A Key-Value-Store Server

An implementation by Harry Mumford-Turner.

Build the server using `make` and run the server with `/build/server 5000 6000`.
This will create a new thread that handles setting up and tearing down the server, we will call this the main thread. The main thread start point is the `initiate_servers` function in `server.c`. This starts two threads that each open a socket, one for the data port, another for the control port, these are our producers.

The main thread also spawns several worker threads, acting as a thread pool, that wait on a queue of work, these are our consumers. The main thread then waits on a death mutex, that when signalled, will tear down both of these threads and cleanup the producers, consumers and queue.

The producers then start to poll for new connections, and when found, pushes them onto the queue and the workers are signalled. The workers will pop items off the queue and handle the connection, by continually polling for any responses from the connected client.

Access to the queue is synchronised using semaphores so only a single consumer will access an item from the queue at a time. The clients can issue tasks that are handled by the `protocol_manager.c` depending on if the client connected to the data port or control port.

After the workers have performed all the client tasks and the client has disconnected, the worker then waits for new items on the queue. When the client interacts with the Key-Value-Store, via the worker, a single Mutex locks access while the command is processed, this prevents data races and helps thread synchronisation.

A test harness, that acts as a client robot, has been written to programmatically send commands to the server. Build the tests using `make tests` and run the tests with `/build/tests 5000 6000`. The test harness will spin up a server and fire lots of commands to test each area of the program. From invalid commands to 100+ clients connected at once.