CS118 Project 1

Lawrence Ouyang: 504128219
Louis Truong: 904170046

Note: We had to adjust test script 21.grading.sh because of the script did not give enough time for the tracker to print out its Test Passed messages. Sleeping for 2 seconds before killing the tracker allowed the tracker to populate the test.result file properly.

All functions are located in the client.cpp file. Modularization, unfortunately, could not be achieve with the given time constraints and the way the file was already implemented. However, implementing the program with modularization would have improved both performance and clarity.

We start in main by opening a thread that accepts connections. Following that, we send our initial request, which contains the event status "started" Following that, we spawn another thread that connects to peer lists returned to the tracker. From there, main loops over and over until requested to stop or failed to connect several times.

In the thread that accepts connections, we start by opening a socket, binding, and listening. After that, we wait for a connection, which will trigger accept.

Once a peer has connected to us, the accept manager spawns another thread that handles the connection with the peer. We wait for a handshake, reply with a handshake, wait for a bitfield, reply with a bitfield, wait for an interested, and reply with an unchoke. Once this is done, we wait for requests and haves, while we send pieces and update the bitfield. This is all done on a thread that is detached from main, and we don't accept connections that are connected via the other thread.

In the connection manager thread, we wait scour through the peer list returned by the tracker thread. We run this based on the interval of the torrent. Once we find a peer that is not connected, we add it to a connected list and spawn a thread to connect to it. In this new thread, we run through a similar command as accept manager spawned threads. We send a handshake, expect a handshake, send a bitfield, expect a bitfield, send an interested, expect an unchoke. After that, we loop through our bitfield, finding pieces that are missing. If a piece is missing we look at the connected peer's bitfield and see if they have the piece. Once we confirm they have the piece we send a request and

set the bit in our bitfield for the piece to 1, and once we received the piece we send a have.

We do this until our file is completely downloaded. Once we recognize that our file is finished, that is m_left is zero, we stop looking for peers to connect to and instead just keep a accept connection thread to give connecting peers a chance to download our files, or seeding.

Although our program mostly behaves as intended, there are some hiccups. The threads accept messages sequentially, meaning if there is an unexpected message the program will crash. Other issues involve limited buffer sizes and unexpected delays in recieving pieces.