Jared Craig
4 October 2015

Threaded Server

For my threaded message handler, I used two mutexes and one condition variable to keep the threads in sync. The first mutex is used in the Buffer object to restrain the worker threads until a client is in the queue. The worker threads sleep and wait for a signal telling that a client has arrived. When a client arrives, the client is appended to the queue then sends a signal to the worker threads--waking up a single worker thread and removing the lock. Once awake, the worker reenables the mutex lock--locking out the other workers, and proceeding to grab the first client from the queue.

The second lock is used in the server code to protect and synchronize access to data structures and the code that handles the client's request. The mutex is locked before each thread calls handle, and is unlocked once the thread has completed the request.

```
//----------------------------------------------------------------------
---
void Buffer::append(int c) {
    pthread_mutex_lock(&lock);
    buffer.push_back(c);
    setCache(c, "");
    pthread_cond_signal(&not_empty);
    pthread_mutex_unlock(&lock);
}
//----------------------------------------------------------------------
---
int Buffer::take() {
    pthread_mutex_lock(&lock);
    while (buffer.empty())
       pthread_cond_wait(&not_empty, &lock);
    int client = buffer.front();
    buffer.erase(buffer.begin());
    pthread_mutex_unlock(&lock);
    return client;
}
// HANDLE THREADS(server)
=============================================================
void * threadHandle(void *arg) {
    Server *server = (Server*) arg;
    while (1) {
       pthread_mutex_lock(&server->lock);
       server->handle();
       pthread_mutex_unlock(&server->lock);
    }
    return NULL;
}
```