MP2 Design Document

Client

The client's `processCommand()` and `processTimeline()` are implemented very similiarly to how they were in MP1 (`processTimeline()` is effectively `processChatmode()`), except they will be using gRPC methods and data structures defined in the client stub instead of syscalls.

The `processTimeline()` uses a separate thread to handle the `grpc::ClientReaderWriter` to receive timeline messages and write them to `STDOUT`. In the main thread, we handle the blocking I/O from the `getPostMessage()` and create a `Message` object to pass to the `ClientReaderWriter` instance.

The `connectTo()` function is a wrapper around gRPC's `NewStub()` which also sends a RPC to `Login()` to instantinate the user on the server side after a successful connection.

Server

The server contains an `unordered_map` to map usernames to individual `User` objects.

The `User` object contains `following` and `followers` which contain usernames of users. The `timeline` member is a `std::deque` that contains `Message` objects; the `deque` is a useful data structure as it allows us to easily insert/pop from either end, allowing us to maintain a 20 message timeline without any erase/remove.

Because gRPC maintains a thread pool, any members of the SNS server or User objects can be accessed simulataneously, I use `std::mutex`es and `std::unique_lock`s to maintain atomicity when needed.

Server Timeline Impl

As gRPC handles threads for us, the `Timeline()` enters an infinite loop listening for the next message from the client or until the `ServerContext` is cancelled.

Upon receiving a user message, the function accesses the `User` object associated with the message's `username` field.

If it is a user message and not a magic string (see below), then the function iterates through all users contained `following` vector of the user. In each iteration, we access the follower's user object and `Write()` the message to their timeline stream.

Because of the construction of the `Timeline` RPC, we do not initially know which `ServerReaderWriter` stream is associated with which user. As a result, the client sends a magic string `0xFEE1DEAD` before entering timeline mode. The `0xFEE1DEAD` is a string and not a series of bytes, as gRPC does not like to mix byte literals with strings when expecting a `utf8` encoding. Upon receiving the `0xFEE1DEAD` message, the server adds the stream to the user object, allowing it to be invoked from other threads.

To handle a fringe case (see comments in code), all messages containing `0xFEE1DEAD` are dropped; as the client sends all messages with the newline character, we will not drop any messages from actual users should they decide to send the magic string.

Server Persistence Model

In order to maintain persistence, the server creates a `server.dat` file. This file contains, on separate lines, the usernames of all user that have logged into the server at any point.

The constructor of `SNSServerImpl` reads the usernames from `server.dat` then executes the static `User::from_file()` to create a `shared_ptr` to a user object.

Each user's data is stored in a separate file; the filename is their username followed by the `.usr` extension.

The `.usr` file format stores the `followers`, `following`, and `timeline` objects in different sections; each section is headed by predetermined magic numbers. The file format is flexible in the ordering of these sections.

The `followers` section is headed by `0x1BADFEED` and `following` by `0xC001D00D`, and timeline by roastbeef (`0x120457BEEF`). Usernames are stored on separate lines; each message is split into 3 lines, one each for sender, message content, and timestamp.