October 11th, 2018

Lab 1: Peer-to-Peer File System

CSE 514 - Computer Networking

Valentin VIÉ

Introduction

The project is divided in two scripts, one for what we call the main server (the tracker) and one script for the clients (peer/source). The main server is essentially just a TCP server capable of handling multiple request at the same time. The client is more complicated because there are three different side: the client to main server side [C2MS], the server to peer side [S2P] and the peer to server side [P2S]. We will detail later which side communicate with each other and we will explain the protocol in later section.

Multiple display options are available at the very top of the file client.py, for example VERBOSE_C2MS set to 1 will display every message the client is having with the main server side. VERY_VERBOSE_P2S will display every message (mostly chunk request) between a peer and a source.

Protocol options are available at the top of the two files server.py and client.py. You can for example change the frequency at which a client requests the file locations to the main server (UPDATE_FILE_LOCATION_FREQUENCY), or the name of the temporary directory created to write the chunks of data (TMP_DIRECTORY). Most of the parameter combinations haven't been tried, I advise to keep the initial configuration as it is (especially the buffer size or the IPs).

A video of the demonstration is available at the following address (PSU box, 170Mb):
https://psu.box.com/s/1u798oubnrdkotdu9t50mhf82a16rixi

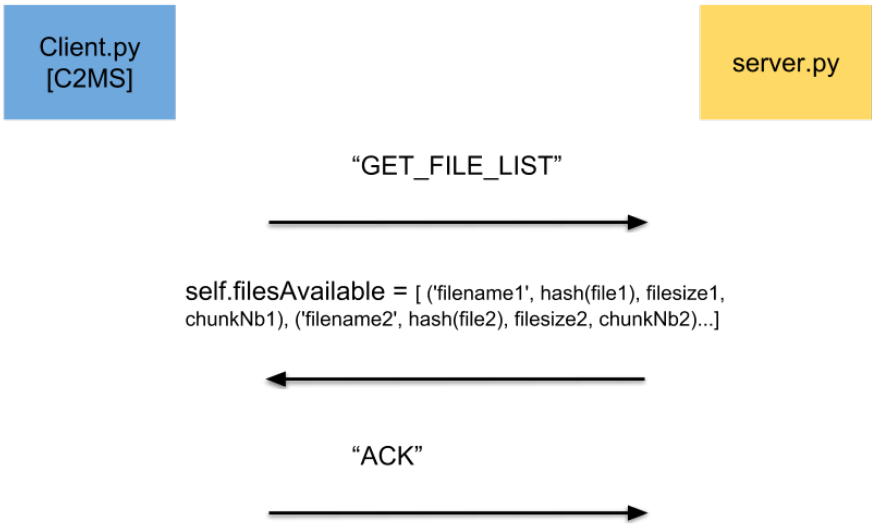
To run the scripts: `python server.py` *for the server,* `python client.py ./sharedFolder` *for the client.*

Client to main server side [C2MS]

The purpose of this side is for the client to be able to request the list of files he can download and which peer has which chunk. In this side the two scripts server.py and client.py [C2MS] interact. The server.py is constantly waiting for a connection, once a client wishes to connect to it the server creates a thread to handle the request. The thread will run until the client disconnects. All requests are handled by the server in `handleClient(clientSocket, addr)`, it selects which function to run between `handleFileListRequest(self, clientSocket, addr)` or `handleFileRegisterRequest(self, clientSocket, addr)` for example.

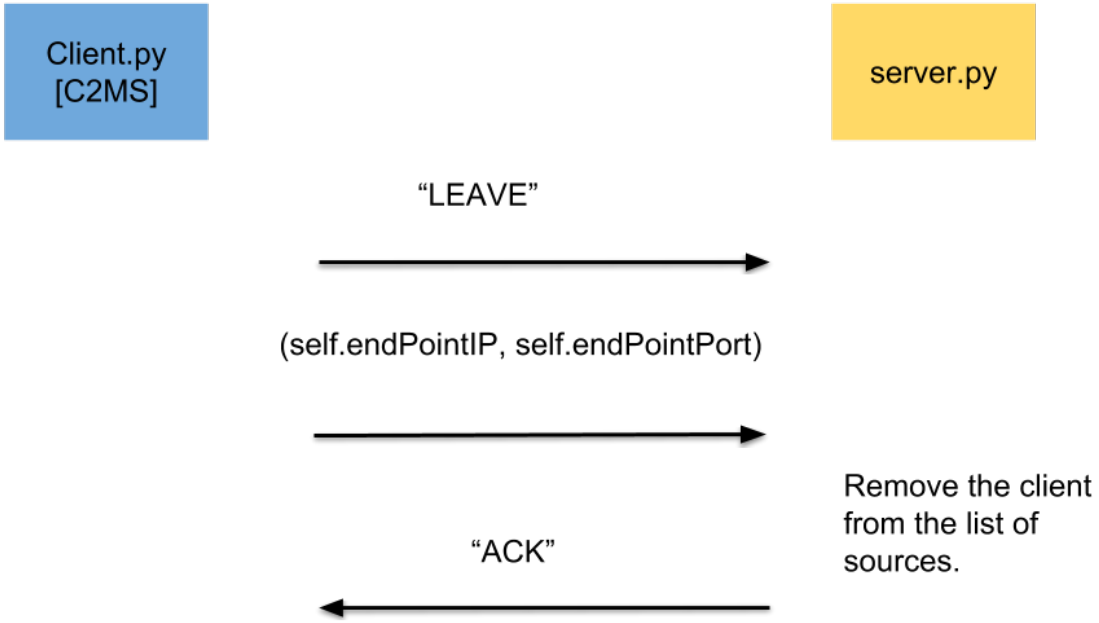
a) handleFileListRequest

This function returns the files available to downloads. It doesn't specify which peer has which chunk.



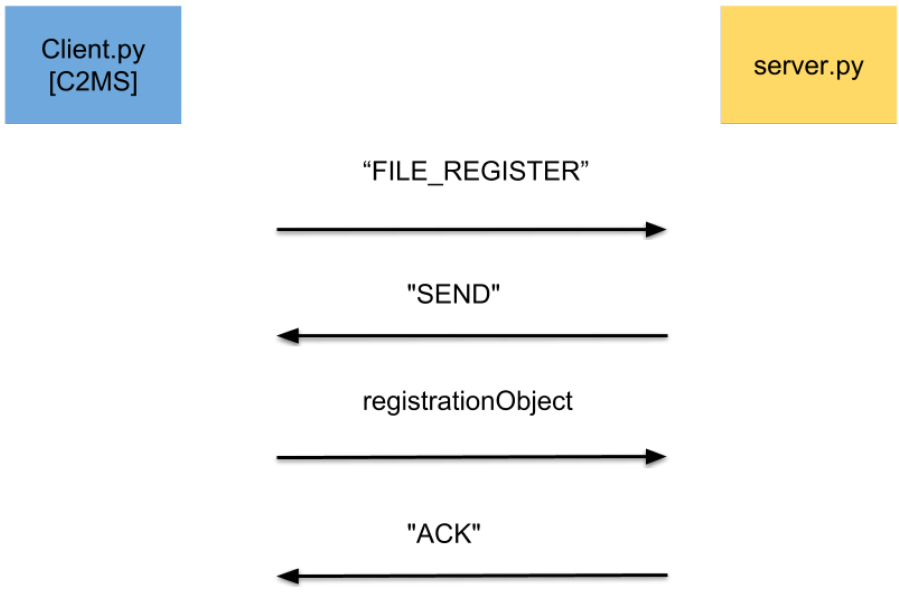
b) handleLeaveRequest

This is the function called when the client wants to disconnect from the server in a clean way. The main server will then remove the client's file from the list of files available.



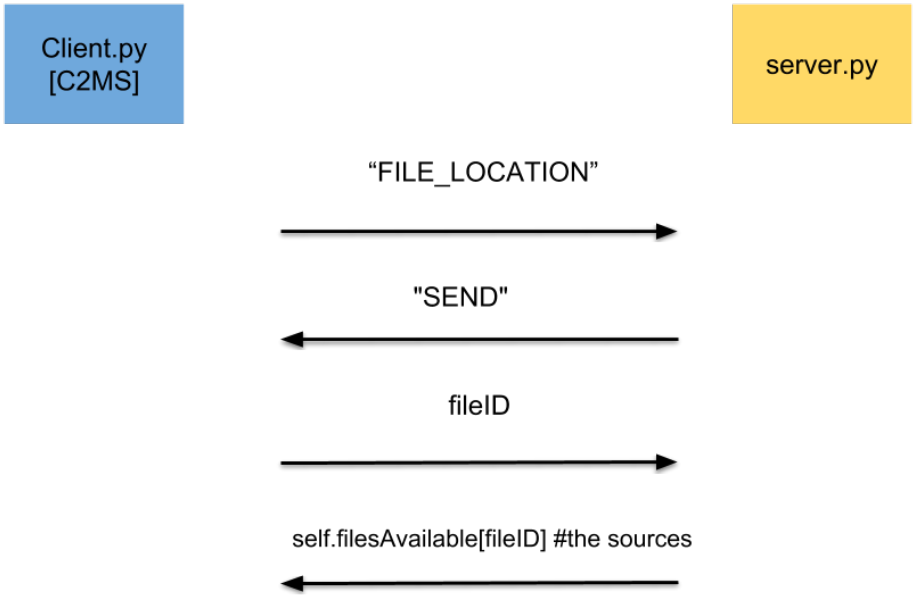
c) handleFileRegisterRequest

This is the method used by the server to handle a file register request. The client first sends a registrationObject which is a dictionary with registrationObject['nbOfFiles'] the number of file shared from the self.shareFolder; registrationObject['endPointIP'] and registrationObject['endPointPort'] the IP and port of the client's server side; registrationObject['filesMetadata'] the metadata about the files. registrationObject['filesMetadata'] contains a list of quadruple [('file1', globalhash of file1, size of file 1, chunkNb), ...].



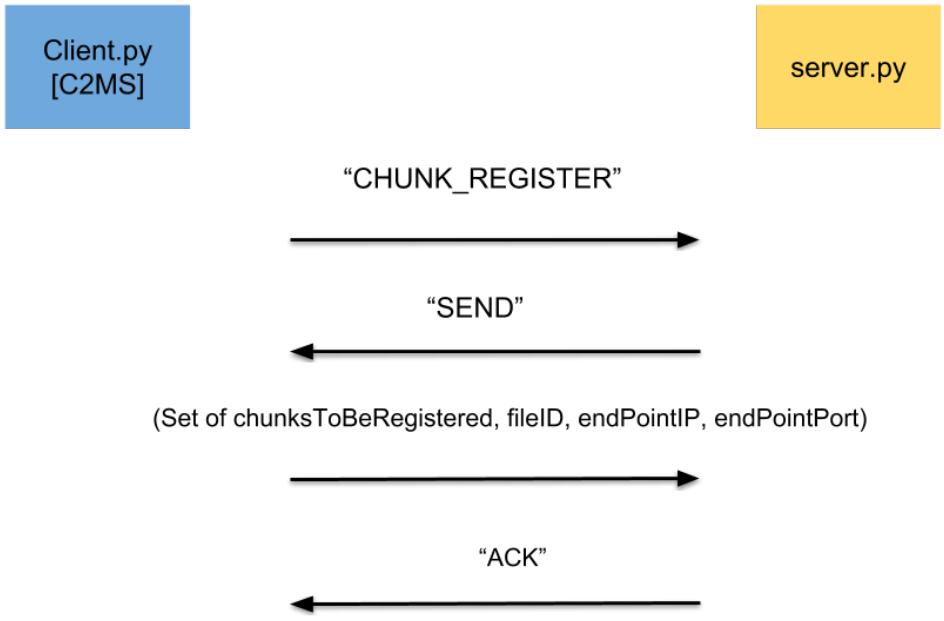
d) handleFileLocationRequest

Sends the file sources to the client. The client needs to specify the fileID which is a quadruple with the filename, the hash of the file, the size of the file and the number of chunks.



e) handleChunkRegisterRequest

The client sends all the chunk he received to the tracker. That way, when another client asks for a fileLocation he has all the new sources available.



Client's server side [S2P] to client's peer side [P2S]

Everything that happens in this section will be used only after a call on the function `startDownload(self)`. This function will launch all the downloading threads to download the chunks from a source. The number of threads is limited by MAX_QUEUED_REQUEST because python limits the number of threads active at a time. `startDownload(self)` also updates the file locations and sends the chunk register requests by exchanging with the tracker (main server).

The client's server side is completely similar to what we did in the script server.py. We just create a thread starting the client's server side and another one to start the client peer side. The thread for the client's server side can create new threads to handle multiple peer requests. The client's sever side can receive two types of requests: GET_CHUNK and LEAVE.

We have to use locks to download a file because the set of chunks to be downloaded, the source's socket and other variables need to be protected. If we don't protect the source's socket we might start a protocol but leave in the middle of it, start a new one and finish the first one later and it causes problems.

Once the download is complete we reassemble the file and we check the global hash.

a) chunkRequest

In this function a peer downloads a chunk of data from a source. The first thing to do is to specify the fileID, the chunkID and the hash. Then once the chunk received we check the hash, write the chunk in the TMP_DIRECTORY and mark the chunk as received and as to be registered.

b) leavePeerRequest

This function is simply a normal way to end the connection between two peers.