SevenDeadlySYNs

6/2/2016

**Design Specification**

1. Input
   1. Client
      * Command Input: ./client\_app [tracker IP address] [synced file system]
      * Example Command Input: ./client\_app 10.41.36.145 ~/dart\_sync
      * [tracker IP address]
      * Requirement: must be parsed by inet\_pton
      * Usage: used to connect to the tracker on a specific standard host
      * [Synced File System]
      * Requirement: must exist
      * Usage: used to sync with the tracker’s master filesystem. The tracker’s files will overwrite the local versions initially.
   2. Tracker
      * Command Input; ./tracker\_app
2. Output
   1. Client
      * Print statements that describe the client’s actions on both network and logic threads
      * File additions and deletions that the client receives from its peers that are written and committed to disk atomically
   2. Tracker
      * Print statements that describe the tracker’s actions on both network and logic threads
3. Data Flow
4. Data Structures
5. Error Conditions Detected and Reported

**File Directory Overview:**

**How Synchronization Is Achieved: "Session Synchronicity"**

We are able to maintain a synchronized local directory by coordinating information exchanges about file updates between each client and the tracker. Each peer can request files from the other peers in the network.

**Overview of Network Operations:**

The client and tracker logic threads are separated from all actual network functions by asynchronous queues that the tracker and client logic threads can use like APIs. These asynchronous queues deliver information received by the network thread to the logic thread and deliver network requests to the network thread to disseminate. By separating the logic from the actual network calls, we can preserve a simple two thread but asynchronous communication model between the logic and the network.

The advantage of these communication queues is that each thread can focus on reacting to incoming data and acting appropriately. The single thread model also prevents Heisenbugs from developing.

**Things that the client logic can do to interact with the network:**

Tracker to Client (incoming)

- Receive a master file system from the tracker

- Receive a master file table from the tracker

- Receive an update to the file system from the tracker (peer changed something)

- Receive a new peer added message / receive a peer deleted message

- Receive a peer acquisition update message

Client to Tracker (outgoing)

- Send a chunk acquisition update

- Send a request for master file system and file table

- Send locally updated files

Client to Client (incoming)

- Receive a chunk request

- Receive a chunk (request fulfilled or rejected)

Client to Client (outgoing)

- Send a chunk request

- Respond to chunk request by sending chunk or rejection response

Client Network Operation:

The network thread continually scans it's set of open socket descriptions with a select() call. After a timeout period, if no data is received on any of the sockets, then all of the queues from the logic thread are checked. (If there is data on the open sockets, then that data is processed before moving into the communication queues.)

The set of open socket connection will include the following connections. If the network thread is servicing a client, the network will always maintain a connection to the tracker. The network will also leave a listening socket open for new peer connections. If there are any existing client connections, those sockets will also be watched.

The client network thread also automatically sends a heartbeat message to the tracker every 6 seconds to assert it's "alive" status.

When a client to client connection is made:

When the client logic makes a chunk request, the logic must specify a peer to request a chunk from. The network thread inspects its record of that peer, and if no connection exists, a connection is opened. A record of the number of requests made on a connection is kept. For every response, that record of requests is decremented until the outstanding requests reach zero, at which point, the originator of the requests will close the client to client connection.

**Things that the tracker logic can do to interact with the network:**

Client to Tracker (incoming)

- Receive file system update

- Receive request for master file system and master file table

- Receive a file acquisition update from a client

- Receive notice that a client has joined or left a network

Tracker to Client (outgoing)

- Distribute a file acquisition update of a client

- Distribute a file system update to clients

- Distribute information about a new peer or lost peer

- Send master file system and master file table

Tracker Network Operation:

The tracker thread scans its set of open socket connections for new data to process and then checks each of the request queues from the tracker logic to the network thread. The tracker keeps an open socket devoted to listening for new client as well as open connection to each existing client in the network.

The tracker network thread keeps the master peer list of clients. This master peer list contains information about each client including: id (the identifier given to the logic side of things), the ip address, the time last alive, and the socket. (The tracker network thread periodically scans all of the open client connections and terminates any client that hasn't sent a heartbeat within a 6 second time frame.)