Best-Effort Broadcast

Properties:

- *Validity*:

  If *p* and *q* are correct, then every message BEB-Broadcast by *p* is eventually BEB-Delivered by *q*.

- *Integrity*:

  Message *m* is BEB-Delivered by a process at most once, and only if it was previous BEB-Broadcast.

Pseudocode (executed by p):

```
upon BEB-Broadcast(m):
    foreach q in Π:
        send m to q

upon receive(m):
    BEB-Deliver(m)
```

Reliable Broadcast

Properties:

- Validity:
  If a correct process R-Broadcasts message *m*, then it eventually R-Delivers *m*.

- Integrity:
  Message *m* is R-Delivered by a process at most once, and only if it was previously R-Broadcast.

- Agreement:
  If a correct process R-delivers message *m*, then all correct processes eventually deliver message m.

Reliable Broadcast

Pseudocode (executed by p):

```
initialization:
    SET delivered := {}

upon R-Broadcast(m):
    send m to $\Pi\setminus\{p\}$
    R-Deliver(m)
    delivered := delivered $\cup$ {m}
```

```
upon receive(m) from q:
    if m not it delivered:
        send m to $\Pi\setminus\{p, q\}$
        R-Deliver(m)
        delivered := delivered $\cup$ {m}
```

Project requirements: Basics

- Allowed languages:
  - C11 and/or C++17
  - Java 11
- Complication method:
  - CMake for C/C++
  - Maven for Java
  - We provide a template for both.
- Allowed 3rd party libraries: **None**

Project requirements: Messages

- You are allowed to use only UDP packets in their most basic form:
  - Point to point (no broadcast)
- You are not allowed to use third party libraries
  - Everything must be implemented on top of UDP packets
- TCP is used only for the barrier, during process initialization (explained later)
- Application messages are numbered sequentially at each process
  - They starting from number 1 and can go up to 2^64 - 1,
  - By default, their only payload is the sequence number.

Project requirements: Interface

Your deliverables should support the following command line arguments:

Usage: ./run.sh --id ID --hosts HOSTS --barrier NAME:PORT --output OUTPUT [config]

Where:

- **ID** is the id of the process.
- **HOSTS** is the path to a file that contains information about every process in the system.
- **NAME:PORT** is the IP and port of the barrier, which ensures that all processes have been initialized before broadcasting starts.
- **OUTPUT** is the path to a file that stores the output of the process.
- **config** is the path to a file that contains specific information required from the deliverable (e.g. processes that broadcast)

We provide source code that does the parsing for you!

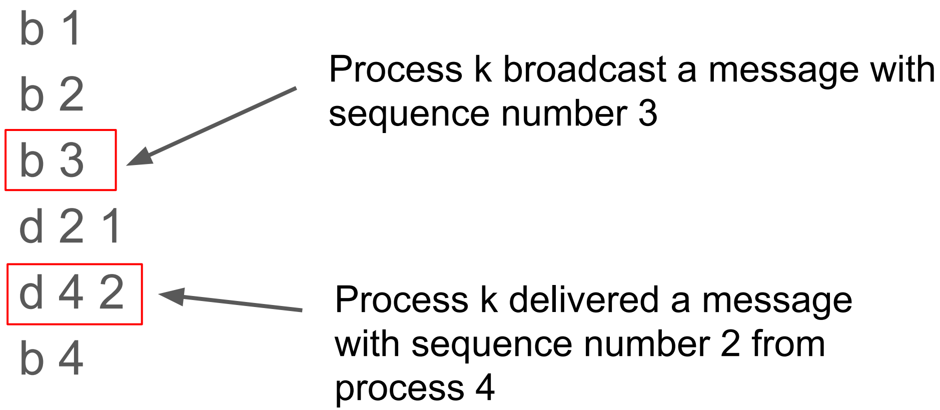
Project requirements: Signal Handlers

- Processes perform all necessary initialization tasks on startup.
  - Subsequently, they wait on a barrier.
  - The barrier is released when all processes reach it.
- A process that receives a SIGTERM or SIGINT signal must immediately stop its execution.
  - The process can is only allowed to write to the output log file after one of the signals is received.
  - It must not send or handle any received network packets.
  - This is used to simulate process crashes.
  - You can assume that at most a minority (e.g., 1 out of 3; 2 out of 5; 4 out of 10, ...) of processes may crash in an execution.

Project requirements: Output format

- The output of a process is a text file.
  - You specify the location of the output file using the **--output** argument.
  - The files contains events. Each event is represented by one line of the file:

E.g. the output of process k may contain:



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

Process k broadcast a message with sequence number 3

Process k delivered a message with sequence number 2 from process 4

TCP vs UDP

TCP	*UDP*	*Comment*
Connection-oriented	Connectionless	UDP does not establish a connection before sending data.
Reliable stream	Unreliable datagram	UDP is unreliable, it does not provide guaranteed delivery and a datagram packet may be lost in transit.
Flow control	No control	With UDP, packets arrive in a continuous stream or they are dropped.
Ordered	Unordered	TCP does ordering and sequencing to guarantee that packets sent from a server will be delivered to the client in the same order they were sent. On the other hand, UDP sends packets in any order.

Source: https://www.privateinternetaccess.com/blog/tcp-vs-udp-understanding-the-difference/

Initialization barrier

We allow you use a barrier to easily initialize your applications.

Start the barrier:

```
Usage: barrier.py [-h] [--host HOST] [--port PORT] --processes PROCESSES
```

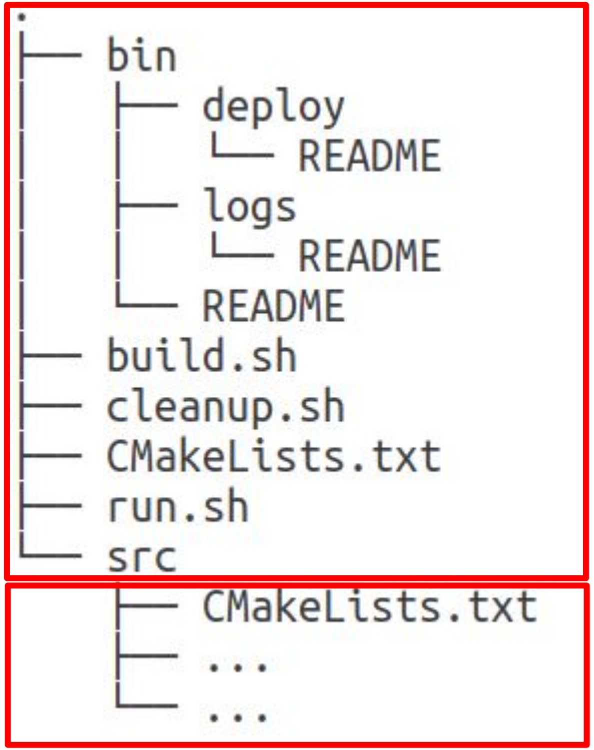
E.g. to wait for 3 processes, you can run as follows:

```
./barrier.py --processes 3
```

When 3 connections are established to the barrier, the barrier closes all the connections, signaling applications to start.

C/C++ Project Template

Deliver the project in a .zip file that has the following structure:



```
.
├── bin
│   ├── deploy
│   │   └── README
│   ├── logs
│   │   └── README
│   └── README
├── build.sh
├── cleanup.sh
├── CMakeLists.txt
├── run.sh
└── src
    ├── CMakeLists.txt
    ├── ...
    └── ...
```

Do not edit this section!
Your binary should not create make use of directories “deploy” and/or “logs”.
Run:

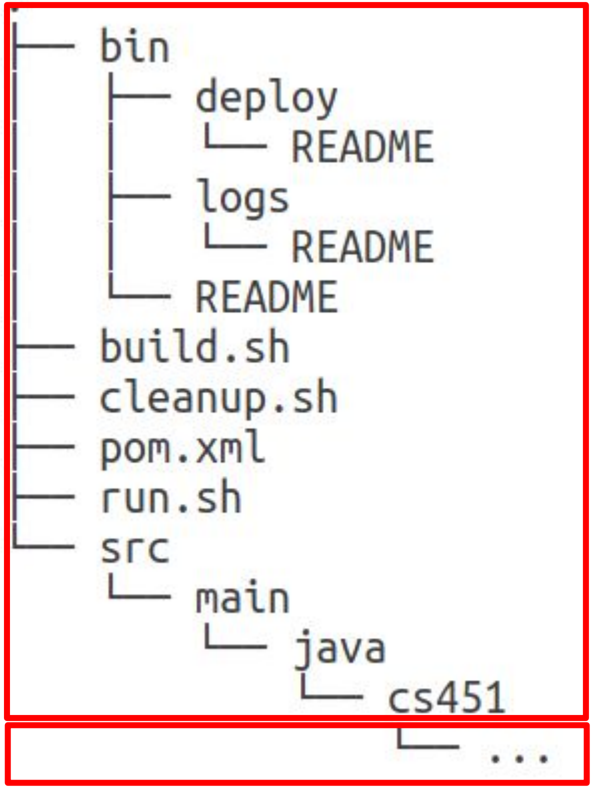
- *build.sh* to **compile** the code
- *run.sh <args>* to **run** the code
- *cleanup.sh* to **delete build artifacts** (use when submitting your project)

You can arrange your project files as you see fit.

Do not change the name of the executable in the CMakeLists.txt

Java Project Template

Deliver the project in a .zip file that has the following structure:



```
├── bin
│   ├── deploy
│   │   └── README
│   ├── logs
│   │   └── README
│   └── README
├── build.sh
├── cleanup.sh
├── pom.xml
├── run.sh
└── src
    └── main
        └── java
            └── cs451
                └── ...
```

Do not edit this section!
Your binary should not create make use of directories “deploy” and/or “logs”.
Run:

- *build.sh* to **compile** the code
- *run.sh <args>* to **run** the code
- *cleanup.sh* to **delete build artifacts** (use when submitting your project)

Note: Do not edit pom.xml, i.e. do not use 3rd libraries in maven!

You can arrange your project files as you see fit.

Compilation environment (1/2)

- All submitted compilation will be tested using Ubuntu 18.04 (x86_64) with:
    - **gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0**
    - **g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0**
    - **cmake version 3.10.2**
    - **OpenJDK Runtime Environment (build 11.0.8+10-post-Ubuntu-0ubuntu118.04.1)**
    - **Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)**

Compilation environment (2/2)

- **Using the provided templates is mandatory!**
- **It is your responsibility to make your project compile using the templates!**
- **Projects that fail to compile will not be considered!**
- We provide a virtual machine with the exact versions of build tools mentioned above
  - Check that your project compiles (and runs) there, without any modification to the VM (e.g. without any added package),
  - During development, you are allowed to modify the VM.