Yazan Halawa

Lab 2 Report

I created a thread-safe class called buffer which holds the queue of 10 clients and uses mutexes and condition variables whenever it accesses the queue(synchronization). Here is an excerpt of the Buffer Class:

```
void Buffer::append(int client){
    // Lock the critical section
    pthread_mutex_lock(&mutex);

    // Wait till the queue is not full
    while(buff.size() == 10){
      pthread_cond_wait(&not_full, &mutex);
    }
    // Add the new client to the queue
    buff.push(client);

    // Signal that the queue is not empty
    pthread_cond_signal(&not_empty);

    // Unlock the critical section
    pthread_mutex_unlock(&mutex);
}

int Buffer::take(){
    // Lock the critical section
    pthread_mutex_lock(&mutex);

    // Wait till the queue is not empty
    while (buff.empty()){
      pthread_cond_wait(&not_empty, &mutex);
    }

    // Grab the first element in the queue
    int client = buff.front();

    // Remove the client from the queue
    buff.pop();

    // Signal that the queue is not full
    pthread_cond_signal(&not_full);

    // Unlock the critical section
    pthread_mutex_unlock(&mutex);

    // Return the client for the server to handle
    return client;
}
```

So the for both functions I am using the mutex to lock the critical section. Then in the append function, I wait for the signal that the buffer is not full(using the condition variable) and then I add a client to the queue. In the same way, in my take function, I wait for the buffer to be not empty(using the other condition variable) and then I remove a client from the queue.

Furthermore, in my handler class whenever I read or write to the vector of messages that is shared between the threads I use the mutex to lock and unlock and make sure that the critical sections are safe.