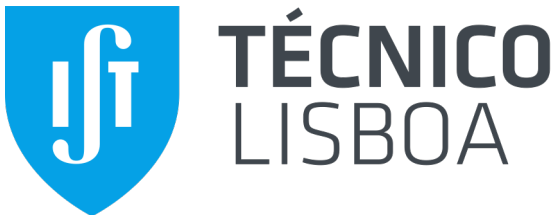
INSTITUTO SUPERIOR TÉCNICO



MASTER'S IN ELECTRICAL AND COMPUTER ENGINEERING

SYSTEM PROGRAMMING
2017/2018 – 2º SEMESTER

DISTRIBUTED CLIPBOARD

Authors:

João Cardoso, nº 84096
e-mail: joao.pedro.cardoso@tecnico.ulisboa.pt

João Sebastião, nº 84087
e-mail: joaofpsebastiao@tecnico.ulisboa.pt

Faculty: Prof. João Silva
Prof. Ricardo Martins

TABLE OF CONTENTS

INTRODUCTION

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

Upon creation, the user is able to define whether the clipboard is to run in single mode or in connected mode. If there are no preexisting clipboards, the first clipboard has to be created in single mode (because there is no other clipboard it can connect to).

After the creation of the first clipboard in single mode, the user is provided with the IP address and the port to which other clipboards can connect to. On the other hand, if there are preexisting clipboards, new clipboards can run in connected mode and connect to one of the preexisting clipboards using their IP address and port.

Each connection is made using INET domain sockets from which clipboards send and receive information. After making the connection, the new clipboard receives all the data that is stored in the clipboard it connected to (which, since connected clipboards are synchronized, is the same in other connected clipboards) and, from this point on, is now able send and receive updates.

Multiple applications can connect to the local clipboard and provide/access information to/in the distributed clipboard which can be used by other applications remotely.

Each time an application copies something, that local clipboard replicates the data to other clipboards so there is Data consistency between clipboards.

ARCHITECTURE AND COMPONENTS

Regarding the architecture of the distributed clipboard, the components are:

- API: Set of functions that allows programmers to develop applications that use the distributed clipboard.

- Library: Implementation of the API, containing the code for the application to interact with the clipboard

- Local Clipboard: Component responsible for receiving connections from the local applications and other clipboards running on different computers through remote connections.

An example of a distributed clipboard is shown in the picture below.

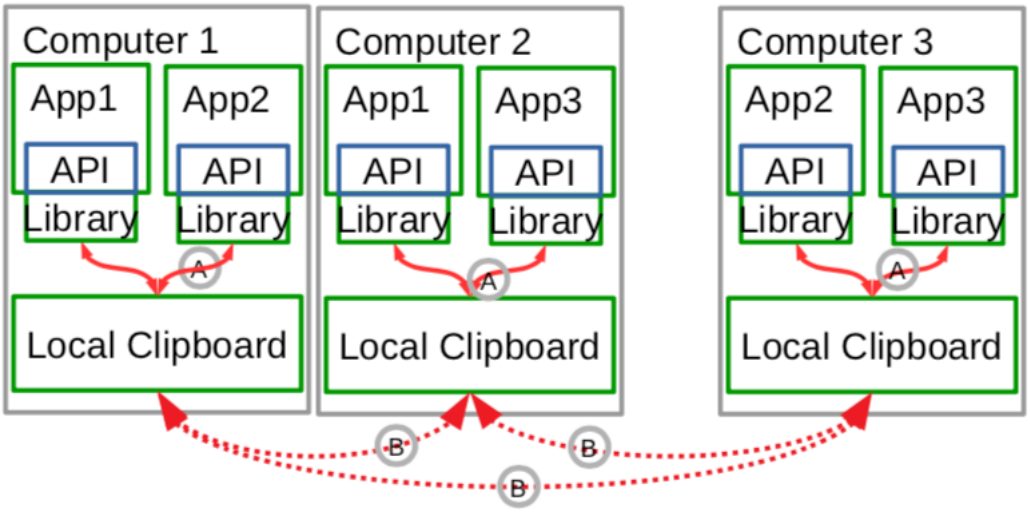


ILLUSTRATION 1: DISTRIBUTED CLIPBOARD INTENDED FUNCTIONALITY

The clipboard is composed of several subcomponents that run simultaneously through the usage of threads. Below are presented a few flowcharts representing some main processes of the program.

The first flowchart is related to the main thread. Since any signals, sent by the operating system or by the user, can affect the program at any point, it is required a function to handle any signal. Both INET and UNIX domain sockets are created, for remote and local communication respectively.

If the clipboard is running in connected mode, data is replicated from the preexisting clipboard to the newly connected clipboard.

Finally, two loops are created, one for INET connections that waits for other clipboards, and one for UNIX connections that waits for local applications.

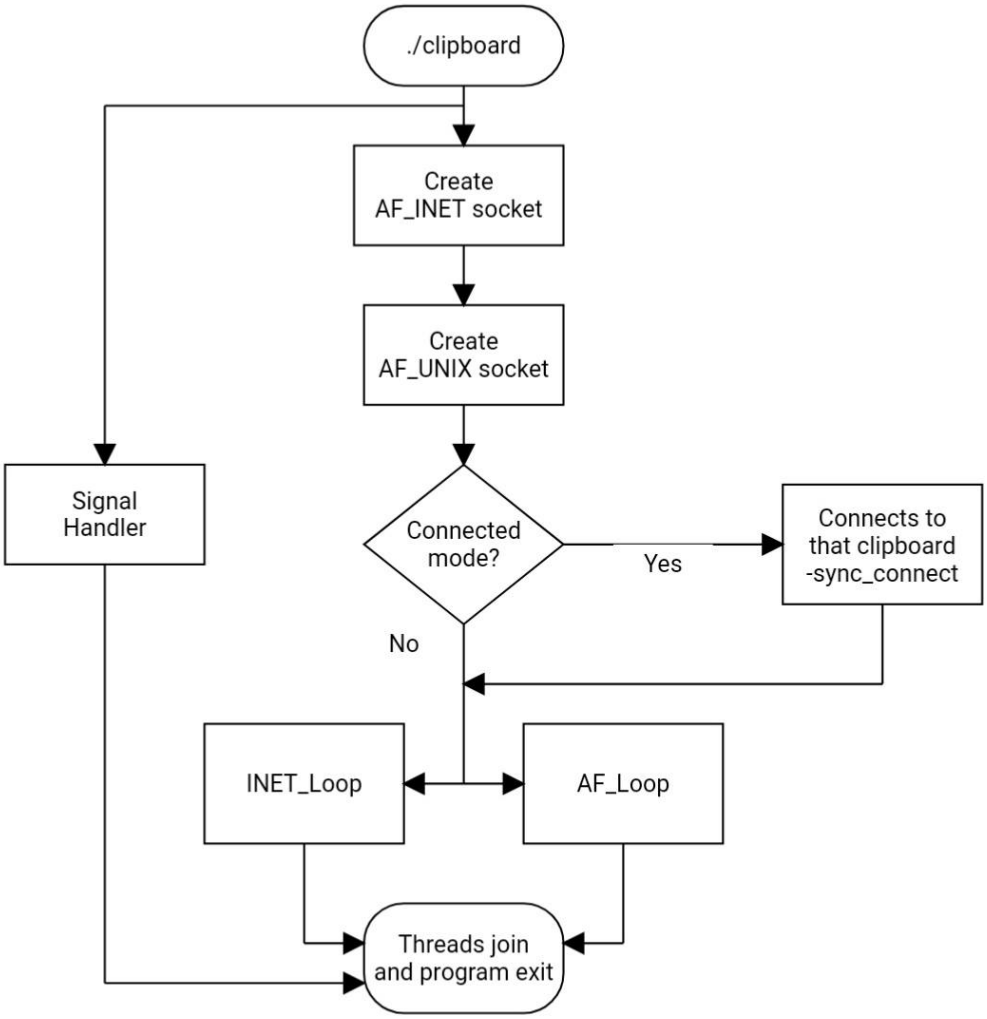


ILLUSTRATION 2: PROGRAM ARCHITECTURE

After accepting a local application or a clipboard, the Loop creates Communication threads. In the case of a connecting clipboard, the data saved in memory is sent to that clipboard and then 2 threads of communication are created: “wait_for_friend” and “wait_for_sync”. The first is about starting the replication and the second is to continue the replication that was started.

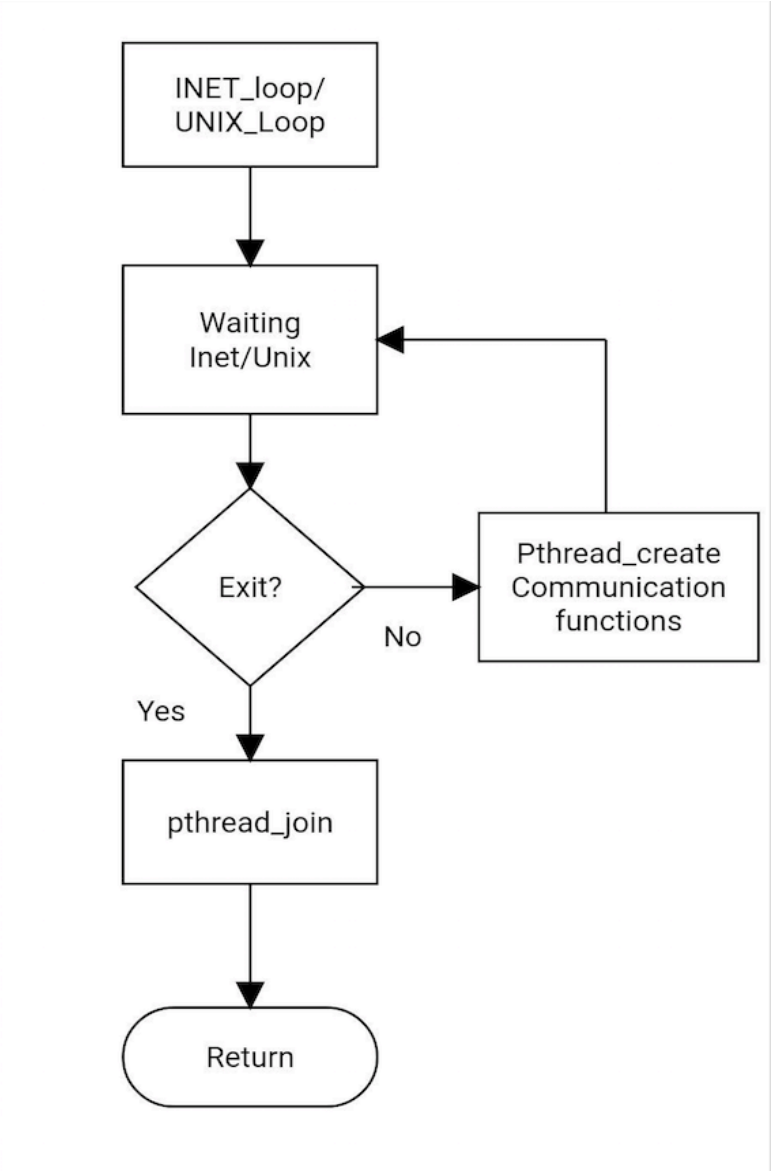


ILLUSTRATION 3: THE LOOPS FOR INET/UNIX CONECTIONS

LOCAL COMMUNICATION

Local communication between applications and clipboards is done by UNIX domain sockets. After the creation of the socket, a loop is created that only stops until a signal is sent to the program to terminate it.

The communication is made using a structure called message which sends the request information such as the operation (Copy, Wait, Paste) and data parameters (region, size).

For each application, the clipboard creates one thread that handles the communication.

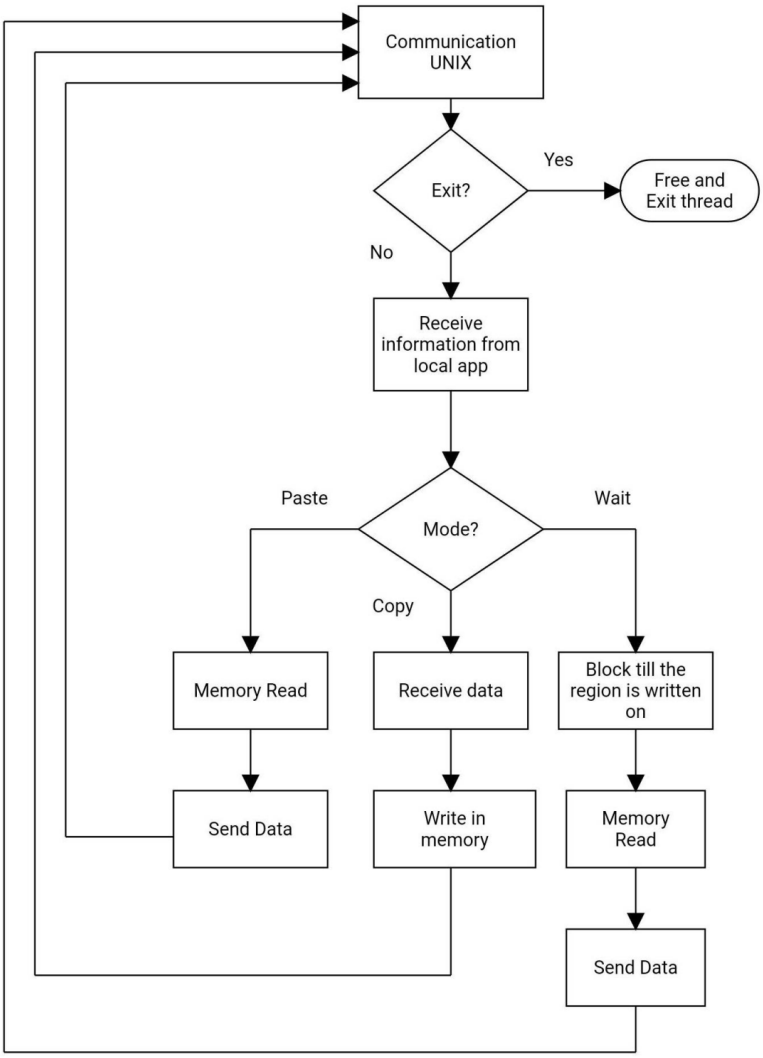


ILLUSTRATION 4: LOCAL COMMUNICATION THREAD

REMOTE COMMUNICATION

Like local communication, while no signal is caught, the loop remains running, waiting for new clipboards to connect.

Remote communication is performed through INET domain sockets. Messages are sent through the sockets in order to replicate data between clipboards.

Like in the local communication, in order to assure no information is lost and the correct memory is allocated, the message is divided into two parts, the message structure which contains details about the data and the data itself. In this case, the mode part of the structure is used as a control variable.

DATA REPLICATION

The final flowchart is related to the replication of data among various clipboards.

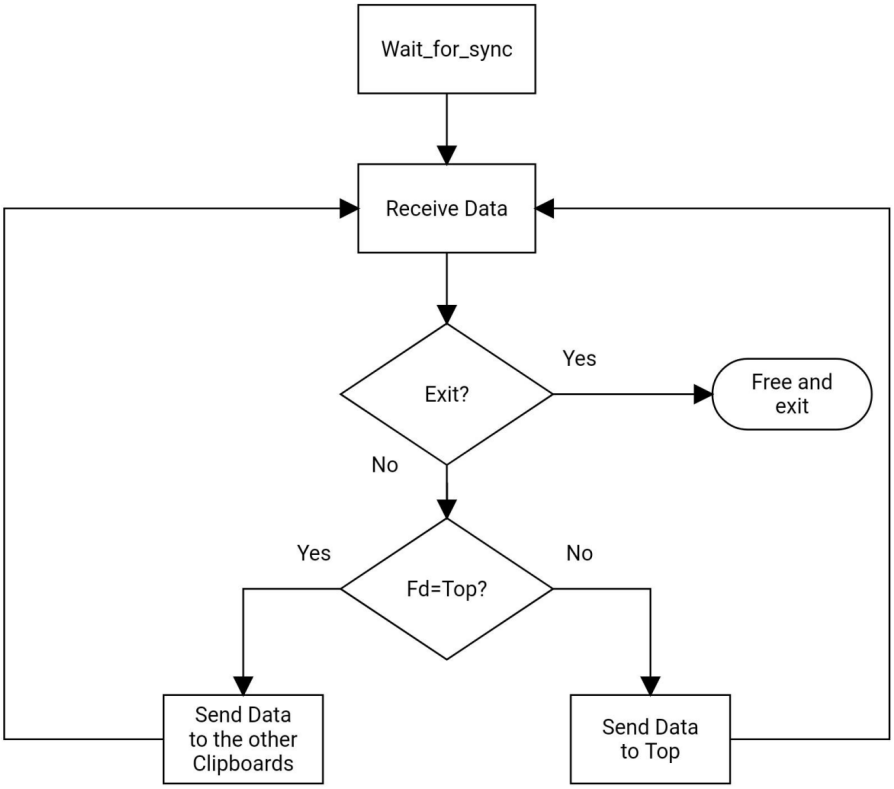


ILLUSTRATION 5: WAIT_FOR_SYNC

Since connected clipboards have to keep synchronism, once one clipboard receives information from an app, it should replicate that information to all the clipboards connected to it, so they can save that information in the same region.

When a connection is made, the clipboard received the connection is now considered the “top” of the now connected clipboard. Therefore, unless the clipboard was created in single mode, every clipboard in connected mode has one “top”.

The only exception to this is if a clipboard is disconnected, so every clipboard has their non-top “wait_for_friend” threads blocked in case this becomes a reality.

If a clipboard in connected more receives new information, that clipboard will send that information to his “top” and so on until it reaches the top of the tree, i.e. the clipboard that is not connected to anyone (created in single mode or the clipboard it was connected to disappeared).

After reaching the top of the tree, the information is sent down to every connected clipboard and stored in memory in the same region as the clipboard that first received the update.

SYNCHRONIZATION

The program only has one critical region that was implemented in the memory function called “memoryf”. That’s the function that every other function uses to either read or write data without having conflicts.

Since it’s used by several functions, threads that communicate with local applications or other clipboards can’t access the same space at the same time.

The functions that access this critical region are:

- “Communication_with_friends” – Local communication threads;

- “wait_for_sync” - Writes the information if it came from the clipboard above him in the tree;

- “wait_for_friend” - Reads the information just written by a local communication thread;

- “send_my_data” - sends all 10 regions of the clipboard to the clipboard that just connected to this one;

- “receive_friend_data” – receives all 10 regions of the clipboard that it connected to.

“Memoryf” has several modes:

- WRITE

- READ

- DELETE

- WRITE_SYNC

WRITE and WRITE_SYNC diverge as the former triggers the thread(s) of “wait_for_friend”. Both trigger to stop the wait by local applications if applicable.

It was also used pthread_cond_wait for the WAIT function of the clipboard and to trigger the replication (memoryf broadcasts to these 2 conditional variables).

THREAD MANAGEMENT AND DATA STRUCTURES

Below are listed the several threads created, the variables and data structures used and how they exit.

Signal handling:

- pthread_create(&sig_thr_id, NULL, signalHandling, (void *)&signal_mask)

Thread that invokes the function "signalHandling" that handles all signals received. The variable "sig_thr_id" is the thread identifier and the variable "signal_mask" is the thread argument and it holds the set of signals to be blocked (in this case all signals).

After the thread is created, the thread is joined and when a interruption/termination signal is caught, memory allocated by the several components of the clipboard is freed and the thread exits (which then leads to the termination of the program.

INET main loop:

- pthread_create(&inet, NULL, INET_loop, (void*)&net)

Thread that invokes the function "INET_loop" that waits for incoming connections and creates 2 threads per connection to exchange information from application to clipboard and between clipboards.

The variable "inet" is the thread identifier and the variable "net" is the thread argument and it's a structure of type "ad_in" that holds the INET socket file descriptor and information regarding the socket address. These 2 threads are:

- pthread_create(&sync[clip_clients], NULL, wait_for_sync, (void*)&new_fd)
- pthread_create(&r[clip_clients], NULL, wait_for_friend, (void*)&new_fd)

These threads invoke the functions "wait_for_sync" and "wait_for_friend", whose function is to wait for information from newly accepted connection and to wait for information provided by local applications, respectively. The thread argument for both is the socket file descriptor of the newly accepted connection and the variables "sync" and "r" are arrays of thread identifiers to keep track of the threads each connection uses.

After the main thread is created, when it's time to terminate the program, both "wait_for_sync" and "wait_for_friend" threads, for every connection, are joined, any memory allocated is freed and both threads exit, to which the main thread frees both arrays used by the threads and then exists.

New connection replication:

- pthread_create(&server, NULL, sync_connect, (void*)&argz)

Thread that invokes the function "sync_connect" that connects to another clipboard using the port and IP address and replicates all the information stored to the new clipboard as well as wait for new information from other clipboards. The variable "server" is the thread identifier and the variable "argz" is the thread argument and it's a structure of type "argument", that holds the IP address and port available to receive connections and socket file descriptor of the INET socket created.

A thread is also created that waits for information provided by local applications, and the format is similar to the one mentioned in the INET main loop:

- pthread_create(&re, NULL, wait_for_friend,(void*)&fd)

The variable "re" is the thread identifier and the thread argument "fd" is the INET socket file descriptor to be connected. After the main thread is created, the thread is joined waiting for the termination signal to exit and free any memory allocated, and then the main thread exists.

UNIX main loop:

This loop creates 2 threads:

- pthread_create(&wa,NULL,waiting_UNIX,(void*)&unix1)
- pthread_create(&c[client_counter], NULL, Communicating_With_friends, (void*)&par)

The first thread invokes the function "waiting_UNIX" that is constantly waiting for new applications to connect to the UNIX domain socket. The variable "wa" is the thread identifier and the variable "unix1" is the thread argument and it's a structure of type "ad_un" that holds the UNIX domain socket that receives the connections and information regarding the socket address.

The second thread invokes the function “Communicating_With_friends” that handles the communication with the application by receiving and/or sending information (defined by the operations “copy”, “paste” and “wait”) through the socket created after accepting the connection.

The variable “c” is an array of thread identifiers to keep track of the threads used for every application and the variable “par” is the thread argument and it’s a structure of type “sck” that holds the host’s socket file descriptor, the file descriptor of the created socket and the “ID” of the client.

After the main thread is created, when a termination signal is received, the termination flag is turned ON and the “waiting_UNIX” thread, is canceled and it joins. After being canceled, the UNIX domain sockets for each connection are closed and the remaining communication threads are joined so they can finish and exit, and the thread frees the memory of the array of thread identifiers and corresponding array of UNIX socket file descriptors.

Evaluation Criteria

This implementation of the distributed clipboard meets the specified requirements and the intended design. The program could be more efficient with the use of shared memory instead of UNIX domain sockets for the data, the critical region could be a bit smaller and more and/or better error treatments (small oversights) across the program.