Manual

To run the code of the programming assignment, one will need to run at least 2 executables: one for the server and one for each peer. In this document, we will start by describing how to use these two executables.

Some additional programs have been developed to help deploying the server and the peers on different machines. The usage of these programs is explained in the latter section.

All of the source code is included in the folder `p2p`.

1. Main executables

1.1. How to run the indexing server?

The code for the indexing server is contained in the file IndexingServer.py. To run it, one needs to use the command `python IndexingServer.py config.json` where `config.json` is a configuration file in the following format:

```
{
    "listening_ip": "localhost",
    "listening_port": 4000,
    "pool_size": 5
}
```

- `listening_ip` is the public IP of the machine on which the indexing server is running. This IP address will be the one used by the peers to connect to the indexing server. When running the server and the peers on the same machine, we can use localhost.
- `listening_port` is the port on which the server is listening for peers to connect.
- `pool_size` is the maximal number of simultaneous connections that the server can handle.

Once running, the server does not accept interaction with the user. Its behavior is described in detail in DesignDocumentation.pdf.

1.2. How to run a peer?

Similarly to the indexing server, the peer's code can be found in Peer.py. To start it, one needs to use the command `python Peer.py config.json` where the configuration file is in the following format:

```
{
    "listening_ip": "localhost",
    "listening_port": 5000,
    "idxserv_ip": "localhost",
    "idxserv_port": 4000,
    "pool_size": 10,
    "files_regex": "files/u/",
    "download_dir": "files/d"
}
```

- `listening_ip` is the public IP of the machine on which the peer is running. It will be used by the other peers to initiate the file transfer.

- `listening_port` is the port on which the peer is listening for other peers to connect.
- `idxserv_ip` is the public IP of the indexing server.
- `idxserv_port` is the indexing server listening port.
- `pool_size` is the maximum number of other peers that can be simultaneously connected to this peer.
- `files_regex` is a regular expression that defines the files to register for this peer.
- `download_dir` is the directory when the files downloaded from other peers will be stored.

Once the peer is running, the user can interact with it and give it command through a command line interface. The following actions are possible:

- `exit` deregisters the files from the indexing server (IS), terminates the connection with it, and exit the peer interface.
- `lookup <filename>` requests the IS for the lists of other peers that have the file and give the choice to the user to download the file from the available peers.
- `search <filename>` requests the IS for the lists of other peers having that file. It is different from `lookup` because the user cannot download the file with this command.
- `register` registers the peer's files to the IS.
- `list` lists all the files indexed by the IS.
- `help` displays the help screen.
- `getid` returns the peer's id.
- `echo <msg>` send any message to the IS, wait for the IS to answer the same exact and display it. This function can be used to test the connectivity with the IS.
- `benchmark <lookup|search|register> <n>` runs `<n>` `lookup`, `search`, or `register` requests sequentially. It computes the total time of the `<n>` requests in seconds as well as the average time of one request in milliseconds and display it.

2. Additional executables

In addition to the peer and the indexing server executables, one can use the two following scripts to create files with random content and generate configuration files for the peer and the indexing server.

2.1. Create files for the peers

In order to benchmark our system, we need to create files that will be transfered from one peer to another. To do that quickly, one can use a shell script located in `p2p/files` as follow:

```
./create_files.sh <id> <out_folder>
```

One call to this script creates 10 different files filled with random content. There are 2 files of size 1KB, 2KB, 4KB, 8KB, and 16KB. The `<id>` parameter is a way to differentiate the files that are registered to the IS. Indeed, each peer is going to use this script to generate its file and the IS uses the filename as a file identifier. Therefore, the IS needs a way to differentiate the files from one peer to another and this is the purpose of `<id>`. The `<out_folder>` parameter is the directory where the created files will be stored.

2.2. Create a configuration file for the server and the peer

To simplify the task of creating the configuration file, one can use the script `gen_config.py` as follow:

```
./gen_config.py server [listening_port=P] [pool_size=S]
./gen_config.py peer server_ip server_port [listening_port=P] [pool_size=S] [files_regex=R] [download_dir=D
```

The listening IP address is automatically computed. The other parameters are matching with the parameters described in section 1.