Ryan Trihernawan

UID 904-063-131

February 15, 15

CS 118 Lab 1 Report

I use multiple threads to implement the bit torrent program. I use pthread specifically to create multiple threads and coordinate them to carry on different purposes. I separate the logic of my program into three components: connection manager, tracker connection handler, and peer connection handler. The connection manager's job is as follows: it first checks if the actual file exists in the current directory. If the file does not exist, it creates a new file and allocates space as big as the expected length of the file as indicated in the torrent file. It then scans the file and checks if the corresponding pieces exist and set the bit in the bitfield appropriately to indicate whether pieces exist or not. Secondly, the connection manager listens to specified port waiting for incoming requests. Before it actually accepts any connections, it spawns a thread to handle tracker connection. This thread allows the program to continue sending requests to tracker without using the socket select function. During the interval, the thread simply sleeps, allowing for more precise request to the tracker. This thread accesses the global variables such as "uploaded", "downloaded", and "left" and send them to the tracker. Other threads update these aforementioned variables after uploads and downloads. Pthread provides an easy and convenient method to lock the variables for atomic writes and reads. Thirdly, the connection manager waits for incoming requests for one second until timeout occurs and checks the metainfo, which is fetched by the thread handling the tracker connection, for peers that the program can connect to if downloads are necessary. If timeout does not occur because there is an incoming request, it accepts the connection and spawn a thread to handle the peer connection. If the connection with the peer already exists, it simply terminates the new connection, suggesting the bi-directional nature of bit torrent protocol connection.

A new thread is spawn for each new peer connection, allowing parallel uploads and downloads. Such performance forces one to handle shared variables with great caution. As mentioned earlier, pthread provides easy and convenient methods to lock theses shared variables. I use "pthread_mutex_lock" and "pthread_mutex_unlock" to lock and unlock shared variables, respectively. Despite using threads to handle multiple connections, I still need to use "select" in the main thread to alternate between listening for incoming requests and checking for new peers in the metainfo. However, since there is only one socket file descriptor that needs to be monitored, the implementation is trivial. I use a timeout of one second to allow "select" to timeout if there are no incoming requests.