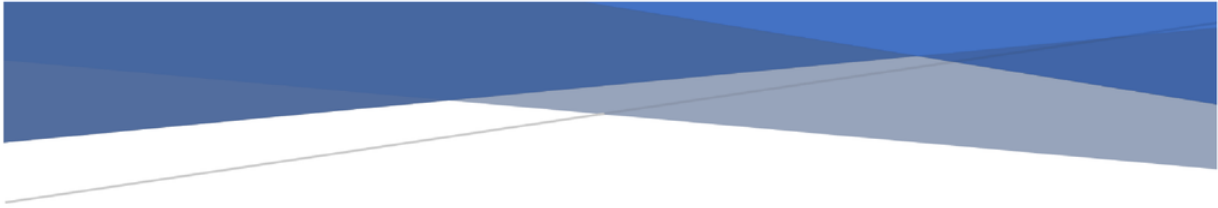


Nick Hinder Assignment

by Nicholas Hinder

FILE	2927_NICHOLAS_HINDER_NICK_HINDER_ASSIGNMENT_113614_100444 8619.DOCX		
TIME SUBMITTED	10-MAY-2018 05:00PM (UTC+0100)	WORD COUNT	1739
SUBMISSION ID	88889950	CHARACTER COUNT	9849



TCP/IP CLIENT SERVER IMPLEMENTATION AND EVALUATION

System Software And Multi User Applications Engineering

nick hinder

Contents

Introduction	2
Connectivity	2
Client Application.....	2
Server Application.....	2
Data Transfer	3
Future Considerations.....	4
References	4

Introduction

This report details the technologies and approaches used in order to create a multiple user, networked, chat room application.

The application will allow users to connect to a server from their client application and communicate with all other users connected to the same server. The client application presents the user with the ability to read messages that have been sent, by themselves or others, and to send new messages.

Connectivity

Connections are established between client applications and the server through use of ARPA INet protocol. This functionality is added to the application through including the `<arpa/inet.h>` header file. The inet protocol grants the ability to communicate via sockets in a structured manner; these connections are set using appropriate addressing, ports and settings. The ports have to be set up correctly in the hton format (host to network), this will ensure endianness has been accounted for and will enable correct working of the connections regardless of cross-system connections.

More modern libraries for this form of connectivity are available; these code files obfuscate the settings of the protocol and allow an easier to code and use interface, whilst retaining the functionality of the protocol itself.

Client Application

The client application makes use of a QT forms display in order to provide a user interface. The form input elements utilise QT signals and slots in order to allow form elements to act as observable elements, granting the ability to call the various methods that will perform server connection activities such as joining the server, leaving the server and adding messages.

The client has a built in timer that emits a signal every second, this signal is connected to the GetMessages slot meaning that there is a regular refresh of the messages that are displayed in the chat window. The timer is stopped when whilst the GetMessages function is running but is started again as it ends, preventing the same message from potentially being called multiple times at once.

Server Application

The servers main function opens a listening socket that it can bind to then waits for a connection to the socket that it can accept. This connection is then handled and the code loops indefinitely, after handling a connection it will await another connection to handle.

The server application makes use of POSIX threading in order to operate more efficiently in handling client connections. The servers main thread of operation will create a new thread to handle each incoming connection, thus allowing multiple connections to be open and processed simultaneously (provided the system in question is a multi-processor system), whilst the main thread can continue to accept new connections.

There are drawbacks to utilising this threading; primarily the added complexity that these bring to developing the solution. It is also important to ensure that the threads are properly terminated upon their completion.

A consideration that has to be made with the introduction of threading are the issues of thread safety and re-entrancy. Threads run simultaneously and autonomously of one another, this can mean that the threads can run unsafely; reading and writing values that another thread is currently

or is about to access. This parallel nature of the threads accessing shared memory can lead to unforeseen and unpredictable outcomes. In order to prevent such a race condition from presenting itself, it is important to ensure code is re-entrant or thread safe. According to a document on the subject provided by ICS for developing Qt applications (1) "...a class is said to be reentrant if its member functions can be called safely from multiple threads, as long as each thread uses a different instance of the class. The class is thread-safe if its member functions can be called safely from multiple threads, even if all the threads use the same instance of the class".

Reentrant code therefore will not have an effect on other threads running the same code as each thread will manipulate data held within its own instance that no other thread will have scope of.

Thread safe code actually reduces the efficacy of multi-threading as, in order to synchronise the threads to prevent unsafe access to shared data, the access to the shared memory must be made serial through use of a mutex. A mutex (mutual exclusivity) acts as a lock and key system for multi-thread synchronisation. When a thread approaches a mutex, the thread locks the mutex and proceeds into the code nested within. When another thread approaches a mutex that is currently locked, the thread will wait until the mutex is no longer locked. When a thread reaches the end of a mutex it is currently within, it will unlock the mutex to allow another thread access to the mutex section.

Whilst it would be possible to fork sub processes rather than utilise threading, there is still a great deal of complexity added the parallel processes but threads are able to share memory space. This shared memory usage is shown in the application whereby the server will store a list of messages and this list of messages can be accessed for both read and write purposes by each of the threads, allowing each connected client to have access to the same list of messages.

It is also worth noting that there are libraries that ease the difficulty of working with threading whilst not detracting from their utility or efficacy. Use of such libraries can reduce the number of bugs that are introduced due to a simplification of their usage, however it does also mean that there is another layer of code between the applications code and the code to perform the threading; this can mean issues can take longer to fix and occasionally may actually need to be fixed within the managed library itself as opposed to locally to the application. These issues should be rare however as these libraries are tried and tested and widely used.

Data Transfer

In order to transfer data from client to server, and vice-versa, there have been a number of classes created to structure the data in a uniform way so that when encoding/decoding, both applications know the schema of which the data should take.

There are numerous such classes used for this purpose such as the Message, Join, Leave and UpdateMessage classes, amongst others.

These classes are instantiated in order to occupy a place in memory which is then set with the appropriate values for the current point in time, for example the Join object can have the nickname of the user set, and then can be encoded and sent to the connected destination (be it client or server).

Before data is actually sent across the network, the size of the encoded object is sent ahead of the body of the object, this integer value will dictate to the destination the length of the message; allowing the recipient to know when to stop reading and to allow the destination to know the length of the whole object for use with decoding the object.

The readSize and writeSize functionalities, which use the recv and send methods respectively, are both blocking methods which will wait until they have successfully completed their intended activity with the socket; when intending to read from the socket the method will block the thread until a message is indeed read from the socket, this read value can then be returned and the thread can continue onto the next line (most likely the readN in this example).

Future Considerations

If redeveloping or supporting this application in the future, there are many changes that could be made to make the system more robust and more feature full.

The server currently only has one main thread to listen to and set up client connection handling threads as they are required. Whilst this is an adequate solution for small scale usage; if the application were to see very heavy traffic, this could cause to the system experiencing slight delays as creating threads is a relatively resource heavy activity and a backlog of incoming connections could start to develop. This could be eased by creating threads during quiet periods ,then assigning them to connections as they are needed.

The server currently does not expect errors, if an error is thrown within the sever, it will be unhandled and could cause the application to terminate. The application does not currently follow good defensive coding practices, which would ensure that object instances actually exist properly before accessing properties and methods that belong to it. If supporting this application to the next stage, this defensive coding would be needed to prevent a great number of show stopping incidents and increase the overall robustness of the application.

There is plenty of room for additional features in the application; common chat features such as whisper or announcement could be beneficial additions. Announcement would work similar to the current message style, however some form of tag would need to be sent with the message to denote it as an announcement; this would then be retained by the server in the message list and cascaded to all clients as normal, however when the client reads a message with this tag it will print the message to the chat window in a more elaborate style. Whisper however would be more difficult; this would require the message to be sent with a whisper tag and a destination user by nickname, the server will store this as a message and when message updates are requested, the message list is manipulated on a user by user basis so that the connection for that user can only see a message list that contains general messages and whisper messages to them. This will of course require larger changes in the way messages are stored and retrieved on the server as simply omitting messages will lead to issues with indexing. However it is certainly better to perform this filtering on the server as otherwise all messages, even whispers to other users, will be returned to every client just that whispers to other users will not show; that does not mean of course they are not available as these will now be a part of network traffic and is physically in memory. Using this simpler method could lead to unethical situations and potentially data security issues.

References

1. **ICS.** Reentrancy and Thread-Safety. *QT Reference Documentation*. [Online]
<https://www.ics.com/files/qtdocs/threads-reentrancy.html>.

Nick Hinder Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

78/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

GRADING FORM: SMU GENERIC_GJ

NICHOLAS HINDER

78

RESEARCH-INFORMED LITERATURE



A good report, well-written. A little more than one reference would be preferable - consider diversifying your reading as you ultimately move into the next academic year.

3

KNOWLEDGE AND UNDERSTANDING



Very good knowledge and understanding evident - future considerations section is especially good. It's

7

interesting, especially, to read this from the perspective of someone who understands the concepts, but who's enjoyed abstractions before this point!

ANALYSIS



Some good evidence of analysis - very good conclusions drawn, though slightly limited evidence of synthesis owing to the small number of sources.

6

PRACTICAL APPLICATION AND DEP.



A very good implementation - I know how much work went into it, and how challenging this no doubt was, coming from a non-C++ background. Very well done Nick!

58

SKILLS FOR PROFESSIONAL PRAC.



A well-written report, with good spelling, punctuation and grammar.

4