Distributed Systems – HS 2017
Assignment 3

Mihai Bâce:
mihai.bace@inf.ethz.ch
Outline

- Review of logical time and UDP
  - Causality
  - Lamport Timestamps
  - Vector Clocks

- Assignment 3

Dates:
Start: October 23, 2017
End: November 2, 2017 11:59 PM
The User Datagram Protocol

- Simple transmission model
  - No hand-shakes, ordering, data integrity
  - Datagrams can be delayed, out of order, missing
TCP vs UDP (a brief comparison)

<table>
<thead>
<tr>
<th>Transmission Control Protocol</th>
<th>User Datagram Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection oriented</td>
<td>Connectionless</td>
</tr>
<tr>
<td>High reliability applications,</td>
<td>Fast, efficient applications</td>
</tr>
<tr>
<td>time is less critical</td>
<td></td>
</tr>
<tr>
<td>Heavyweight</td>
<td>Lightweight</td>
</tr>
<tr>
<td>Handle reliability</td>
<td>No guarantees</td>
</tr>
<tr>
<td>Congestion control</td>
<td></td>
</tr>
<tr>
<td>Data remains intact and in the</td>
<td>No ordering of messages</td>
</tr>
<tr>
<td>correct order</td>
<td></td>
</tr>
</tbody>
</table>
UDP Effects

"What is the first prime number after 1000000?"

"QBot"

"P2 answered correctly!"

"P1"

"???!!"

"P2"

"Yeah!"
Interesting property in Distributed Systems

Causal relationship < (“happened before”)

\[ x < y \iff (x, y \text{ on same process, } x \text{ happens before } y) \text{ or } (x \text{ is sent and } y \text{ is correspondingly received}) \text{ or } (\text{transitivity}) \]

I USED TO THINK CORRELATION IMPLIED CAUSATION.

THEN I TOOK A STATISTICS CLASS. NOW I DON’T.

SOUNDS LIKE THE CLASS HELPED. WELL, MAYBE.
Software clocks

- Ideal real time
  - Transitive, dense, continuous

- No access to global clock
- Difficult to perfectly synchronize local clocks

- Logical time
  - Lamport Timestamps
  - Vector clocks
  - Matrix clocks
Lamport timestamps

- Use a single clock value
  - Local event: Local clock tick
  - Send event: Attach local clock value
  - Receive event: Max(local clock, message clock)  
    First Max, then tick

- Satisfies clock consistency condition:
  \[ e < e' \rightarrow C(e) < C(e') \]
Lamport Timestamps

- Do not satisfy the **strong clock consistency condition**

\[ e < e' \iff C(e) < C(e') \]
Vector Clocks

- Refinement of Lamport timestamps
- Each process keeps one counter

- Satisfies the strong consistency condition!

\[ e < e' \iff C(e) < C(e') \]
Vector clocks

"What is the first prime number after 1000000?"

"P2 answered correctly!"

"Yeah!"
Vector clocks

Process i stores local information on what it thinks about the local time of process (1,…,n)
Outline

- Review of logical time and UDP
  - Causality
  - Lamport Timestamps
  - Vector Clocks

- Assignment 3

Dates:
Start: October 23, 2017
End: November 2, 2017 11:59 PM
A mobile chat-like application

- Task 1: Getting familiar with Datagrams (UDP)
- Task 2: Lamport Timestamps and Vector Clocks
- Task 3: Message ordering based on Vector Clocks
- Task 4: Mini-Test
1. Getting familiar with Datagrams

- Client “registration” and “deregistration” service
- Use Datagrams
- Send message to the server, wait for acknowledgement
- Retry mechanism
  - If there is no “ack”, retry 5 times
- When successful, display a notification (e.g. Toast) and transition to a new activity
1. Getting familiar with Datagrams

Hints:

- Sending / Receiving UDP packets are network operations
- Do not use the main UI thread
  - One solution: AsyncTask
  - Careful with multiple AsyncTasks! They are executed sequentially.
- The client must always listen for received/incoming messages (up to a certain timeout)
- Receiving messages is a blocking operation!
1. Getting familiar with Datagrams - The Server

- Server will be deployed on your local machine
- Launch “chat_server.jar” from the command line
- Can use the emulator or the phones

```
java -jar chat_server.jar
```

Server started
Server IP address : 192.168.192.38
Server port : 4446
2. Implementing Lamport Timestamps and Vector Clocks

- Clock interface
- Implement all the methods
- For each type, some additional methods (check sheet)

- Use the unit tests for validation

- No server needed for this task

```java
package ch.ethz.inf.vs.a3.clock;

public interface Clock{
    /**
     * Update the current clock with a new one, taking into
     * account the values of the incoming clock.
     *
     * E.g. for vector clocks, c1 = [2 1 0], c2 = [1 2 0],
     * the c1.update(c2) will lead to [2 2 0].
     *
     * @param other
     */
    public void update(Clock other);

    /**
     * Change the current clock with a new one, overwriting the
     * old values.
     *
     * @param other
     */
    public void setClock(Clock other);

    /**
     * Tick a clock given the process id.
     *
     * For Lamport timestamps, since there is only one logical time,
     * the method can be called with the "null" parameter. (e.g.
     * clock.tick(null).
     *
     * @param pid
     */
    public void tick(Integer pid);

    /**
     * Check whether a clock has happened before another one.
     *
     * @param other
     * @return True if a clock has happened before, false otherwise.
     */
    public boolean happenedBefore(Clock other);

    /**
     * toString
     *
     * @return String representation of the clock.
     */
    public String toString();

    /**
     * Set a clock given it's string representation.
     *
     * @param clock
     */
    public void setClockFromString(String clock);
}
```
3. Message ordering based on Vector Clocks

- Client requests a chat log from the server
- Datagrams
  - Messages can arrive in any order. Cannot display them yet!
- Store messages in a buffer
- Order them
- Use the happened before method
3. Message ordering based on Vector Clocks

- Buffer the incoming messages in a Priority Queue
  - Priority Queue: priority heap, which orders the elements according to their natural order or according to the comparator specified at construction time
  - Use the provided implementation of the PriorityQueue from the code skeleton

- Implement a Comparator for your messages

- Every incoming message will be inserted in the correct place
Message Structure - Sample

- JSON

- “header”
  - “username”: ”John” (String)
  - “uuid”: ”ae4e15ff-b589-4e85-a07c-594b16e4e645“ (String)
  - “timestamp”: "{"0":2,"1":0,"2":0}" (Map/HashMap for Vector Clocks)
  - “type”: “message” (String)

- “body”
  - “content”: “Hello” (String)
{  
  "header": {  
    "username": "server",  
    "uuid": "ac31f345-a8b1-4241-b939-9d3527f14483",  
    "timestamp": "{\"0\":2,\"1\":0,\"2\":0}\"  
  },  
  "type": "message",  
  "body": {  
    "content": "A1"  
  }  
}
Sample Application Design

Register

Settings

Enter your name

john

Join
Settings

Server address:
10.0.2.2

Server port:
4446

Retrieve chat log

Deregister

GET CHAT LOG
Android SDK Tools

- **Android Debug Bridge (adb tool)**
  - You can find the adb tool in `<sdk>/platform-tools/`

- **Android Emulator**

- **Setting up a port forwarding**
  - `adb forward tcp:port1 tcp:port2`
  - Forwards the local port `port1` on the machine to `port2` on the emulator.
  - Example: `adb forward tcp:12345 tcp:8088`

- **JUnit Testing**
  - [http://tools.android.com/tech-docs/unit-testing-support](http://tools.android.com/tech-docs/unit-testing-support)
Have fun!

**How to write good code:**

1. Start project.
2. Do things right or do them fast?
   - Fast: Code fast, does it work yet?
   - Right: Code well, are you done yet?
   - No: Almost, but it's become a mass of kludges and spaghetti code.
   - No, and the requirements have changed: Throw it all out and start over.
3. Good code.