Computer Networks Final Project Report

簡介

此次專案主要以Python和部分的C寫成，搭配Tcl/Tk圖形函式庫，分為伺服器端和客戶端，以下分別介紹兩方間通訊的協定規範，以及建置、使用說明和程式架構等。

通訊協定規範

通訊的部分，使用TCP/IP協定的socket進行連線，每筆訊息中第一個位元組代表請求的類型(Request Byte)，之後則是訊息內容，大致可以分為伺服器端使用、客戶端使用以及雙方都使用這三類。

在客戶端登入前的通訊較為單純，客戶端在沒有進入任何請求Routine(登入、註冊)的狀況下，第一次發送的訊息會附帶Request Byte，伺服器端收到後會進入對應的Routine，而該Routine的執行進度會被保存在連線的資訊中，因此在該Routine處理完成前，雙方之間的通訊內容皆不需再附帶Request Byte，處理完成後，伺服器端會發送代表處理結束的Response Byte告知客戶端。

當客戶端登入後雙方的每一筆訊息都會附帶Request(Response) Byte，告知對方請求類型，或執行結果。若部分訊息需要額外資訊(如檔案傳輸需要Port號碼、IP位址，文字訊息需要發送對象)，則以 `\n` 進行分隔。

Client使用的Request Byte

- `REGISTER_REQUEST` ：進行註冊
- `LOGIN_REQUEST` ：登入
- `LIST_REQUEST` ：請求線上使用者清單
- `DISCON_REQUEST` ：結束連線
- `LOGOUT_REQUEST` ：登出
- `RSOCK_INIT` ：登入後告知使用者的rsock資訊(參照客戶端架構)

Server使用的Response Byte

- `REQUEST_FIN` ： 請求處理完成
- `LOGIN_SUCCEED` ：登入成功
- `LOGOUT_SUCCEED` ：登出成功
- `HISTORY_END` ：歷史訊息結尾

雙方皆使用的Request(Response) Byte

- `MSG_REQUEST` ：發送訊息(傳送者發送給Server、Server發送給接收者)
- `TRANSFER_REQUEST` ：要求傳輸(傳送者發送給Server、Server發送給接收者)
- `TRANSFER_ACCEPT` ：同意傳輸(Client)、傳輸已同意(Client)
- `TRANSFER_DENY` ：拒絕傳輸(Client)、傳輸被拒絕(Server)
- `HISTORY_REQUEST` ：請求歷史訊息(Client)、告知訊息為歷史訊息(Server)

關於以上Byte的詳細定義位在 `codes.py` 中。

用戶及管理者指南

伺服器之運行環境

伺服器以Python撰寫，使用SQLite3資料庫儲存資料，執行要求環境安裝Python版本3.5或以上，以及SQLite3。

伺服器之設定

伺服器預設使用Port 16666監聽連線，可在 `config.py` 中對 `PORT` 參數進行修改，若Server需要傳透NAT，也必須在路由器中設定Port Forwarding。
伺服器使用之資料庫路徑也可於config.py中，對 `USERS_DB_PATH` 參數進行設定。

伺服器之建置與執行

伺服器並不依賴於需要編譯之檔案，惟初次執行前須先執行 `$ ./setup_sql.py` 以建立SQLite資料庫及資料表。
接著輸入 `$ ./server.py` 即可執行server，輸入Ctrl+C結束。

客戶端之運行環境

客戶端以Python及Tcl/Tk撰寫，檔案傳輸的功能部分則使用C語言撰寫，其中引用了Windows不支援之POSIX相關函式庫，因此Client僅能在UNIX-like系統上執行，且要求環境安裝Python版本3.5或以上，及安裝Tcl/Tk相依之套件。

客戶端之設定

客戶端預設使用Port 16888~16898進行檔案傳輸，執行時會從 `config.py` 中讀取伺服器之位址，因此執行客戶端前請在 `config.py` 中設定伺服器之網域名稱或IP位址。

客戶端之建置與執行

需先輸入 `$ ./make` 編譯檔案傳輸相關之功能，之後輸入 `$ ./client.py` 即可執行client。

客戶端之使用

剛開啟Client時，若成功連線，會進入command line之介面，顯示如下的prompt:

```
(start)>
```

在此介面下，可以進行登入、註冊及離開三種動作，分別需輸入指令 `login` 、 `register` 、 `exit` 。
如欲註冊，輸入 `register` 後遵循介面的提示，分別輸入帳號、密碼，並再次確認密碼，若沒有錯誤則註冊成功。之後可以 `login [username]` 或 `login` 進行登入。

成功登入後，會彈出圖形視窗，在此介面下，若有未讀訊息，則登入時會一併跳出。
主視窗共有三個按鈕 `Online Users` 、 `New Chatroom` 以及 `Logout` ，點選 `Online Users` 後會在上方顯示出目前在線上的使用者，點選 `New Chatroom` 則可以選擇一名使用者進行對話，點選Logout則會登出並關閉視窗，回到Command Line介面。若關閉是窗而未Logout，則可以在Command Line介面鍵入 `exit` 指令以外的任何字串重新開啟視窗。

點選 `New Chatroom` 輸入欲進行的對話的使用者後，會開啟聊天視窗，在視窗的下方有一個文字輸入區，一個 `File` 按鈕以及 `History` 按鈕。

於文字輸入區輸入想要傳送之訊息後，按下Enter即可傳送給對方。點擊File按鈕後，會跳出檔案選擇視窗，選擇檔案後會跳出確認視窗，此時仍可以取消，若確定則會向對方發送檔案傳輸請求，若對方亦接受，則會開始傳輸，傳輸結束後會顯示訊息於聊天視窗中。點擊History按鈕後會跳出視窗，要求輸入要查詢的歷史紀錄筆數，之後會將最後幾筆顯示在聊天室中。

程式架構與設計

專案的架構分為伺服器(Server)及客戶(Client)端。

伺服器之架構

伺服器端運行時，會先初始化監聽用的socket，以及測試資料庫連線，完成後以 I/O Multilplexing架構處理請求，當接受新的連線時會加入到polling清單中，連線關閉則從中移除。對於每個連線，伺服器會記錄該連線的：

- 登入狀態
- 使用者名稱
- 使用者ID
- 上次登入時間
- 某些請求的處理狀態

接收到請求後會解析請求的類型代碼(參照通訊協定規範)，再依照代碼呼叫相應的handler進行處理。涉及資料儲存的部分則使用SQLite3資料庫進行儲存。當收到SIGINT訊號時，Server才會終止運行。

伺服器端之資料庫設計

users資料表

id	username	password	reg_time	last_login
INT (PRIMARY KEY)	TEXT (UNIQUE NOT NULL)	TEXT (NOT NULL)	TEXT (NOT NULL)	TEXT

其中 `id` 是每名使用者唯一的編號，為主鍵， `username` 為使用者名稱， `password` 為以sha256及salt雜湊加密過之密碼(salt可在 `config.py` 中修改 `PASSWORD_SALT` 參數)， `reg_time` 為使用者之註冊時間， `last_login` 為使用者上次登入之時間。

messages資料表

id	src	dest	time	msg	read
INT (PRIMARY KEY)	TEXT (NOT NULL)	TEXT (NOT NULL)	TEXT (NOT NULL)	TEXT (NOT NULL)	INT (NOT NULL)

`id` 為每一筆訊息唯一的編號，為主鍵， `src` 為訊息的發送者， `dest` 為訊息的發送對象， `time` 為伺服器端收到訊息時的UTC時間(ISO格式)， `msg` 為訊息內容， `read` 為一boolean值，標記發送對象是否已接收過本訊息(若發送時對象離線，則會是True)。

客戶端之架構

客戶端大致可分為兩個階段，其一為登入前，一則為登入後。

第一階段

在第一階段時僅有文字介面，可以指令進行註冊、登入或離開，會由簡單的指令解析器處理這些指令並送出請求，在此階段socket每次送出請求後，即阻塞等待伺服器回應，而當登入請求認證成功後，隨即切換到第二階段。

第二階段

在第二階段有Tcl/TK之圖形介面，在登入後，為了簡化設計及同步問題，客戶端會另外建立一個僅用

於接收訊息的socket，稱為rsock，而送出訊息仍然使用在開啟時建立的原本的socket，稱為sock。

sock每次送出訊息後，都會阻塞等待伺服器端傳回確認訊息。而rsock則以I/O Multiplexing模型進行處理，當接收到訊息時，會判斷為一般訊息、歷史紀錄或是檔案傳輸請求，並進行相應的處理。

聊天室

第二階段的處理主要以兩個類別實作，分別是 `LoginManager` 和 `Chatroom` 。一個 `Chatroom` 物件表示一個與特定對象的對話視窗，而 `LoginManager` 會開啟一個主視窗並處理使用者在主視窗中送出的請求。

為了將不同對象送來的訊息送到正確的聊天室， `LoginManager` 會儲存目前開啟的所有聊天室以及其對應的對象。在主視窗開啟的同時， `LoginManager` 也會開啟rsock，接著處理從rsock接收到的訊息，利用訊息中的使用者名稱資訊將訊息轉送給對應的 `Chatroom` 。若傳送訊息的使用者不在 `LoginManager` 儲存的名單中，就建立新的 `Chatroom` 並將之加入儲存的聊天室當中。

雖然使用兩個socket已經避免一部份的同步問題，但是多個視窗同時使用仍然可能會有其他同步問題發生。因此，在 `LoginManager` 和 `Chatroom` 傳送訊息到sock之前，會先取得一個共同的lock，並在接收到伺服器端回應時釋放。在建立聊天室、根據聊天室存在與否做出相應動作和關閉聊天室時會索取另外一個lock，以免訊息傳送到不存在的聊天室產生錯誤。

檔案傳輸

主程式

檔案傳輸是以C語言實作的獨立模組，編譯完共有file_recv和file_send兩個程式，客戶端收到請求後，傳送端會建立新的file_send程序，接收端則建立新的file_recv程序。
兩個程式直接呼叫的方法如下：

```
./file_recv <port> <filename>
./file_send <ip> <port> <filename>
```

若是十秒內無法成功連線，則會自動結束，並回傳0告知客戶端。傳輸完成後，會以checksum進行檢查，若有錯誤則會刪除檔案，並一樣回傳0告知客戶端，若成功則回傳1。

Python介面

在使用者從GUI中選定要傳送的檔案後，傳送端會傳送 `FILE_REQUEST` 給伺服器，伺服器收到再轉送給接收端。在轉送前，伺服器會檢查接收端的狀態，若對方不在線上就會將 `FILE_DENY` 連帶不在線訊息傳送給傳送端。

另一方面，接收端用戶若同意接收檔案，則會從Synchronized Queue取出一個預設的Port，將 `FILE_ACCEPT` 連帶Port訊息傳送給伺服器，並用Subprocess呼叫file_recv程序。若用戶不同意接收訊息，就直接傳送 `FILE_DENY` 給伺服器。

當伺服器收到接收端送來的 `FILE_ACCEPT` ，就將 `FILE_ACCEPT` 連帶接收端的 `<ip>:<port>` 訊息傳送給傳送端。若收到 `FILE_DENY` ，就直接轉送給傳送端。最後，當傳送端收到 `FILE_ACCEPT` 就用Subprocess呼叫file_send。