Computer Network 2018 Fall
Final Project Report

Members

B05902017 王盛立

- 工作比例: 45%
- 工作內容: 參與討論、Debug，程式撰寫，整理Report

B05902105 余友竹

- 工作比例: 55%
- 工作內容: 參與討論、Debug，主要程式撰寫，整理Report

How to run

我們有Makefile。因此只要make後你就能得到server跟client的執行檔。

對於server: ./server port_number

對於client: ./client ip port_number

測試平台是工作站，但在linux本機跟mac os也都測過。

在test.sh中是我們生成1g大檔案的方法。如果是mac os把大寫M改成小寫。

Implement Detail

使用c++實作因為c++比起c有了更多方便的函式庫可以使用。

在程式方面我們用了八個檔案，以及一個makefile。這八個檔案分別是server.cpp, protocal.cpp, protocol.hpp, command_handler.cpp, command_handler.hpp, client.cpp, client_command_handler.cpp, client_command_handler.cpp。在下面會針對每一個程式做說明。

server.cpp

在server端先照上次的作業內容建好socket。

接著我們打開一些檔案例如關於我們有哪些使用者，使用者間跟朋友的關係等等資料。並把他們存進對應的unordered_map中。

由於為了要實現傳輸檔案的功能，我們使用了**multithread**的結構，每一個client連進來，我們就分配一個thread用來跟那個client溝通。

除此之外我們還另外開了一個thread用來處理server端的stdin，因此我們可以在server端輸入一些簡單的指令用來觀察現在運作的狀況。

每一個分配用來跟client溝通的thread做的事情就是等待一個我們用class定義好的封包。而我們在頭尾都加了header，當我們收到東西後我們會去檢查他，如果頭尾都是正確的我們才承認我們收到的是正確的。這在之後傳輸file很重要，也是花很多時間才寫好的重要功能。

protocal.hpp

在裡面我們定義了各式各樣傳輸的常數。例如我們設定OP_REGISTER = 0x00，這是由client端若要註冊在傳輸封包時所要標明的，當我們的server收到時會去檢查client希望做什麼事於是回傳相對應的訊息。

我們所有的訊息封包都是由Request這個class所定義的。在這個class中有著header這個struct以及client所要傳的data。在header當中包含著是誰傳來的以及像是剛剛所提到的client要做什麼operation。以及我們剛剛所提到關於header的資訊。

另外我們還有一個user的class。裡面存著user的資訊例如他是誰以及他現在是上線還離線等等資訊。

protocal.cpp

這邊就是實作剛剛我們所說的Request跟User這兩個class中的內容，裡面有這兩個class中可以做的函式。

command_handler.hpp

包含剛剛所說那些unordered_map以供cpp使用。另外還有許多對應的mutex，避免race condition的發生，我們在幾乎所有資源當要寫入的時候，都先掛上了mutex以避免不可預期的事情發生。這點也是維持資料的一致性。

command_handler.cpp

這是由server端所處理所有來自client的種種request。具體有哪些功能將在下面說明。

client.cpp

與之前的作業差不多，先與server相連起來。同樣我們有做檢查header的處理。

我們開了兩個thread，一個負責接收來自server回傳的訊息，另一個負責處理stdin也就是鍵盤輸入指令。

client_command_handler.hpp

提供cpp一些必要的資訊。

client_command_handler.cpp

實作client所要做的事，為了code的整潔性，實際上client跟server所要做的事內容都寫在相對應的command_handler中。在client_command_handler中分成剛剛提到的處理server respond跟處理stdin， 具體有哪些功能將在下面說明。在server respond方面通常是client傳訊息後server的回應。server通常會回傳一句話表明你的client request是成功還是失敗，失敗的話是什麼原因。

Function

Register

在client端會要求你輸入你所要註冊的使用者名稱，密碼以及要求你在輸入一次密碼。

在server端會檢查你所要的名字有沒有註冊過，如果沒有就代表你註冊成功將你的資訊寫入檔案中。

在client傳輸封包之前由於我們都是用request這個class包起來的，所以在我們完成這個封包前我們都會在protocal.cpp中定義的函式中先檢查。例如如果你的名稱或密碼過長，或是你兩次密碼輸入的不一樣，你就無法完成封包也就不會傳給server。其餘功能的檢查也是一樣的道理。

Login

client端就是輸入你的名稱跟密碼。

server會檢查是否有這個使用者跟你輸入的密碼是否正確，如果有錯就會回傳對應的錯誤訊息。另外當你上線的時候你將會收到來自server所幫你儲存的離線資料。

Logout

方便的功能。

Send message

在client端就是輸入你要傳給誰以及你所要說的訊息內容。另外我們所設定的是你所要傳訊息的那個人，他必須要把你當成好友，你的訊息才能傳送給他，這點跟line是一樣的。如果對方不加你你的訊息他就無法收到。

在server端判斷完他有沒有加你好友後，就會接著判斷對方是否在線上。如果在線上他就會收到訊息，如果不在就會到離線資料等他登入後自動傳給他。

Receive message

嚴格說起來這並不算是一個功能。這是寫在server respond的方面，當有人傳訊給你你就會收到。

History message

你可以得知你過往的對話紀錄

Send File

當client端確認要送檔案後，他就會開啟另一個thread去執行這項工作，因此在client端你並不會因為傳檔案被block住。

在一開始我們的server並沒有使用multithread的設計，也沒有做header的特殊處理。因此儘管在localhost端的傳檔案是順利的，但只能到大約100mb的大小。如果是透過網路傳更是會有掉資料的狀況。因此我們改成multithread並加上了header，以確保傳的資料是對的。因為網路傳輸的關係，我們可能收到前面的卻沒有收到後面的。因此再加上header後，我們如果只收到部分的資料我們會把他先存入buffer，等剩餘的資料傳來也就是後面的header傳到我們才作處理。現在我們可以支援1G的大檔案傳輸。

ADD Friend

Client端接收你要加誰好友，而server端則更新好友的資料庫。

ADD FRIEND(MORE)

我們還支援刪好友，列好友等等功能。

Personal Imformation

我們能夠讓你設定你的個性簽名(跟line一樣)。同樣的你也可以去看別人的個性簽名。

這些功能都是透過定義好的常數讓client跟server間彼此知道要幹嘛，然後去做該做的事。

Encryption

我們有獨特的演算法處理client端對話間的加密功能。

Requirement

Registeration

如上所示，在register跟login，我們都有做到這方面的要求。

Messaging

如上所是我們可以send跟receive message。我們也有history跟offline message，這部分的要求我們也都有做到。

File Transfer

要求只有同時檔案傳輸，但我們還可以支援同時大檔案傳輸。

User Interface

當你剛登進client時，你會先看到要註冊或是登入。在完成登入後，你可以看到你能做的事。這部份兩階段的user interface我們有作出來。

除了這個以外，我們還有做另一版本的ui，他有更漂亮的介面，與更方便的使用者功能，我們相信這是實用的功能。

Bonus

我們有加好友功能以及對好友功能的更多操作。

另外你可以透過dd if=/dev/urandom of=test.txt bs=1M count=1000 來建立1g的檔案因為我們有大檔案傳送的功能。

此外我們還有加密client間的對話。

最後是我們對user端做了更多操作，例如你能夠登出或是你能夠設定你的個人牆讓你能說你想說的話，而大家都能看到。

以上是我們做的4個bonus。