<u>* THIS DOCUMENT IS ONLY FOR LOGS NOT FOR ELABORATION *****</u>**

First meeting 12/02/2015 Maarit

Attendance : We all participated

Data exchange
- Name
- GPS Location
- Postal Address

Server behaviour:
- The servers store location of devices as GPS position.
- We start with 2 servers receiving information from clients.
- Servers synchronize information to keep updates copies of locations
- OPTIONAL - Store queries that can not be processed yet
- Module 1 - Database manager - Raghu
- Module 2 - Tcp manager to keep sync among servers - Anteneh
- Module 3 - Udp manager to accept and replay queries from clients - Jaume

Client behaviour:
- It tries to connect to a pool of addresses and chooses one randomly.
- Informs about present location (automatically)
- Send query
- Receives informs and prints
- OPTIONAL - Ask about client location within a time frame
- Module 4 - Client (takes care of locations and data) - Raul

Protocols:
- UDP between client - server
- TCP between servers
- Client have identifier no security requirements

<u>Second meeting 23/02/2015 Maarit</u>

Attendance : We all participated

Questions to be addressed

We all need to come with our homework done, at least answers to this questions:

- What tasks is my module responsible for?
- What communications my module is involved with? Is acting as server or client?
- What kind of server architecture better addresses my requirements? Do you have firewall / NAT problems?
- With what other modules I share program? Do we share features? Do we share libraries? What interfaces do you plan to use? Pipes? Shared memory?
- Do I need to define a state machine? A timeline execution?
- Is there any module you need to validate your logic?

Overall architecture

Server side:

- Thread 1 - Module 1 - Database manager
- Thread 2 - Module 2 -

Module 1 - Database manager - Raghu

Module is responsible for

- Generate client IDs
- Receiving querys from clients and encode response
- Receiving msgs from server and change particular entry

With what other modules I share program? Do we share features? Do we share libraries?

- How do we encode messages in the query?
- Communications with Module 3 and Module 2 are message queues.

Module 2 - Tcp manager to keep sync among servers

Module is responsible for

- General objective: Synchronizing database between servers
- Send 1 x n notification of database updates

Module 3 - Udp manager to accept and replay queries from clients

Module is responsible for:

- UDP communication in the server side
    - Open socket server and listen to clients
    - Replay to clients about other client position
    - Replay to clients about position correcly updated
- UDP communication in the client side
    - Inform server about its new position
    - Request position of other client

What kind of server architecture better addresses my requirements?

- We are talking about a UDP server receiving requests from multiple clients.
- The number of clients can be large but the traffic exchanged is small
- No need for threads. Select with 1 thread can be enough. Max parent + 1 pre-fork child.
- We assume work in IPV6 network with no firewalls.

With what other modules I share program? Do we share features? Do we share libraries?

- I share with both client and server programs
- Client: I have to know when to update my position
- Client: I have to inform about server replays
- Server: Inform about client new position
- Server: Inform about client request
- Communication : message queue.

Do I need to define a state machine? A timeline execution?

- Yes. State machine to coordinate with other modules.
- Timeline execution only for my UDP communication.

Is there any module you need to validate your logic?

- Client: Raul
- Server: Database manager

Module 4 - Client (takes care of locations and data)

CLIENT TO SERVER COMMUNICATION

NEW CLIENT

- CLIENTID$TYPEOFCOM$SERVICE$<string>$LOCATIONUPDATE\n
- example: 0$\n

QUERY MSG

- CLIENTID$TYPEOFCOM$SERVICE$<string>$LOCATIONUPDATE\n
- example: 000001$QUERY$NAME$raghu$10.19.19\n
- example: 000001$EMAIL$rahgu@aalto.fi$2890182\n

UPDATE MSG

- CLIENTID$TYPEOFCOM$SERVICE$<string>$LOCATIONUPDATE\n
- example: 000001$UPDATE$NAME$raghu$10.19.19\n

Third meeting 13/03/2015 Maarit

Attendance : Raghu, Anteneh and Jaume

Objective : Present code that everyone has developed until today.

Raghu presents the data structures that modules are going to use to share information through messages queus. Information about client is stored in a hastable that stays in memory.

Master server role:

- Servers only send client's updates to master server
- Master server is responsible for sending updates to all other servers
- OPTIONAL - All servers shall send keep alive msg to master so master can update them properly.
- Master provides ids to all servers
- Master server keeps list of running servers and their status
- OPTIONAL - Servers fight to become master

Decisions:

- **Modules will work as independent threads and exchange data through ONE SINGLE message queue.**
- Server and client modules send recieved information using msg structure
- **Msg strucure is made of client or server address and exchanged message and CLIENT OR SERVER UDP MSG.**
- Database thread will send replays straight to clients and servers using information exchanged in the data strucure.
- **Server to server is UDP connection.**
- At start-up, server will read IP addresses from all servers from a file and try to establish connections
- Server module establishes tcp sockets with a other servers, creates select and shares with database module.

SERVER TO SERVER (MASTER) MESSAGES

NEW SERVER

- SERVERID$
- example: 00000001$\n

SERVER BACK ONLINE

- SERVERID$TYPEOFCOM$
- example: 000001$NEW$000001$NAME$raghu$10.19.19\n

TASKS

Raghu

- Read from both mq with select and test current core.
- Initialize mutex inside database module
- Sending code to servers and clients
- Share logging code

Anteneh

- At start-up, server will read IP addresses from config file try to send hello word "message connections" with the master.
- Initial SERVER ID is something like 000000001 and shares with master server. In return it gets new ID and stores back to file and stores as global variable.
- Server ID should be stores on disk so it's premanent
- Develop unit test

Jaume

- 2 UDP socket that listens to different ports and fills msg queue
- Select reads from them and puts to the msgq
- Develop unit test

<u>Further improvements</u>

- How to sync new servers with full database
- Send ACK to clients and maybe check for integrity with checksum

Integration date : **Integration meeting on Friday 20th in Maarit.**

Forth meeting 20/03/2015 Maarit

Attendance : Module to module meetings have taken place to address specific integration issues.

Objective : Accelerate integration.

Module 1 and Module 3.

Message queu will exchange pointers instead of data because threads share memory. This will be faster.

Merging will be done by Raghu. Jaume will coordinate with Raul to move udp testing forward.

Fifth meeting 01/04/2015 Maarit

Attendance : All

Objective : Testing of applications.

We have 3 applications communicating: client, server and master server. During this meeting correct exchange of messages between client and server was performed as well as proper forwarding by the master server.

Updates on documentation were also discusssed as new packet diagrams for communication protocol chapter, block diagram for implementation chapter and tables to be filled by all participants.