Assignment 2: BitTorrent Part 2

***Due date:** Monday, 17 March 2025 at 22:00*

This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own. In particular, do not use automated code-generation tools such as ChatGPT or GitHub copilot.

This is a continuation of the first assignment in which you are required to implement a BitTorrent client[1] using the C programming language. To complete this assignment, you must have a working solution for the first assignment. In particular, for this second part, your solution must (1) compute the `info_hash` value, (2) validate existing pieces from a partially downloaded file, (3) accept and handle connections from other peers, (4) complete a peer handshake, and (5) keep a connection with the tracker. In practice, as in the first assignment, you have to provide implementations for a number of functions that are already declared for you in a template source package available on iCorsi.

Given a torrent file, the value of `info_hash` is the SHA1 value of the bencoded version of the `info` value. Therefore, you will have to extend the bencode functionality from the previous assignment to also implement the encoding. You should compute the `info_hash` in the `metainfo_file_read` function. After you decode and validate the torrent file, you must re-encode the `info` value again to compute the SHA1 hash. To compute the SHA1 hash of some data, you can use OpenSSL.[2] We provide some introductory notes on OpenSSL on iCorsi.[3]

You must define a `struct client` data type to keep track of the status of the download of a given file. When instantiating a `struct client`, the `client_new` function must check whether the file listed in the torrent file is available locally, perhaps from a previous incomplete download. In that case, the function must iterate over the pieces of that file, and validate their integrity using their SHA1 hash listed in the `pieces` field of the `info` dictionary. Otherwise, if the file does not exist, the function simply creates an empty file. Also, `client_new` must generate the peer ID of the client. The ID of a peer is a 20-byte random string. You can generate random strings using OpenSSL.

You must also implement the `client_peer_listener_start` function. This function starts a TCP server that accepts connections from peers. Note that this function must not block the client. A solution is to use a separate thread to implement this function. Also, the server must accept peer connections from both IPv4 and IPv6 addresses.

A BitTorrent client discovers peers from the tracker. A tracker supports two protocols: the HTTP and the UDP tracker protocol. For this assignment, you must use the HTTP protocol. With this protocol, the client sends periodic GET requests to the tracker to inform the tracker that the client is alive, and also to receive information about other peers. The URL in the GET request should contain, within its parameters, the ID of the peer, the `info_hash`, some statistics about the amount of data downloaded/left/uploaded, the port on which the client accepts peer connections, and the `event` key. The `event` key is optional. Its values are `started`, `completed`, or `stopped`. The value is `started` when the client first contacts the tracker. Periodic announcements do not contain the `event` key. The client then sends a request with `event=completed` upon completion of the download. Finally, when the clients stops downloading, it must also send an announcement to the tracker with parameter `event=stopped`.

The tracker can reply to a client's GET requests in two ways. In case of error, the tracker returns a bencoded dictionary containing the `failure` key. Otherwise, the tracker returns a dictionary containing keys `interval` and `peers`. `interval` specifies a time interval between regular requests. The `peers` key

[1] https://www.bittorrent.org/beps/bep_0003.html
[2] https://docs.openssl.org/master/man3/
[3] https://www.icorsi.ch/mod/url/view.php?id=1193199

specifies a list of dictionaries, one for each peer, each containing the keys `peer id`, `ip`, and `port`, that specify the ID of the peer, its IP address or DNS name, and port number, respectively. In order to send HTTP requests to the tracker, you can use the libcurl API.[4]

Finally, you will have to implement the peer handshake in the `peer_init` function. As a result, upon completion of the handshake, `peer_init` must properly initialize the given `struct peer` object, which represents a connection with the peer. In a peer handshake, the client sends a message containing the byte value 19, the string “BitTorrent protocol”, 8 consecutive zero bytes (value 0), the `info_hash` value, and the client's peer ID. The client must also receive and record the same data from the other peer.

Submission Instructions

You must write your code in the `bencode.c`, `metainfo.c`, `client.c`, `peer.c`, `peer_listener.c`, and `tracker_connection.c` source files provided within the source package available on iCorsi.

Submit those completed files through the iCorsi system. Do not submit other files. Add comments to your code to explain sections of the code that might not be clear. You must also add comments at the beginning of the source file to properly acknowledge any and all external sources of information you may have used, including code, suggestions, and comments from other students. If your implementation has limitations and errors you are aware of (and were unable to fix), then list those as well in the initial comments.

You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file*, such as project description files.

[4]https://curl.se/libcurl/c/