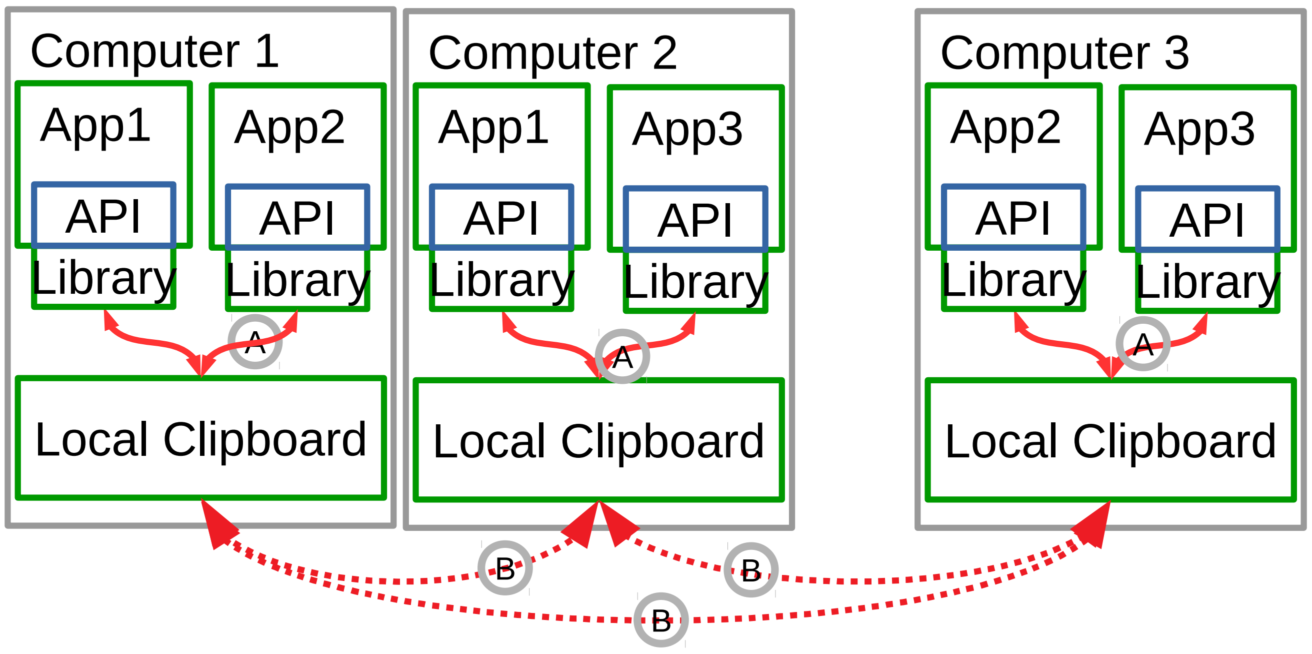
System Programming (MEEC/MEAer)

Project Assignment 2016/2017

In this project the students will implement a simple **distributed clipboard.**

Applications can copy and past to/from the **distributed clipboard** using a predefined API. Students will implement a set of components that allow other programmers to use the **distributed clipboard**. The **distributed clipboard** will run on reversal machines, where data copied to one computer will be available to be pasted on another computer.

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



1 Architecture

The components of the system are:

- **API** — Set of interfaces that allow programmers to develop applications that use the **distributed clipboard**.
- **Library** — Implementation of the API, with the code for the application to interact with the clipboard.
- **Local Clipboard** — component responsible for receiving connections from the local applications and the clipboards running on different computers (remote clipboards). This component receives the commands from the applications, stores the data and replicate it to other remote clipboards.

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

- Simple applications that exercise the **distributed clipboard**.
- A Library composed of a set of .c files that implement the API

- The **local clipboard** process that can be launched in multiple computers

1.1 Overall functionality

When an application starts, it first needs to connect to the **local clipboard**, from that moment on the application can issued copies and pastes. If the **local clipboard** is not running, the initialization should return an error.

The **local clipboard** can be started in single mode or connected to another one. In the case that the clipboard is connected to another, all copy commands on either clipboard should be replicated to the other. The organization of the various clipboard will be presented in Section 3.2.

The **distributed clipboard** holds 10 different **regions** that can be accessed independently. Whenever a values is copied to a **region**, the old value in that region is deleted and this new value is also replicated to all the other remote clipboards.

2 API

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

---

int clipboard_connect(char * clipboard_dir)

This function is called by the application before interacting with the **distributed clipboard**.

Arguments:

- **clipboard_dir** – this argument corresponds to the directory where the **local clipboard** was launched (see section 3.1.1)

The function return -1 if the **local clipboard** can not be accessed and a positive value in case of success. The returned value will be used in all other functions as **clipboard_id**.

---

int clipboard_copy(int clipboard_id, int region, void *buf, size_t count)

This function copies the data pointed by **buf** to a region on the **local clipboard**.

Arguments:

- **clipboard_id** – this argument corresponds to the value returned by **clipboard_connec**t
- **region** – This argument corresponds to the identification of the region the user wants to copy the data to. This should be a value between 0 and 9.
- **buf** – pointer to the data that is to be copied to the **local clipboard**
- **count** – the length of the data pointed by **buf**.

This function returns positive integer corresponding to the number of bytes copied or 0 in case of error (invalid **clipboard_id**, invalid **region** or **local clipboard** unavailable).

int clipboard_paste(int clipboard_id, int region, void *buf, size_t count)

This function copies from the system to the application the data in a certain region. The copied data is stored in the memory pointed by **buf** up to a length of count.

Arguments:

- **clipboard_id** – this argument corresponds to the value returned by **clipboard_connec**t
- **region** – This argument corresponds to the identification of the region the user wants to paste data from. This should be a value between 0 and 9.
- **buf** – pointer to the data where the data is to be copied to
- **count** – the length of memory region pointed by **buf**.

This function returns a positive integer corresponding to the number of bytes copied or 0 in case of error (invalid **clipboard_id**, invalid **region** or **local clipboard** unavailable).

int clipboard_wait(int clipboard_id, int region, void *buf, size_t count)

This function waits for a change on a certain region (new copy), and when it happens the new data in that region is copied to memory pointed by **buf**. The copied data is stored in the memory pointed by **buf** up to a length of count.

Arguments:

- **clipboard_id** – this argument corresponds to the value returned by **clipboard_connec**t
- **region** – This argument corresponds to the identification of the region the user wants to wait for. This should be a value between 0 and 9.
- **buf** – pointer to the data where the data is to be copied to
- **count** – the length of memory region pointed by **buf**.

This function returns a positive integer corresponding to the number of bytes copied or 0 in case of error (invalid **clipboard_id**, invalid **region** or **local clipboard** unavailable).

void clipboard_close(int clipboard_id)

This function closes the connection between the application and the local clipboard.

Arguments:

- **clipboard_id** – this argument corresponds to the value returned by **clipboard_connec**t

2.1 API implementation

All of the previous function run on the application that needs to interact with a **local clipboard** that runs on a different process. The code of these functions form 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 application and clipboard to accomplish the desired functionality. These messages can be defined as simple C structures that should be known by the **Library** and the local Clipboard.

Since this API implements local communication (all processes run on the same computer) several option exist for the communication channel: FIFOs, sockets or even shared memory. Student should define the best approach.

The Library should be composed of a **.h** file (called **clipboard.h**) and a set of **.c** files.

3 Local Clipboard

The **distributed clipboard** system is composed of several local clipboard processes that execute in different computers and that exchange information between them to maintain the various clipboard regions synchronized.

3.1 Startup

At startup it is possible for the user to define if that clipboard is to run in single mode or in connected mode.

If started in connected mode, the clipboard will begin by contacting a preexisting clipboard to register himself and send/receive updates.

In order to launch the clipboard in connected mode it is necessary for the user to use the command line argument **-c** followed by the address and port of another clipboard. Example:

- **`clipboard -c 146.193.41.12 1337`** – 146.193.41.12 corresponds to the address of the host where the remote clipboard is running and 1337 corresponds to the port.

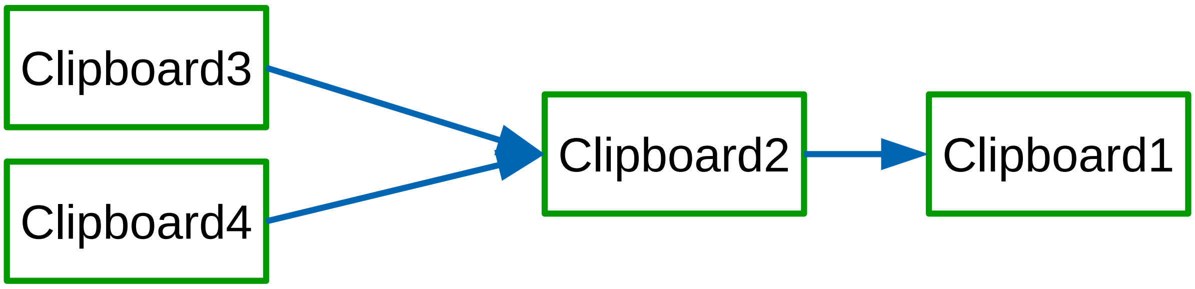
Although a clipboard can work in single or connected mode, it should be able to always receive connections from the local applications or remote clipboards.

3.1.1 Local connection

Local connection to a clipboard (done by applications) should be performed using UNIX domain sockets. The name of the socket is **CLIPBOARD_SOCKET** and will be located in the directory where the clipboard was executed.

3.1.2 Remote connections

Every clipboard (independently of acting in single or connected mode) should be able to receive connections from other clipboards. This connections and communication is performed using INTERNET domain sockets. The port of this socket should be printed on the screen, so that other users use it in the command line (Section 3.1)



In the previous example:

- Clipboard1 was created as single
- Clipborad2 is in connected mode (connected to Clipboard1)
- Clipborad3 and Clipboard4 are in connected mode (connected to Clipboard2)

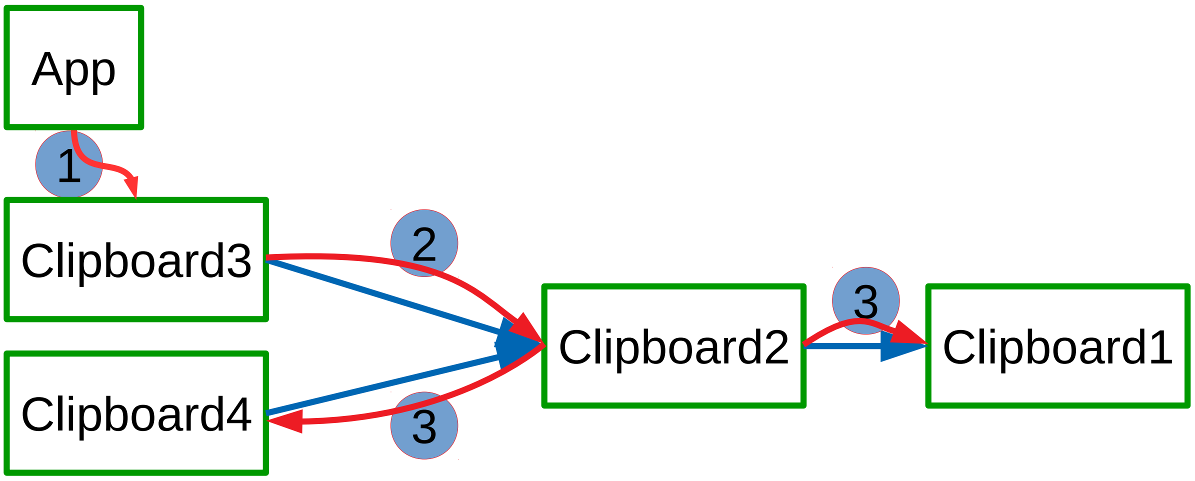
3.2 Regions and data processing

The total number regions (stored data in the **Distributed Clipboard**) is 10, meaning that at most the system can hold 10 different values.

Every time a value is copied/pasted to the local clipboard, the application should state on what region that data will be stored (functions **clipboard_copy**, **clipboard_paste** or **clipboard_wait**.

Every time data is copied to a local clipboard that data should be replicated to all other clipboards. For instance, in the previous example data that is copied to the Clipboard3 should be replicated to the other 3 clipboards:

- Clipboard3 replicated data to Clipboard2
- Clipboard2 Replicates data to Clipboard1
- Clipboard2 replicates data to Clipboard4



4 Client Applications

Students should implement simple client applications that will allow the validation of the implemented API and Clipboards.

The application only needs to know the name of the location of the local clipboard (directory where it was running and call the **clipboard_connect** function). From this moment on, all interaction are with that clipboard.

Students should implement different applications that allow the validation of the various cases, for instance:

- Multiple application copying to the same region
- Multiple applications copying and pasting from the same region
- Multiple application connected to different or to the same clipboard
- …

5 Multi-threading

Since every **local clipboard** should receive requests from multiple applications and remote clipboards and execute different tasks at the same time, these processes should be multi-threaded. For instance:

- Each local clipboard should have one thread that receives connections from local applications
- Each local clipboard should have one thread that receives connections from cooperative remote clipboards
- Each local clipboard should create one thread for every connected applications

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

5.1 Synchronization and Shared Data

Since multiple threads in one process will access the various shared data structures (for instance the regions) 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.

6 Performance

Students should guarantee that the time spent copying and pasting data is the lowest possible. This optimization can be accomplished selecting the best communication mechanisms to transfer data from the application to the local clipboard or reducing the size of the critical regions.

The increase of performance should not be accomplished with incorrect synchronization.

7 Evaluation and dates

Students should finalize and submit the project and report during the week ok 28th of May.

The presentation of the projects will be done after the delivery project of the in a date to arrange with the teaching staff.

7.1 Project submission

The project will be submitted on FENIX. Students should provide a report and an archive of all the code. At the same time students should select one date for the evaluation (link will be provided at the FENIX).

7.2 Report

Students should present a reports explaining the developed system. The report should contain clearly the following information:

- Graphical representation of the architecture and components of the system. Students should expand the figure presented in this report stating what functionality are present on each processes.
- The way thread management is perform should be described. For each process student should list all thread, describe their functionality and explaining how they are created.
- What data structures are shared among threads and how synchronization is performed
- How local communication is performed
- How remote communication is performed
- How data is replicated among the various clipboards

In the final section of the report students should clearly describe if the criteria presented in Section 7.4 are meet.

7.3 Project presentation

For the presentation of the project students should prepare a set of clients and usage steps that allow the teaching staff to evaluate the various functionality.

7.4 Evaluation criteria

The evaluation criteria for project grading will be published in the near future.