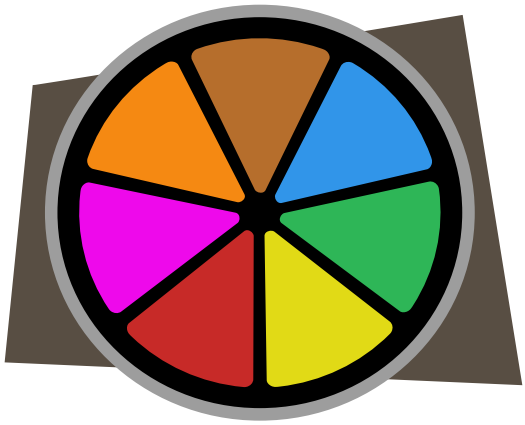
Trivial Torrent Assignment (Project 1)

Xarxes, Grau d'Enginyeria Informàtica



Document version: 1.0
Date: Jan. 2020

Universitat Autònoma de Barcelona

Departament d'Enginyeria de la Informació i de les Comunicacions

Document Information

Document identifier: trivial_torrent_assignment.md
Title: Trivial Torrent Assignment (Project 1)
Number of pages: 4
Date: Jan. 2020
Author(s): Ian Blanes

Change Control

Revision	Date	Changes
1.0	Jan. 2020	(initial version)

Contents

1 Introduction

In this project you will create client and server applications for the file-sharing Trivial Torrent protocol, which is a higly simplified version of the bittorrent protocol.

Main considerations:

1. **Difficulty.** Writing correct network code is *very* difficult. Two preliminary tasks will be given to pave the way to the torrent applications. These assignments may be completed without understanding them; however, not understanding them, will likely result in failure to understand the whole project.
2. **Quality.** Network code that works *most* of the time is worthless. Network function calls are prone to returning errors. Return codes shall *always* be checked, and acted accordingly. If an error code is not accounted for, your code is not correct. Code shall be properly documented.
3. **Standards compliance.** Network code needs to be compliant to POSIX sockets API. Do not rely on how a specific implementation works. Write code that is compliant with what the standard specifies.

2 Tasks

For this project, you will need to complete the following tasks over a period of nine weeks.

2.1 T1: Preliminary Questions

Familiarize yourself with the **POSIX sockets API** by completing the *Preliminary Questions assignment* (see the `Preliminary Questions` document). Provide accurate responses.

Note: it is recommended to complete the preliminary questions up to section 4.5, then complete task T2, and then complete the remaining questions.

2.2 T2: Pong Server

Before moving to the Trivial Torrent implementation itself. You need to do a very small project that it is not related to the main project, but that will help you to familiarize yourself with the POSIX sockets API and the build environment.

Write a UDP server employing the POSIX sockets API that waits for datagrams to arrive and, for each one received, responds with another datagram addressed to the original sender and the data contents of the original datagram. The server must reply to 3 datagrams and then quit.

Use the file `src/pong.c`.

Use the logging API in `src/logger.h`.

Use the POSIX specification in `https://pubs.opengroup.org/onlinepubs/9699919799/`.

The server can be tested by employing the command line tool `netcat` (see `man netcat`).

```
$ netcat localhost 8080
```

2.3 T3: Trivial Torrent Client

Implement the client part of the Trivial Torrent protocol. See Trivial Torrent protocol specification document for details (see section 2.3.1).

Use the file `src/ttorrent.c`.

The `ttorrent` command shall work as follows:

```
$ bin/ttorrent file.metainfo
```

2.4 T4: Trivial Torrent Server

Implement the server part of the Trivial Torrent protocol. See Trivial Torrent protocol specification document for details (see section 2.3.2).

Use the file `src/ttorrent.c`.

The `ttorrent` command shall be extended to work as follows:

```
$ bin/ttorrent -l 8080 file.metainfo
```

2.4.1 Multi-client implementation strategies

The server part can be implemented using two different strategies: employing a *forking server*, or employing *non-blocking sockets*. Employing non-blocking sockets can be *very* challenging. Follow one of the two approaches based on your personal preferences. **Follow the conservative approach if you are not sure which approach is best for you.**

Approach	Description
Conservative	1. Implement the server employing a forking server model (mandatory; 70% grade). 2. Upgrade the server to employ now non-blocking sockets (optional; 30% grade).
Audacious	Implement *directly* a server employing non-blocking sockets (100% grade).

Note: servers employing non-blocking sockets must employ a single process and must not employ threads.

2.5 T5: Trivial Torrent quality assurance

Ensure the quality of the Trivial Torrent applications you have developed. This includes testing them exhaustively, and writing a quality assurance report describing which tests have been performed, the errors encountered, and how they have been solved.

Use the provided template document.

3 General considerations

3.1 Grades

This assignment grade weights 75% in the final lab grade.

Attendance is mandatory to all lab sessions. Missing more than two sessions, during the whole course and without justification, results in a failing grade. A late arrival to a lab session, qualifies as an exclusion motive for that session.

Without exception, plagiarism results in severe disciplinary actions.

3.2 Workspace

Use Debian stable. The gedit and anjuta text editors are available.

3.3 Dependencies

Employ the following command to install missing library dependencies on your personal machine. This step is not necessary, nor possible, in the lab machines.

```
$ sudo apt install libssl libssl-dev build-essential
```

3.4 Building your project

Employ the following command to build your project.

```
$ cd project_folder/
$ make
```

This builds two binaries: `pong` and `ttorrent`.

```
$ bin/pong
$ bin/ttorrent
Trivial Torrent (build Jan 1 2000 00:00:00) by J. DOE and J. DOE
```

Employ the following command to clean your project.

```
$ make clean
```

3.5 Remote work

Machines from `deic-dc14.uab.cat` to `deic-dc26.uab.cat` are available for remote connection. Use `ssh` as follows (replace `USERNAME` by your username).

```
$ ssh USERNAME@deic-dc24.uab.cat
USERNAME@deic-dc-10's password:
USERNAME@deic-dc-10:~$ run_your_commands_here
```

Use `scp` for file transfer as follows.

```
$ scp USUARI_XARXES@deic-dc10.uab.cat:Desktop/file.c .
```

Use `passwd` to change your password.

```
$ passwd
Changing password for USERNAME.
Current password: ...
```

3.6 Debuging tips

- Consider extensively employing the functions in `logger.h`.
- Consider employing `gdb`.
- Consider employing `valgrind` to debug memory problems.

```
$ valgrind bin/pong
```

- Consider employing `strace` to debug system calls.

```
$ strace -e network bin/pong
$ strace -e socket,read bin/pong
```

- Consider employing `libfiu` to test against unexpected failures (`fiu-run`).