

計算機網路概論期末程式實作報告

108020022 周昱宏

(1)實作細節：

程式分成 client 端以及 sever 端兩個部分，先介紹 sever 端的設計，以下是

18020022_sever.c：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h> // for bzero()
4  #include <unistd.h> // for close()
5  #include <sys/types.h>
6  #include <winsock.h>
7  #include <windows.h>
8
9  #define AddressSize 20
```

先引入相關的標頭檔。

```
10 int main(int argc, char *argv[])
11 {
12     WORD wVersionRequested;
13     WSADATA wsaData;
14     int err;
15     //WSAStartup
16     wVersionRequested = MAKEWORD(2, 2);
17
18     err = WSAStartup(wVersionRequested, &wsaData);
19     if (err != 0) {
20         printf("WSAStartup failed with error: %d\n", err);
21         return 1;
22     }
23 }
```

在使用 winsock 前先呼叫 WSAStartup(WORD wVersionRequested,

LPWSADATA lpWSAData) , wVersionRequested 是 DLL 版本 , 使用

MAKEWORD 函式指定是版本 2.2 , LPWSADATA 是 WSADATA 結構 , 如果呼

叫失敗則輸出錯誤訊息。

```
25     struct sockaddr_in serverAddress, clientAddress;
26     int server_addr_length = sizeof(serverAddress);
27     int client_addr_length = sizeof(clientAddress);
28     int serverSocket, clientSocket;
29     int ServerPortNumber;
30
31     if(argc == 2){
32         ServerPortNumber = atoi(argv[1]);
33     }
34
35     serverSocket = socket(AF_INET, SOCK_STREAM, 0);
36     if(serverSocket < 0){
37         fprintf(stderr, "Error creating socket : %s\n", strerror(errno));
38         exit(0);
39     }
40
41     bzero(&serverAddress, server_addr_length);
42     serverAddress.sin_family = PF_INET;
43     serverAddress.sin_port = htons(ServerPortNumber);
44     serverAddress.sin_addr.s_addr = INADDR_ANY;
```

接著是創建 socket 的部分。創建完相關變數後 , 將與檔案一同輸入的參數藉由

atoi 轉換成整數存入 ServerPortNumber , 再來創建 serverSocket , AF_INET

是使用 IPv4 協定 , SOCK_STREAM 是 TCP protocol , 如果創建失敗則輸出錯

誤訊息。最後進行 serverAddress 的相關變數設定 , bzero 函式將該指標指定

長度範圍內清成 0 , PF_INET 指的是 sockaddr_in 是 IPv4 的結構 , htons 函式

將將本機端的字節序轉換成了網路端的字節序 , INADDR_ANY 表示不在乎

load IP 是什麼 , 讓 kernel 決定就好。

```

46     if(bind(serverSocket, (struct sockaddr *) &serverAddress, server_addr_length) == -1){
47         fprintf(stderr, "Error binding : %s\n", strerror(errno));
48         close(serverSocket);
49         exit(0);
50     }
51
52     if(listen(serverSocket, 1) == -1){
53         fprintf(stderr, "Error listening : %s\n", strerror(errno));
54         close(serverSocket);
55         exit(0);
56     }
57
58     printf("Waiting...\n");
59     if((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddress, &client_addr_length)) == -1){
60         printf("accept failed\n");
61         close(serverSocket);
62         exit(0);
63     }
64     printf("Client connect successfully\n");

```

再來是 sever 端蠻重要的三個函式。將 serverSocket 與 serverAddress 利用 bind 連結起來，如果連結失敗則輸出錯誤訊息。並用 listen 等待請求，其中 listen(serverSocket, 1) 的 1 是等待 server 接受連線前同時最大連線數，如果等待失敗則輸出錯誤訊息。以及最後用 accept 接受請求，如果接收失敗則輸出錯誤訊息。跑完後則輸出連結成功訊息。

```

66     int i;
67     int bytesRecv, bytesSend;
68     int itemID, itemNumber;
69     int itemcount[3] = {0};
70     char send_buf[500];
71     char recv_buf[500];
72     char str[100];
73     char messege[60][100];
74     int messege_idx = 0;
75     char *menu = "\
76 \n---Menu---\n\
77 1. Read all existing messages.\n\
78 2. Write a new message.\n\
79 Please type \"1\" or \"2\" to select an option:\0";
80
81     char *type_a_new_messege = "\
82 \nType a new message:\0";
83
84     char *new_messege_send = "\
85 \nNew messege sent.\n\0";
86     char *new_line = "\n\0";
87     char *messege_are_below = "\nAll messegas:\n\0";
88

```

接著是訊息寄送的相關宣告，其中的 messege[60][100] 是存放 sever 端要儲存的訊息、messege_idx 是目前儲存的訊息數量，利用 send_buf[500] 當每次寄送

的媒介、recv_buf[500]當每次接收的媒介。

```
90 // Send menu to client
91 send_buf[0] = '\0';
92 strcat(send_buf, menu);
93 bytesSend = send(clientSocket, send_buf, sizeof(send_buf), 0);
94 if(bytesSend < 0) printf("Error sending packet\n");
95
96
97 while(1){
98     bytesRecv = recv(clientSocket, recv_buf, sizeof(recv_buf), 0);
99     if(bytesRecv < 0) printf("Error receiving packet\n");
100
101     printf("%s\n", recv_buf);
102 }
```

接著利用 send 將主選單的訊息寄送過去，如果寄送失敗則輸出錯誤訊息。接

著進行迴圈利用 recv 接收每次從 client 接收到的訊息(選項 1 或 2)存放在

recv_buf 並把他輸出至螢幕。

```
103 if(!strcmp(recv_buf, "1", 1)){
104     send_buf[0] = '\0';
105     strcat(send_buf, messege_are_below);
106     for(int i=0;i<messege_idx;i++){
107         strcat(send_buf, messege[i]);
108     }
109     char main::send_buf
110     strcat(send_buf, menu);
111     bytesSend = send(clientSocket, send_buf, sizeof(send_buf), 0);
112     if(bytesSend < 0) printf("Error sending packet\n");
113 }
```

如果 client 端傳來的是選項 1，則要將目前的所有訊息輸出至螢幕上，因此先

把" All messeges:" 接到 send_buf 裡，並利用 for 把每一個訊息接到

send_buf 裡，最後一併把主選單附上提供下一次的選擇，利用 send 寄出如果

寄送失敗則輸出錯誤訊息。

```

113     else if(!strcmp(recv_buf, "2", 1)){
114         send_buf[0] = '\0';
115         strcat(send_buf, type_a_new_messege);
116         bytesSend = send(clientSocket, send_buf, sizeof(send_buf), 0);
117         if(bytesSend < 0) printf("Error sending packet\n");
118
119         bytesRecv = recv(clientSocket, recv_buf, sizeof(recv_buf), 0);
120         if(bytesRecv < 0) printf("Error receiving packet\n");
121         recv_buf[sizeof(recv_buf)] = '\0';
122         strcat(recv_buf, new_line);
123         strncpy(messege[messege_idx++], recv_buf, sizeof(recv_buf));
124
125         send_buf[0] = '\0';
126         strcat(send_buf, new_messege_send);
127         strcat(send_buf, menu);
128         bytesSend = send(clientSocket, send_buf, sizeof(send_buf), 0);
129         if(bytesSend < 0) printf("Error sending packet\n");
130     }
131
132     else{
133         bytesSend = send(clientSocket, menu, strlen(menu), 0);
134         if(bytesSend < 0) printf("Error sending packet\n");
135     }

```

如果 client 端傳來的是選項 2，則它可以輸入新訊息，因此 sever 先利用 send 將“ Type a new message:” 寄送過去如果寄送失敗則輸出錯誤訊息。接著接收 client 傳來的訊息存放在 recv_buf，如果接收失敗則輸出錯誤訊息。此時將這個訊息字串後面補上一個‘ \n’ 以便做選項 1 的輸出，並把他儲存於 messege 裡並將 messege_idx 數量加一，接著利用 send 將“ New messege sent.” 加上主選單寄送過去如果寄送失敗則輸出錯誤訊息。

如果 client 端傳來的不是 1 也不是 2 則利用 send 將主選單的訊息寄送過去，如果寄送失敗則輸出錯誤訊息。

139 | **WSACleanup();**

最後是 WSAClean()，即完成 sever 端的程式。

再來介紹 client 端的設計，以下是 18020022_client.c：

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h> // for bzero()
4  #include <unistd.h> // for close()
5  #include <sys/types.h>
6  #include <winsock.h>
7  #include <windows.h>
8
9  #define AddressSize 20

```

先引入相關的標頭檔。

```

11 int main(int argc, char *argv[])
12 {
13     WORD wVersionRequested;
14     WSADATA wsaData;
15     int err;
16     //WSAStartup
17     wVersionRequested = MAKEWORD(2, 2);
18
19     err = WSAStartup(wVersionRequested, &wsaData);
20     if (err != 0) {
21
22         printf("WSAStartup failed with error: %d\n", err);
23         return 1;
24     }

```

在使用 winsock 前先呼叫 WSAStartup(WORD wVersionRequested, LPWSADATA lpWSADATA) · wVersionRequested 是 DLL 版本，使用 MAKEWORD 函式指定是版本 2.2，LPWSADATA 是 WSADATA 結構，如果呼叫失敗則輸出錯誤訊息。

```

28     int server_addr_length = sizeof(serverAddress);
29     int serverSocket;
30     int ServerPortNumber;
31     char ServerIP[AddressSize];
32
33     if(argc == 3){
34         strcpy(ServerIP, argv[1]);
35         ServerPortNumber = atoi(argv[2]);
36     }
37
38     // Create socket
39     serverSocket = socket(AF_INET, SOCK_STREAM, 0);
40     if(serverSocket < 0){
41         printf("Error creating socket\n");
42         exit(0);
43     }
44
45     // Set the server information
46     bzero(&serverAddress, server_addr_length);
47     serverAddress.sin_family = PF_INET;
48     serverAddress.sin_port = htons(ServerPortNumber);
49     serverAddress.sin_addr.s_addr = inet_addr(ServerIP);

```

接著是創建 socket 的部分。創建完相關變數後，將與檔案一同輸入的參數 2 藉由 atoi 轉換成整數存入 ServerPortNumber，然後參數 1 則複製到 ServerIP，再來創建 serverSocket，AF_INET 是使用 IPv4 協定，SOCK_STREAM 是 TCP protocol，如果創建失敗則輸出錯誤訊息。最後進行 serverAddress 的相關變數設定，bzero 函式將該指標指定長度範圍內清成 0，PF_INET 指的是 sockaddr_in 是 IPv4 的結構，htons 函式將將本機端的字節序轉換成了網路端的字節序，INADDR_ANY 表示不在乎 local IP 是什麼，讓 kernel 決定就好。

```

51     // Connect to server
52     if(connect(serverSocket, (struct sockaddr *)&serverAddress, server_addr_length) == -1){
53         printf("connect failed\n");
54         close(serverSocket);
55         exit(0);
56     }

```

接著是 client 端比較特別的函式，利用 connect 與 sever 建立連結。如果建立

失敗則輸出錯誤訊息。

```
58     int bytesSend, bytesRecv;
59     char send_buf[500];
60     char recv_buf[500];
61
62     while(1){
63         bytesRecv = recv(serverSocket, recv_buf, sizeof(recv_buf), 0);
64         if(bytesRecv < 0) printf("Error recving packet\n");
65         printf("%s\n", recv_buf);
66
67         fflush(stdin);
68         scanf(" %[^\\n]", send_buf);
69         bytesSend = send(serverSocket, send_buf, sizeof(send_buf), 0);
70         if(bytesSend < 0) printf("Error sending packet\n");
71
72     }
73     WSACleanup();
74     return 0;
75 }
```

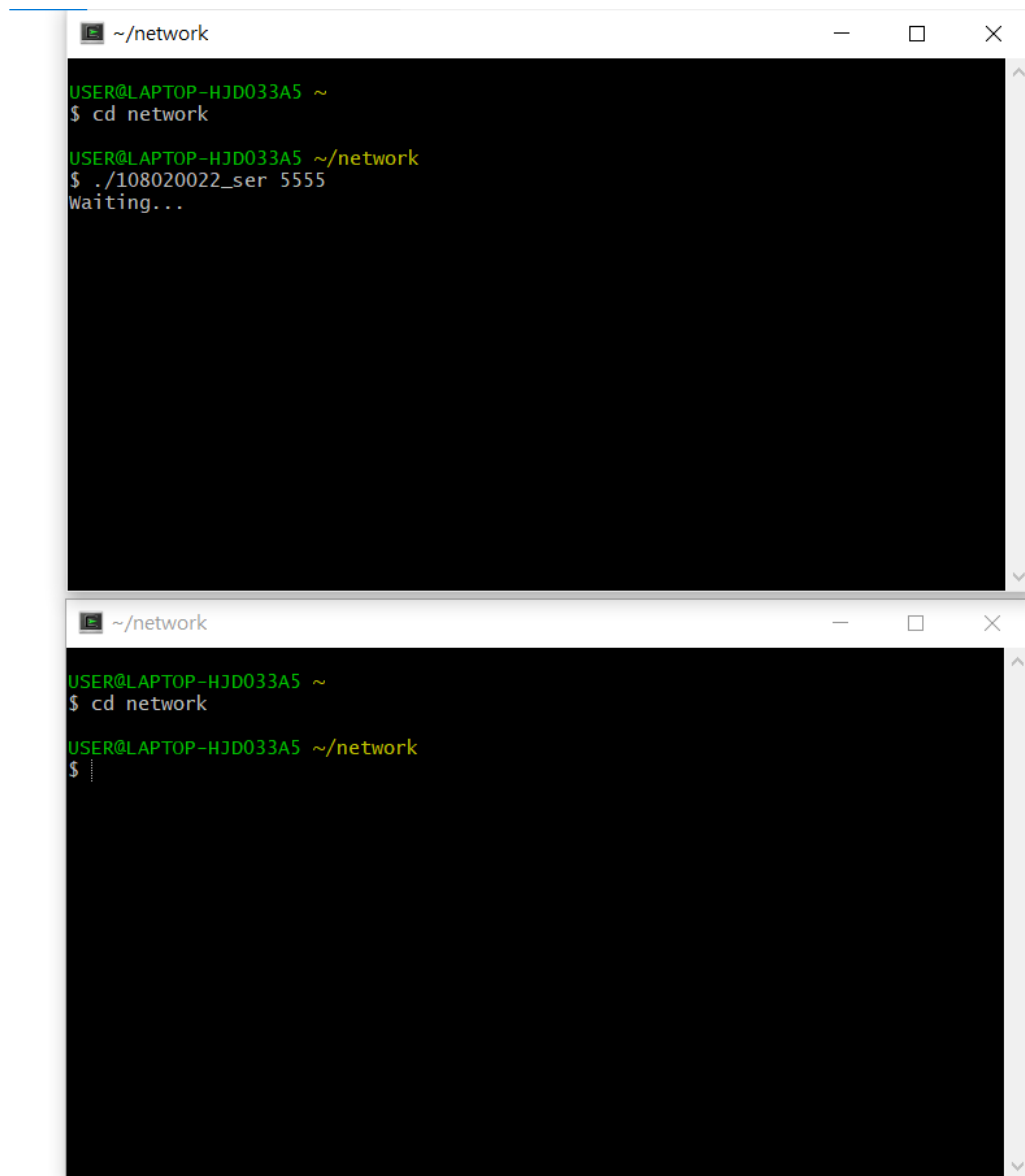
最後是傳遞訊息的部分，client 端每次利用 recv 接收 sever 端寄來的訊息，並

把他輸出至螢幕上，如果接收失敗則輸出錯誤訊息。接著輸入選項 1 或 2 利用

send 寄出，如果寄送失敗則輸出錯誤訊息。

最後是 WSAClean()，即完成 client 端的程式。

(2)程式截圖畫面：



The image displays two terminal windows stacked vertically. Both windows have a title bar that reads '~ /network'. The top window shows the following sequence of commands and output: a prompt 'USER@LAPTOP-HJD033A5 ~', the command 'cd network', a second prompt 'USER@LAPTOP-HJD033A5 ~/network', the command './108020022_ser 5555', and the output 'Waiting...'. The bottom window shows the same initial steps: 'USER@LAPTOP-HJD033A5 ~', 'cd network', and 'USER@LAPTOP-HJD033A5 ~/network', but the command './108020022_ser 5555' has not yet been executed, leaving a blank prompt '\$'.

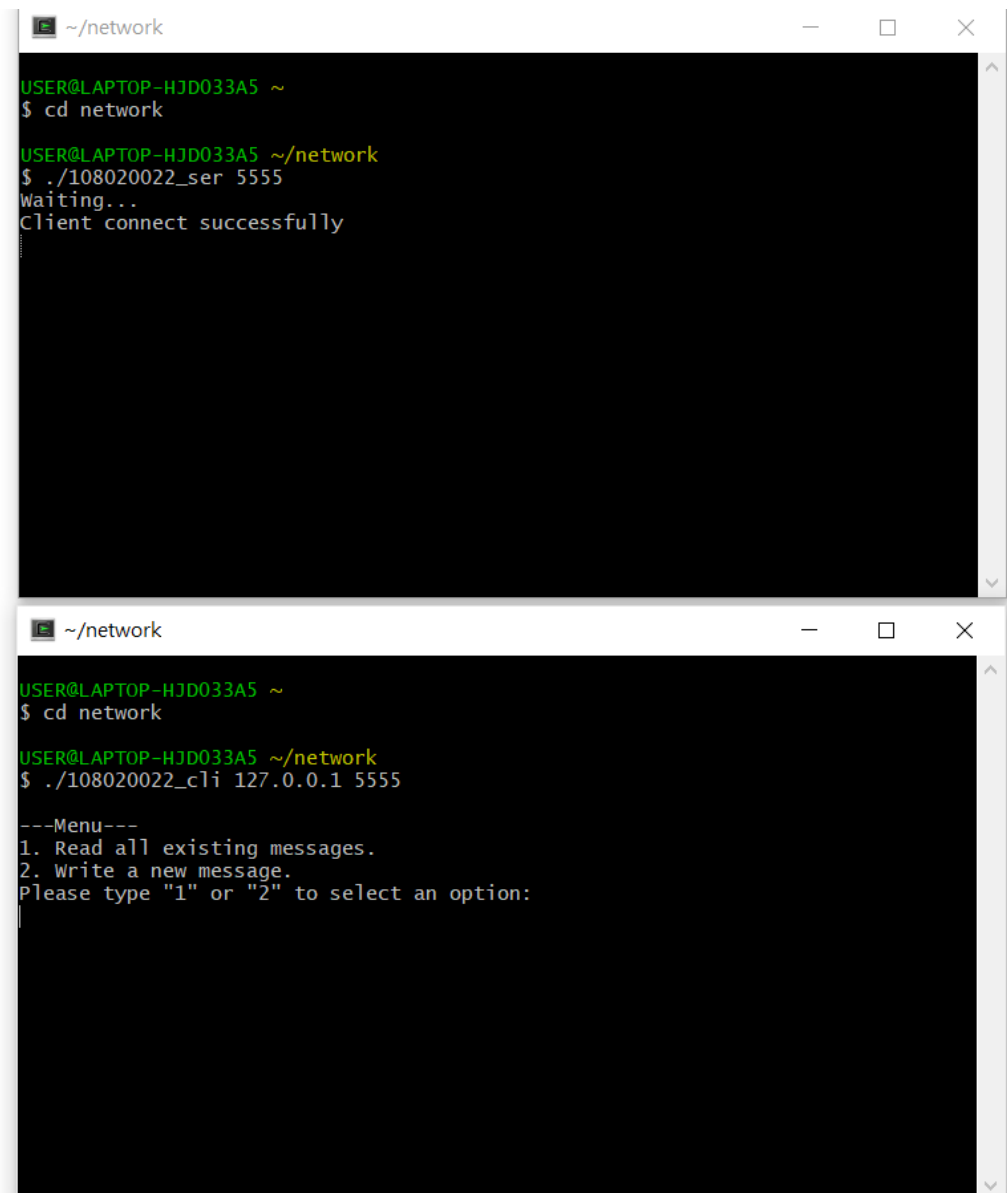
```
USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_ser 5555
Waiting...

USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$
```

先開始執行 sever 端的程式，並輸入 port number 5555。



```
~/network
USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_ser 5555
Waiting...
Client connect successfully

~/network
USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_cli 127.0.0.1 5555

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
|
```

再來執行 client 端的程式，輸入 ip 127.0.0.1 以及 port number 5555，server 端是連結成功，並寄出主選單，client 將主選單輸出至螢幕，此時 sever 端跑完 bind、listen、accept，client 端跑完 connect 的函式。

```
~/network
USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_ser 5555
Waiting...
Client connect successfully
1

~/network
USER@LAPTOP-HJD033A5 ~
$ cd network

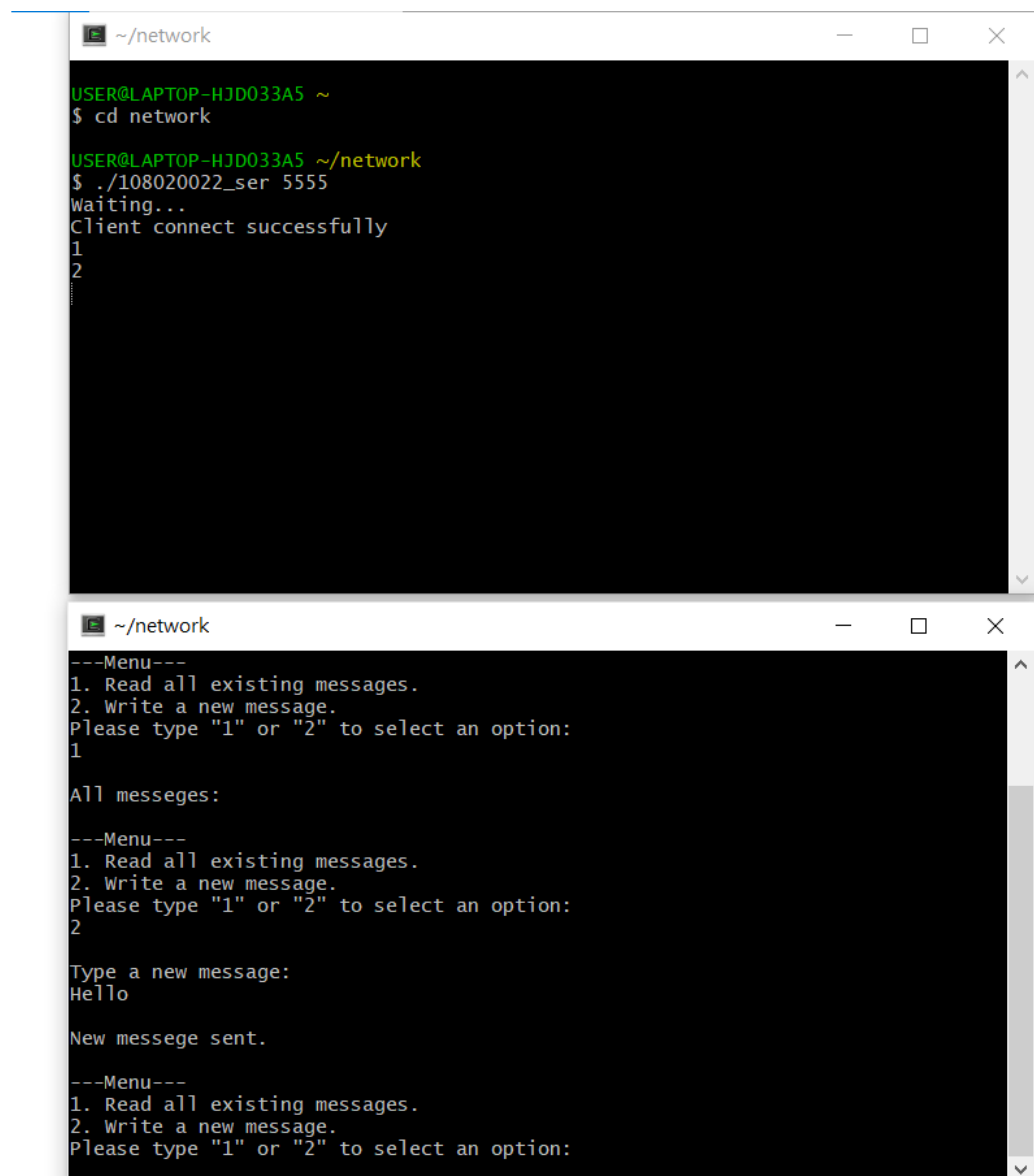
USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_cli 127.0.0.1 5555

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
1

All messages:

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
```

接著 client 端輸入選項 1，sever 端收到選項 1 並把他輸出至螢幕，並把當前所有訊息以及主選單寄給 client 端，client 端收到後輸出至螢幕。



```
~/network
USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_ser 5555
Waiting...
Client connect successfully
1
2
...
```

```
~/network
---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
1

All messages:

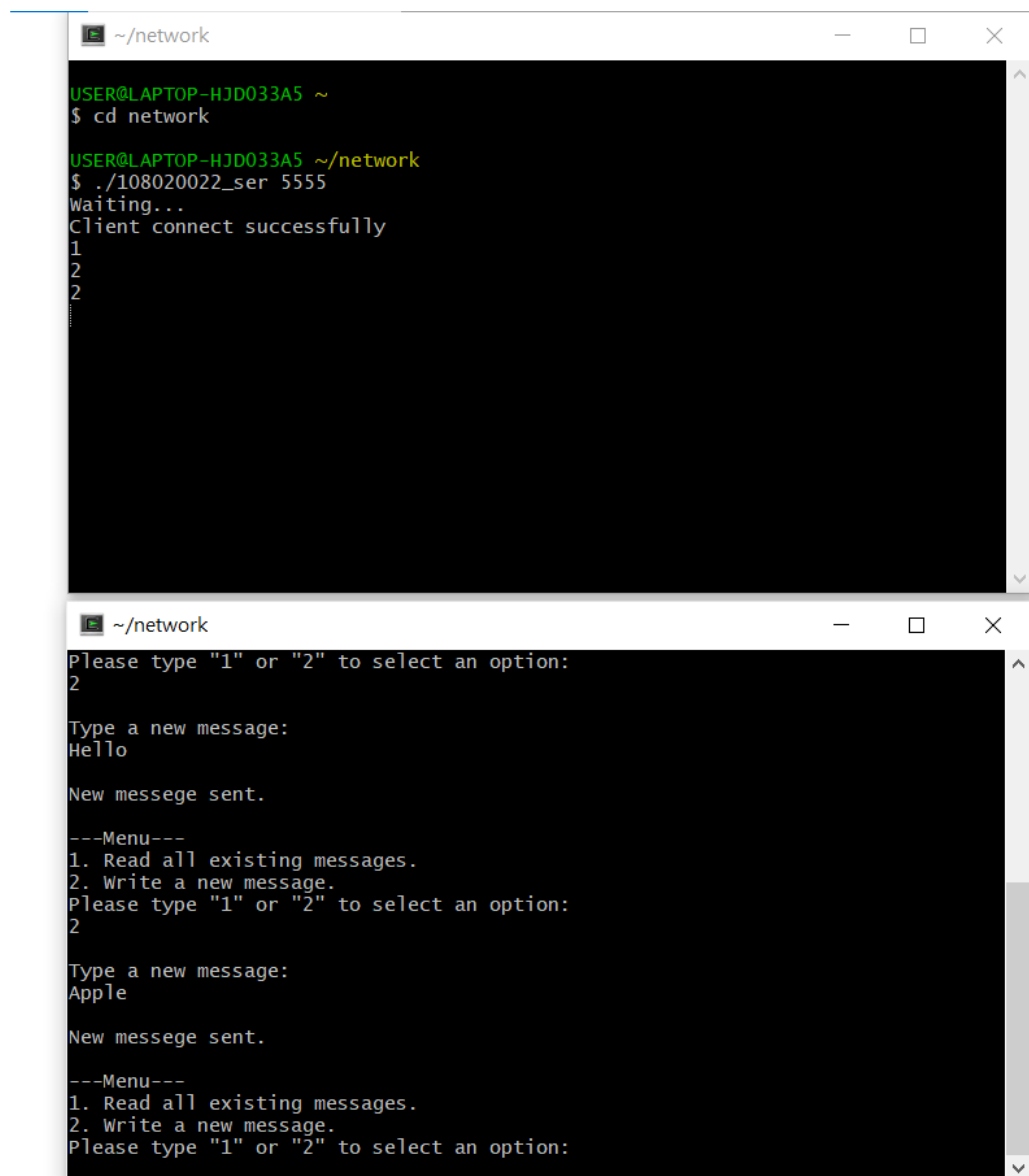
---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
2

Type a new message:
Hello

New messege sent.

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
```

接著 client 端輸入選項 2，sever 端收到選項 2 並把他輸出至螢幕，並寄寫新訊息給 client 端，client 端收到後輸出至螢幕並進行訊息寫入(Hello)寄給 sever 端，sever 端收到後進行訊息儲存，並把新訊息已寄送+主選單寄給 client 端，client 端收到後輸出至螢幕。



```
~/network
USER@LAPTOP-HJD033A5 ~
$ cd network

USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_ser 5555
Waiting...
Client connect successfully
1
2
2
.....

~/network
Please type "1" or "2" to select an option:
2

Type a new message:
Hello

New messege sent.

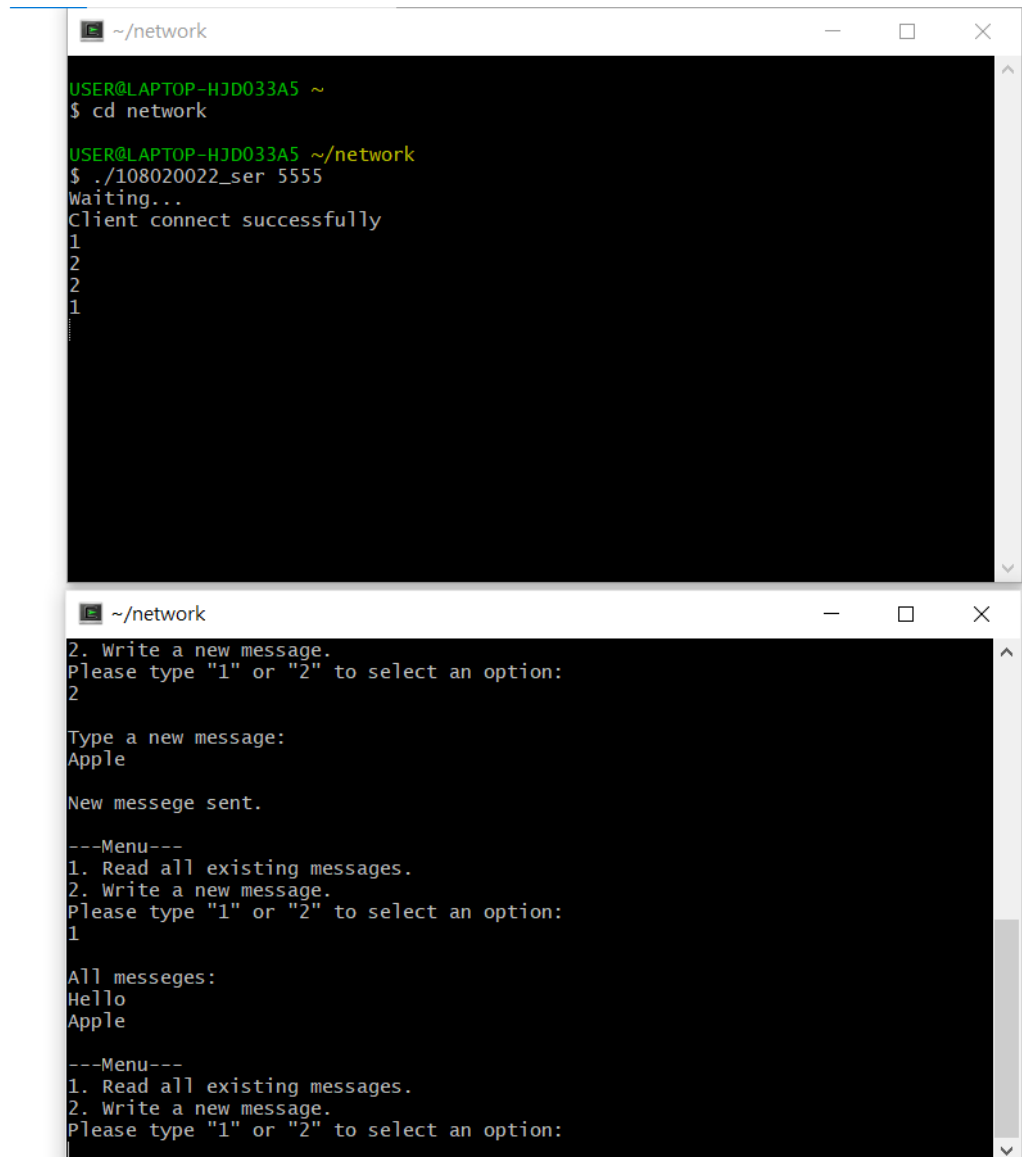
---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
2

Type a new message:
Apple

New messege sent.

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
```

接著 client 端又輸入選項 2，sever 端收到選項 2 並把他輸出至螢幕，並寄寫新訊息給 client 端，client 端收到後輸出至螢幕並進行訊息寫入(Apple)寄給 sever 端，sever 端收到後進行訊息儲存，並把新訊息已寄送+主選單寄給 client 端，client 端收到後輸出至螢幕。



```
~/network
USER@LAPTOP-HJD033A5 ~
$ cd network
USER@LAPTOP-HJD033A5 ~/network
$ ./108020022_ser 5555
Waiting...
Client connect successfully
1
2
2
1
.

~/network
2. Write a new message.
Please type "1" or "2" to select an option:
2

Type a new message:
Apple

New messege sent.

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
1

All messegges:
Hello
Apple

---Menu---
1. Read all existing messages.
2. Write a new message.
Please type "1" or "2" to select an option:
```

接著 client 端輸入選項 1，sever 端收到選項 1 並把他輸出至螢幕，並把當前所

有訊息以及主選單寄給 client 端，client 端收到後輸出至螢幕。

以上是程式實作截圖。

(3)問題與解決：

<1>一開始無法進行寄送 socket：因為沒有做 WSAStartup 的前置作業，參照

助教的 tutorial 及網路上資料才解決的。

<2> 訊息的儲存形式：sever 收到訊息後先加入換行，就可以在 client 輸出選項

1 時直接把訊息逐行輸出出來。