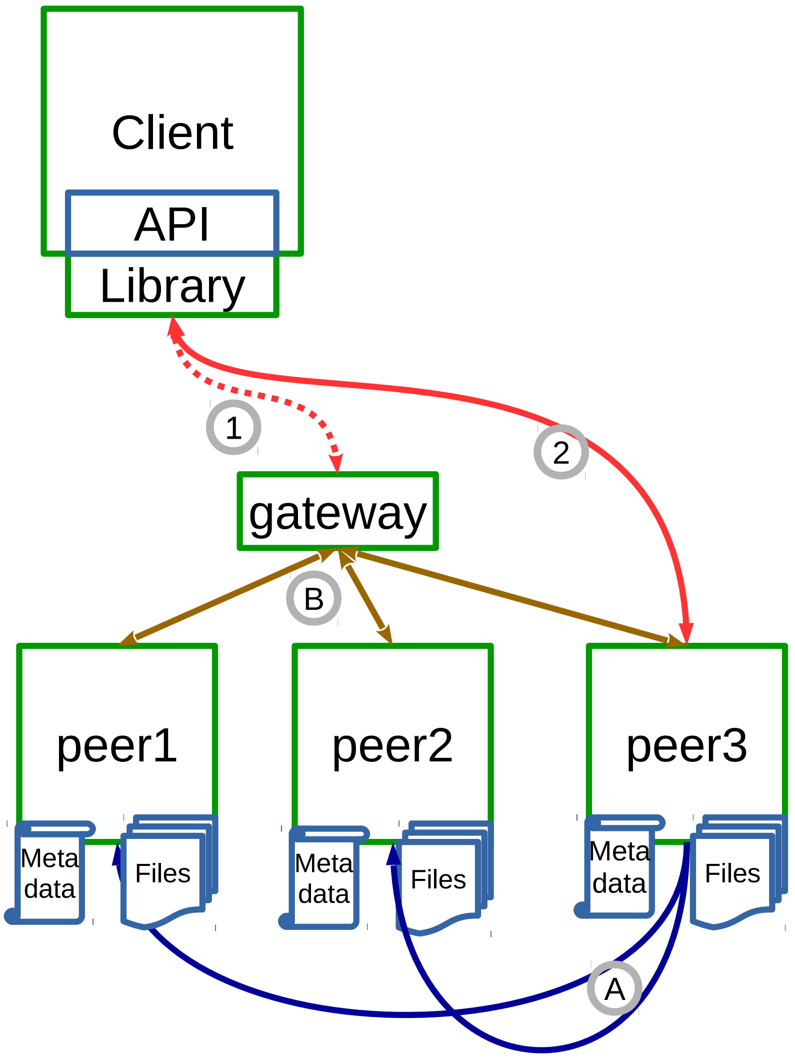
System Programming (MEEC/MEAer)

Project Assignment 2016/2017

In this project the students will implement a simple distributed fault tolerant repository for photos.

Students will implement a set of components that allow other programmers to implement clients that perform queries, and store and retrieve photos. These programms are written with using a predefined API.

The architecture of the repository is distributed, composed of various components, as follow:



1 Architecture

The components of the system are:

- API - Set of interfaces that allow programmers to develop clients that access files on the system.

- Library - Implementation of the API, with the code for the client to interact with the gateway and peers.
- Gateway - component responsible for the establishing of connection between the clients and peers. This component should store an updated list of available peers.
- Peers - components that store the files (and their metadata), and answers to the requests from the clients.

The fault tolerance is maintained by storing a copy of all photos in all peers.

In this project students should implement the following components (marked in green in the previous figure):

- Simple client to evaluate the functionality of the system
- Library - a set of .c files that implement the API
- Gateway - one process in a well known address
- Peers - one process that knows the address of the gateway

1.1 Overall functionality

When a client first connects to the system, it first contacts the gateway (interaction 1), the gateway returns the identification (address) of one of the peers to whom the client connects to (interaction 2).

All requests are performed between the client and one of the peers (iteration 2).

Photos (and keywords) are added to one peer but are replicated to all other peers (iteration 2 and A) . When a photo is deleted it is also removed from all other peers.

When a new peer is added to the system he should notify the gateway (interaction B) and receive a copy of all the available data. The gateway should also be notified when one of the peers dies.

All the components should be able to execute in multiple, different machines.

2 API

In order to implement client applications the system will provide an API composed of the following functions:

```
int gallery_connect(char * host, in_port_t  port)
```

This function is called by the client before interacting with the system.

Arguments:

- **host** and **port** arguments refer to the identification of the gateway server socket.

This function returns one integer corresponding to the socket opened between the client and one of the peers.

The function return -1 if the gateway can not be accessed, 0 no peer is available or a positive integer if a preer is accessible (this returned value is used in all other functions (peer_socket)).

```
uint32_t gallery_add_photo(int peer_socket, char *file_name)
```

This function inserts a new photo into the system.

Arguments:

- **peer_socket** – this argument corresponds to the value returned by connect
- **file_name** – This argument corresponds to the name of the photo file (including extension). The photos should be read from the program directory.

This function returns positive integer (corresponding to the photo identifier) or 0 in case of error (Invalid file_name or problems in communication with the server).

```
int gallery_add_keyword(int peer_socket, uint32_t id_photo,
                        char *keyword)
```

This function add a new **keyword** to a photo already in the system.

Arguments:

- **peer_socket** – this argument corresponds to the value returned by connect
- **file_name** – This argument corresponds to the name of the photo file (including extension)

```
int gallery_search_photo(int peer_socket, char * keyword,
                         uint32_t ** id_photos)
```

This function searches on the system for all the photos that contain the provided **keyword**.

The identifiers of the found photos will be stored in the array pointed by **id_photos**. This argument will point to an array created inside the function using **calloc**. The length of the **id_photos** array will returned by the function.

Arguments:

- **peer_socket** – this argument corresponds to the value returned by connect
- **keyword** – This argument corresponds to the keyword that will be used in the search
- **id_photos** – The id of the photos will be stored in an array (created inside this function). The **id_photos** parameter points to the variable where the address of this array will be stored.

This function returns -1 positive integer (corresponding to number of found photos), 0 if no photo contains the provided keyword, or a -1 if a error occurred (invalid arguments or network problem.

```
int gallery_delete_photo(int peer_socket, uint32_t  id_photo)
```

This function removes from the system the photo identified by id_photo.

Arguments:

- peer_socket – this argument corresponds to the value returned by connect
- id_photo – This argument is the identifier of the photo to be removed from the system

This function returns 1 if the photo is removed successfully, 0 if the photo does not exist or -1 in case of error

```
int gallery_get_photo_name(int peer_socket, uint32_t id_photo,
              char **photo_name)
```

This function retrieves from the system the name of the photo identified by id_photo.

Arguments:

- peer_socket – this argument corresponds to the value returned by connect
- id_photo – This argument is the identifier of the photo to be removed from the system
- file_name – The name of the photo will be stored in an array (created inside this function). The **photo_name** parameter points to the variable where the address of this string will be stored

This function returns 1 if the photo existes in the system and the name was retrieved, 0 if the photo does not exist or -1 in case of error.

```
int gallery_get_photo(int peer_socket, uint32_t id_photo,
              char *file_name)
```

This function downloads from the system the photo identified by id_photo and stores it in the file with the **file_name** name.

Arguments:

- peer_socket – this argument corresponds to the value returned by connect
- id_photo – This argument is the identifier of the photo to be removed from the system
- file_name – This argument is the name of file that will contain the image downloaded from the system

This function returns 1 if the photo is downloaded successfully, 0 if the photo does not exist or -1 in case of error.

2.1 API implementation

All of the previous function run on the client computer but need to interact with the getaway or on peer. The code of these functions from the Library.

The interface (declaration) of the functions implemented by the library should be exactly as described in this section.

For each function, students should define the various messages that are exchanged between the client and gateway/peers to accomplish the desired functionality. These messages cam be defined as simple C structures that are common between the library and the gateway/peers code

If necessary students can define limits to the length of the various strings (photo_name, keyword, ...)

3 Photos

Photos are inserted in the system and retrieved by the various clients.

For each photo, the systems stores the following:

- identifier – 32 bit integer that Unequivocally identifies that photo (each photo as a unique identifier and one identifier is assigned to only one photo). This identifier is assigned when the photo is inserted.
- Name – each photo as a name. This name is provided by the client when inserting it to the systems. Several photos can have the same name
- keywords – Each photo can be assigned one or more keywords.
- Binary data – the system should store the binary data corresponding to the photo.

All photos (and it information) should be replicated between all peers.

4 gateway

The gateway is the components of the system that is contacted by the client. In this case the gateway provides the client with the identification (Ip address and port) of a peer. When deciding what peer to select, the Gateway can use a round-robin approach.

The gateway should implement a linked list that contains the identification of all active peers.

If one peer stops working the gateway should remove it from the list. Students should define a way for the gateway to be notified of this.

5 Peers

Peers are the components of the system that store the photos information. Peers reply to the various requests from the clients with respect to storing, deleting, updating and querying photos.

Each peer will need to store a linked list with the information of the photos (stored in memory) along with the actual photo file (stored in disk).

It is not necessary to persistently store the information of the linked list. When a peer restarts he should retrieve from other peers all the information.

When a peer receives a request for adding/deleting/updating a photo, that peer should forward that information to all other the peers. Student should define how this replication steps are performed.

6 Client

Students should implement a simple client that allows the validation of implemented API, gateway and peers.

The client only needs to know the address of the gateway, call the gallery_connect function. From this moment on, all interaction are with one peer. All photos (to be sent to the systems or retried from it) should be stored in the directory from where the client was executed.

7 Socket communication

Al communication between client/gateway and peers should be done using AF_INET sockets.

The communication between the client and the gateway should be performed using SOCK_DGRAM sockets

The communication with the peers (from the client or other peers) should be performed using SOCK_STREAM sockets.

It may be necessary to define other communication channels (for detecting that peers stooped executing and for replication). Depending on the nature of the information, these sockets can be SOCK_DGRAM or SOCK_STREAM.

8 Multi-threading

Since the gateway and peers should receive information from multiple sockets and execute different tasks at the same time, these processes should be multi-threaded. For instance:

- the gateway should have one thread that receives requests from the clients and another that receives information from the peers
- The peers should have one thread for each connected client

The students can decide to create more threads to guarantee that the systems dos no block and works as efficient as possible.

8.1 Synchronization and Shared Data

Since multiple threads in one process will access the various shared data structures (for instance peers list in the gateway) it is necessary to guarantee that those data-structures are guarded.

Students should define the suitable synchronization variables and use it to guarantee that the multiple peer have the data consistent between them.

9 Evaluation and dates

The date for the conclusion and delivery of the project will be on June 2th. The presentation of the projects will be done after the delivery of the in a date to arrange with the teaching staff.

The evaluation criteria will be presented in a new version of this document.