**Overview:**

This replication system consists of a single primary, with whom all client front ends communicate, and a fixed number of back up replication managers (RMs). Due to development constraints, this system simulates replication on different machines by representing each RM as a different port number on the local machine.

* The global PORT\_PRIM is the port the primary will reside on
* The global NUM\_BKUPS defines the maximum # of RMs in the system (including the primary)
* The global vector rm\_connfds contains a file descriptor for every active RM connection from primary
* The global bool am\_primary indicates whether a given RM is the primary or not

**1. Program Execution:**

* Each new instance of the server first polls all ports from PORT\_PRIM to PORT\_PRIM+NUM\_BKUPS-1
* The RM then takes the first available port as its own. If this is PORT\_PRIM, it becomes the primary.

**2. Primary Execution:**

* A listening socket is created for PORT\_PRIM.
* Primary tries to establish a connection to every possible RM port
* Primary loops waiting for client requests, and spawns a thread to service each request
* Primary relays the received command from the client to every active RM connection it has

**3. Consistency:**

* Due to the multi-threading, client requests may be executed in different orders on various RMs
* However, the ordering within each RM will be internally consistent.
* If the primary goes down, the user may see its followers list in a different order, but the list will still contain the exact same elements.
* Note: since the client request handling functions make use of the user's ID to locate its files, each RM must update the ID of every received user object to the ID it has on record for that user before executing the relayed command.

**4. Backup RM Execution:**

* The RM connects a socket to the primary, and creates a listening socket.
* If both succeed, it notifies the primary of its existence and its port, and waits for a connection.
* Before waiting, it sets its local FILE\_PATH appropriately to where its files will now be stored, and calls the “cp” command to copy over all of the primary's files.

- In a real world system, would not want this “cp” call to be blocking on the primary

- Also should replace this with a call to “scp”, a custom program, or a built in, secure extension to the existing .cpp

* The primary receives the notification command, and adds the RM to its list of active RMs.
* The RM then simply blocks on a read call and executes every command it receives.
* Replies are only sent if a given RM is in fact the primary, to reduce unnecessary message passing.

**5. Primary Failure:**

* Once the primary goes down, the state of each RM's connection becomes invalid.
* All RMs then attempt to create a socket at PORT\_PRIM, and close their connection to primary.
* Whoever gets there first succeeds, and assumes the primary position.
* All other RMs detect they failed, poll for available ports, and try to set themselves up as RMs.
* If an attempt to obtain a port fails due to race conditions, they loop until they succeed.