Programming Assignment (phase3)
B07705049 資管三 林稜凱

1. 編譯與執行

解壓縮後的資料夾結構如下：

```
openssl/
    client                          ← binary 執行檔
    server                          ← binary 執行檔
    makefile                        ← makefile
    src_client/                     ← source code
        main.cpp
        …
    src_server/                     ← source code
        main.cpp
        …
    threadpool/                     ← threadpool 相關 library
        …
    ssl/                            ← ssl 相關 library
        …
    client_CA/                      ← client 用私鑰與憑證
        …
    server_CA/                      ← server 用私鑰與憑證
        …
    CA.crt                          ← 共用 CA
    report.pdf                      ← 此說明文件
```

資料夾內的 client 與 server 是已經編譯完成的執行檔。若想要重新編譯，使用 `make` 可以重新編譯 client 與 server 執行檔，使用 `make client` 或 `make server` 可以單獨重新編譯其中一個執行檔。請注意 source code 中包含 c++11 語法，因此若要使用 make 以外的方法編譯，請使用支援 c++11 語法的編譯器與 standard。此外，makefile 中以 –I 來引入我自己電腦中的

openssl library 路徑，在我的電腦上可以順利執行，但每台電腦的 library 路徑可能不同。若想要刪除 client 與 server 執行檔，可以使用 `make clean`。

程式在 macOS Catalina 10.15.7 與 Kali 2020.3 皆已測試過可以成功編譯並執行，在其他 linux（如 Ubuntu）應也可以順利編譯並執行。

2. 程式操作說明

2–1. client 操作說明

編譯並執行程式之後，程式首先會要求使用者輸入 server 的 IP 與 port：

```
ip address: 192.168.31.58
port: 8080
```

如上圖方式輸入之後，程式會先檢查 IP 與 port 是否 valid（預設的 valid port 為 1025 ~ 65535），若 valid，會開啟 socket 並嘗試與 server 建立 TCP connection。

若成功連上 server，結果如下圖所示：

```
ip address: 192.168.31.58
port: 8080
Connection accepted!

cmd>
```

程式接受使用者以 CLI 方式輸入指令並執行。若輸入的指令沒有定義，會出現以下提示訊息：

```
cmd> idk ...
Invalid command.
Try 'help' for help.
```

輸入 help，會顯示所有指令及其用途：

```
cmd> help

commands:
    reg   : create a new account
    login : log in to your account
    list  : list the user(s) online
    trans : make transaction with another user
    exit  : quit the system
```

每個指令有特定的參數數量與順序，若不符合使用方式，會顯示使用提示：

```
cmd> reg
usage: reg <user account name> <deposit amount>
```

以上 5 種指令，即可完成註冊、登入並接收交易訊息、得到上線用戶資訊、離線等功能。

使用範例：

– 註冊新使用者，使用者名稱為 user1，帳戶餘額為 1500 元：

```
cmd> reg user1 1500
account successfully created.
```

– 登入 user1，並以 port 5050 接收其他使用者傳送的交易訊息：

```
cmd> login user1 5050

welcome, user1!
account balance: 1500

1 user(s) are currently online:

user1#192.168.31.134#5050
deposit amount: 1500
```

– 列出目前上線的所有使用者的 IP、port 與帳戶餘額：

```
cmd> list

user name: user1
account balance: 1500

2 user(s) are currently online:

user1#192.168.31.134#5050
deposit amount: 1500

user2#192.168.31.134#6060
deposit amount: 2000
```

– 傳送交易訊息給 user2，內容為向 user2 付款 300 元：

```
cmd> trans 300 user2
transaction with user2 is done.
```

user2 會收到以下訊息：

```
cmd>
received 300 from user user1.
```

– 離線：

```
cmd> exit
Are you sure you want to exit? [y/n]: y
Bye!
```

2–2. server 操作說明

執行程式時，可以額外傳入 1 至 2 個 arguments，來指定 worker pool 的 max thread 與 max queue。若未指定，則預設 max thread 為 10， max queue 為 256。若只輸入 1 個 argument，則該參數用來指定 max thread。

使用範例：

```
~/Documents/share_with_vms/Computer Networks/server INSERT ./server
Max THREAD: 10
Max QUEUE : 256

port: ^C
~/Documents/share_with_vms/Computer Networks/server INSERT ./server 5
Max THREAD: 5
Max QUEUE : 256

port: ^C
~/Documents/share_with_vms/Computer Networks/server INSERT ./server 5 10
Max THREAD: 5
Max QUEUE : 10

port: ^C
```

程式首先會要求使用者輸入 server 運行的 port，若成功，結果如下圖所示：

```
port: 8080

ready for connection
```

當有新的使用者連上 server，server 會印出新連線的 IP address：

```
client IP: 127.0.0.1
```

當 server 收到 request 時，會印出 request 內容與 response 內容：

```
REGISTER#user1#3000

response: 100 OK
```

若有 client 端意外 shutdown（意即非透過 exit 來終止程式執行），server 端會顯示 shutdown 的 client 的 IP address，若 client 有登入帳號，會一併顯示出帳號名稱，並在 server 端登出此帳號，使此帳號能再次在其他連線被登入。

```
A user (127.0.0.1) disconnected.

User user1 (127.0.0.1) disconnected.
```

對於 REGISTER、登入、LIST、EXIT 這些功能，client 與 server 間所傳送訊息的格式都與 spec 中描述相同。client 端在收到 response 後會將訊息以易讀的方式向使用者顯示。

TRANSACTION 功能的訊息傳送流程如下：

假設 user1 要向 user2 付款 1000 元：

1. user1 向 user2 傳送交易通知 (user1#1000#user2)
2. user2 向 server 傳送 request (TRANSACTION#user1#1000#user2)
3. server 判斷此交易成功與否，以下假設交易成功
4. server 向 user2 傳送 response (Received 1000 from user1.)
5. server 向 user1 傳送交易成功通知 (Transaction done with user2.)
6. 結束交易

其中 request 與 response 指的是 client 向 server 送出 request 後，會等待 server 回傳 response，然後將結果向使用者顯示。通知指的是直接發送訊息給 client，且此訊息並非用於回應 client 的某個 request。

假設上述交易失敗（例如 server 發現 user1 的餘額不足），則流程如下：

1. user1 向 user2 傳送交易通知 (user1#1000#user2)
2. user2 向 server 傳送 request (TRANSACTION#user1#1000#user2)
3. server 判斷此交易成功與否，以下假設交易失敗
4. server 向 user2 傳送 response (NOOUTPUT)
5. server 向 user1 傳送通知 (Deposit amount not enough.)
6. 結束交易

其中步驟 4 的 NOOUTPUT，是 server 的 response 中用來表示「此回覆不需要向使用者顯示」的訊息。user2 的 client 端收到此訊息後，得知此交易已失敗，因此不向使用者顯示任何訊息。user2 的使用者自始至終都不會得知這筆交易的產生。

server 中所使用的 thread pool，使用的是別人已經實作完成的 code。連結在參考資料中附上。

2-3. 安全傳輸實作方法與流程說明

本程式以 openssl 套件來實作 P2P 的安全資料傳輸。

在 client 與 server 程式開始執行時，都會分別透過指定路徑來讀入 .key（私鑰）與 .crt（憑證），以及 CA_FILE（驗證用的憑證清單）。在建立 socket 連線後，client 與 server 雙方會在此連線上建立 ssl 連線，並互相驗證，在連線建立時要求對方出示憑證，並在自己的 CA_FILE 中尋找對應憑證，以驗證真實性。完成驗證後，在 ssl 連線間傳送的資料就會自動在 sender 端被加密（用自己的私鑰），並在 receiver 端被解密（用對方憑證中的公鑰），因此可保證 client 與 server 間的安全資料傳輸（使用的安全傳輸方法為 SSLv23）。

而在處理 client 間的付款時，由於涉及身份冒用的問題，因此不能只使用上述的 P2P 加密解密方法，否則根據 2-2. 中提到的付款流程，user2 可以在 user1 不知情的情況下，向 server 宣稱 user1 要付款給自己，而 server 也只能驗證 user2 的身份，但無法確定 user1 是否有同意付款。因此，在處理 client 間的付款時，要將交易訊息分別用 user1 與 user2 的私鑰加密後再傳給

server，server 再分別用 user2 與 user1 的公鑰解密，以驗證此交易訊息是否真實。交易訊息使用的加密方法為 RSA-2048，padding 方法為 RSA_PKCS1，此加密方式能將 256 個以下的字元（2048 bits）的訊息加密為 256 bytes 的加密訊息，而此 padding 方式需要至少 11 個 bytes 的 padding 大小，因此密文長度不能超過 256 – 11 = 245 bytes。

假設 user1 要向 user2 付款 1000 元：

1. user1 會先產生交易訊息（user1#1000#user2），並用自己的私鑰加密此訊息成為 256-byte 的密文，然後將此密文透過 ssl 連線傳送給 user2。
2. user2 收到此密文後，需要用自己的私鑰再次加密後傳給 server。然而此密文的大小為 256-byte，因此無法使用上述的加密與 padding 方式，因此 user2 會先將密文切割為兩部分（200 bytes + 56 bytes），並分別用自己的私鑰加密兩份訊息，得到兩份 256-byte 的密文，並合併成為一份 512-byte 的訊息。除此之外，user2 也能透過 ssl 連線取得 user1 的公鑰，並解密 user1 傳來的密文，得到明文內容（user1#1000#user2），user2 會在此明文前加上 `TRANSACTION`，以向 server 表示此訊息的用途、付款人、受款人以及金額。user2 將明文與密文合併後送出給 server，此例子中，user2 送出的訊息為 `TRANSACTION#user1#1000#user2#<512-byte ciphertext>`。
3. server 收到此訊息後，得知此交易訊息是由 user1 付款給 user2，因此會先取出 user1 與 user2 的公鑰（在之前 user 向 server 建立連線時取得）。server 先解密最後面的 512-byte ciphertext，先將此密文分割成兩份 256-byte 的密文，並分別以 user2 的公鑰解密，兩份密文解密後會分別成為 200-byte 的密文與 56-byte 的密文。server 再將這兩份密文合併成為一份 256-byte 的密文，並用 user1 的公鑰解密，得到明文。此明文應該為 `user1#1000#user2`，server 會比對訊息最前方的明文與解密後的明文資訊是否相同，若相同則完成此次交易，若不同則停止此次交易。

如此一來，所有沒有 user1 的私鑰的人，包含 user2，則無法偽造此交易訊息，因為此交易訊息若在一開始未經過 user1 的私鑰加密，則 server 解密後就無法還原正確的交易訊息。

3. 參考資料

http://www.linuxhowtos.org/C_C++/socket.htm

https://github.com/mbrossard/threadpool

https://www.openssl.org/docs/manmaster/man3/

https://hackmd.io/@J-How/B1vC_LmAD