網路程式設計 109-2 NP midterm

一、 簡述題目完成度

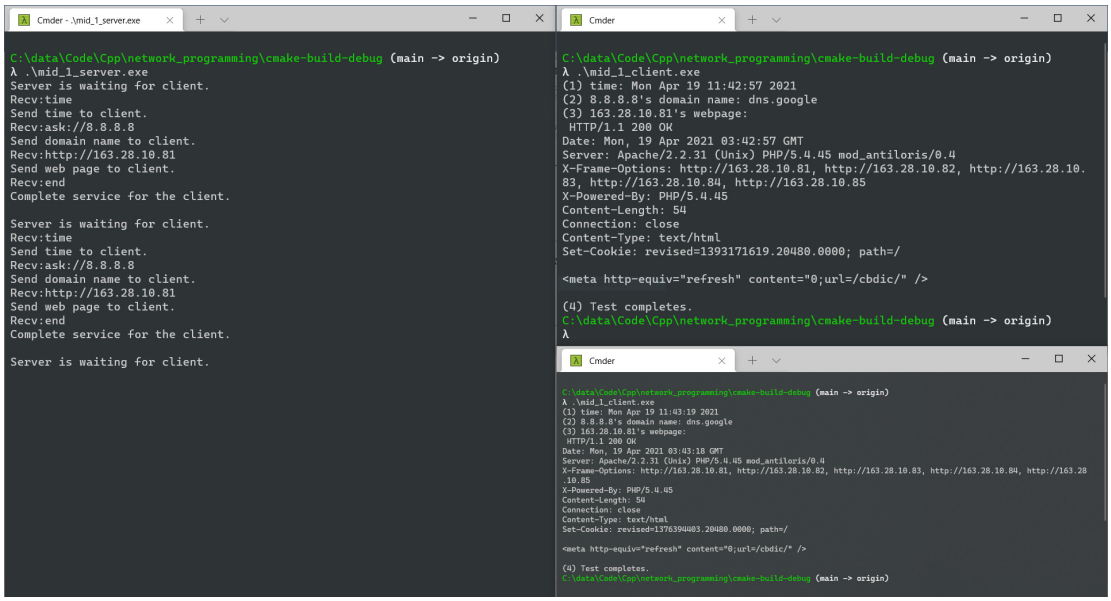
所有題目皆符合要求，全數製作完成。

編譯及執行程式時，請確認終端機編碼為 UTF-8，否則無法正常運作。

二、 執行畫面

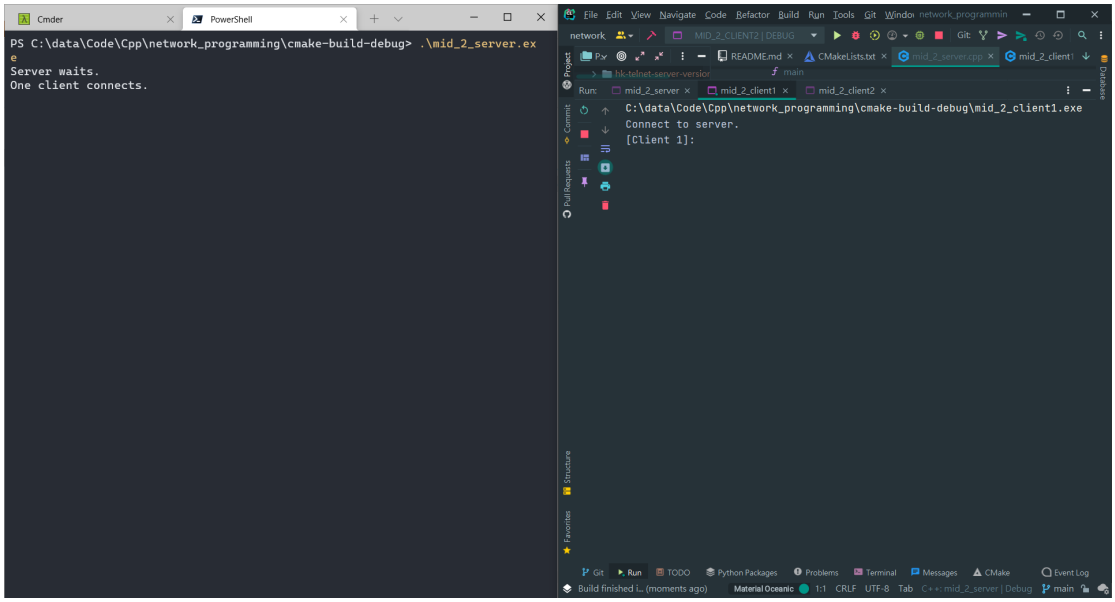
第一題：代理伺服器。

執行結果如下圖，左方為伺服器端畫面，右上方為使用者端畫面，右下方為另一個使用者端畫面。

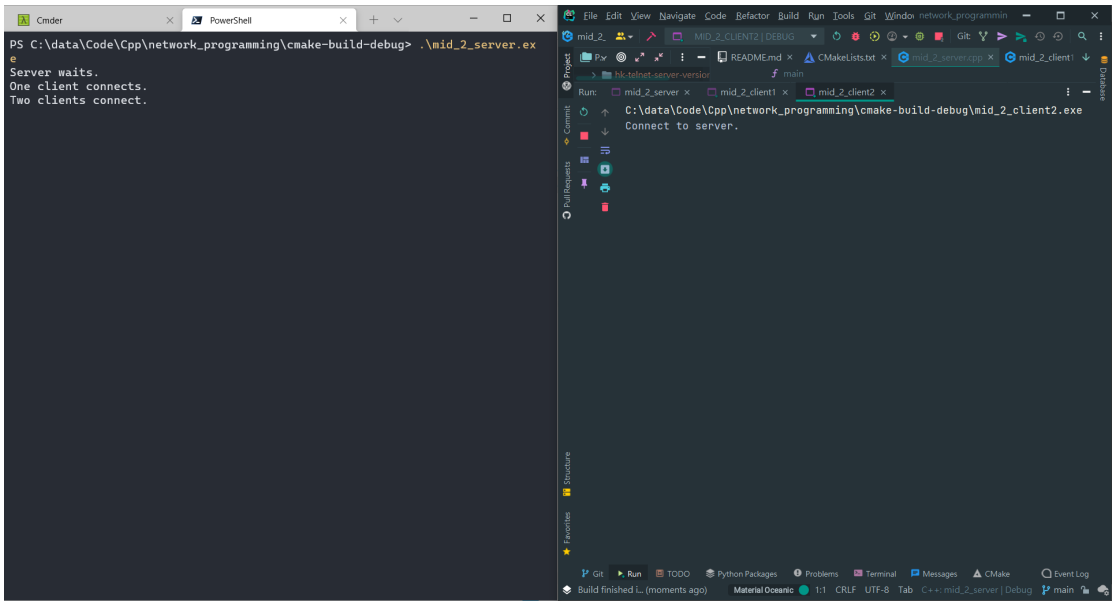


第二題：成語接龍。

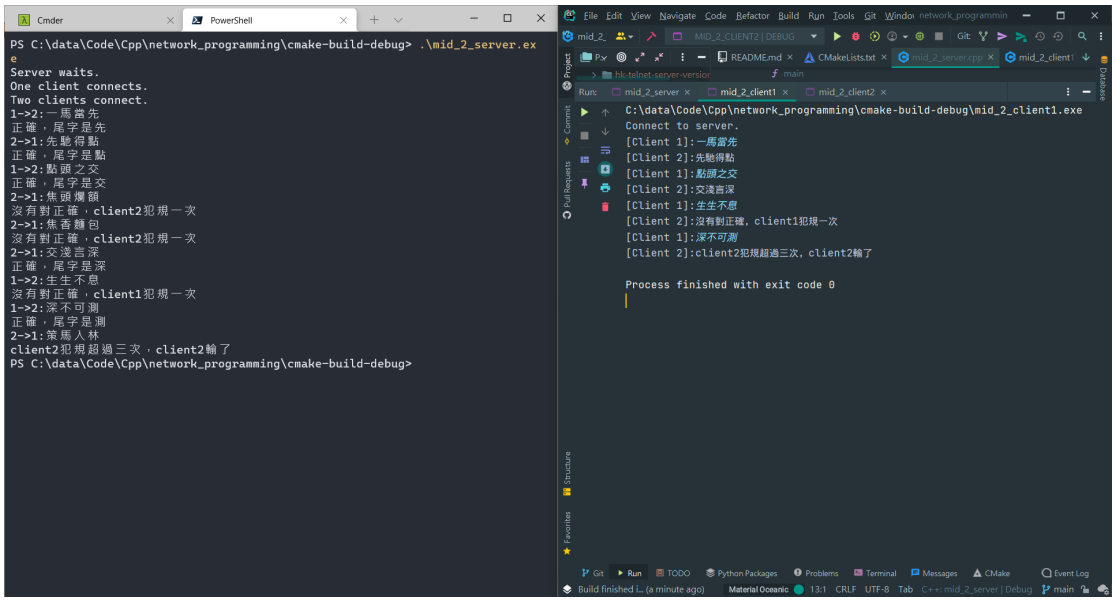
下圖左方為伺服器端畫面，右方為 1 號使用者端畫面，此為 1 號使用者已進入，但 2 號使用者尚未進入時的畫面。



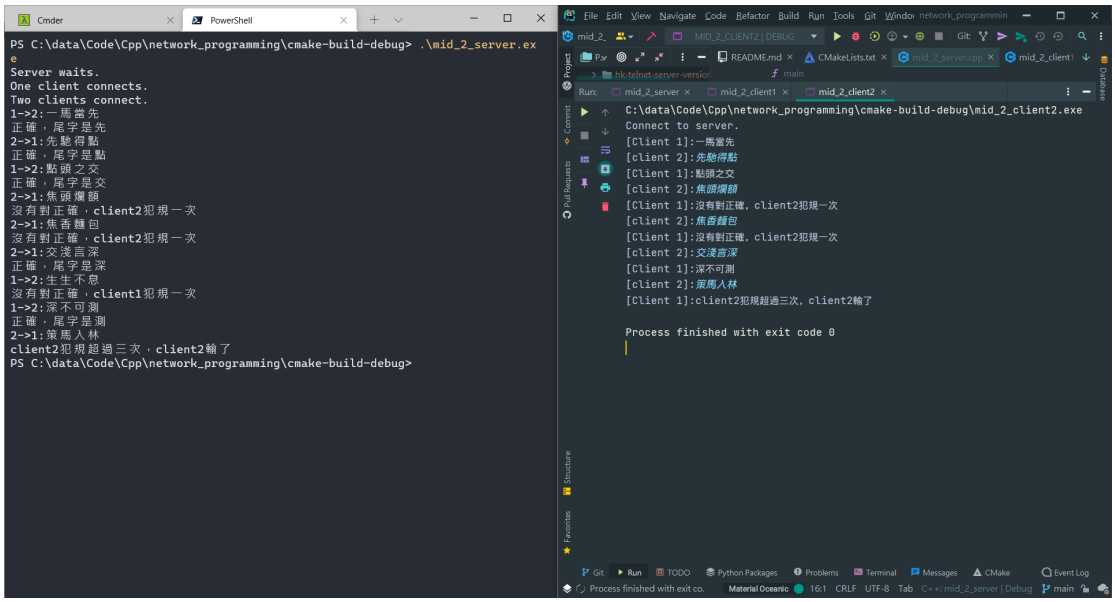
下圖左方為伺服器端畫面，右方為 2 號使用者端畫面，此為 1 號及 2 號使用者皆已進入的畫面。



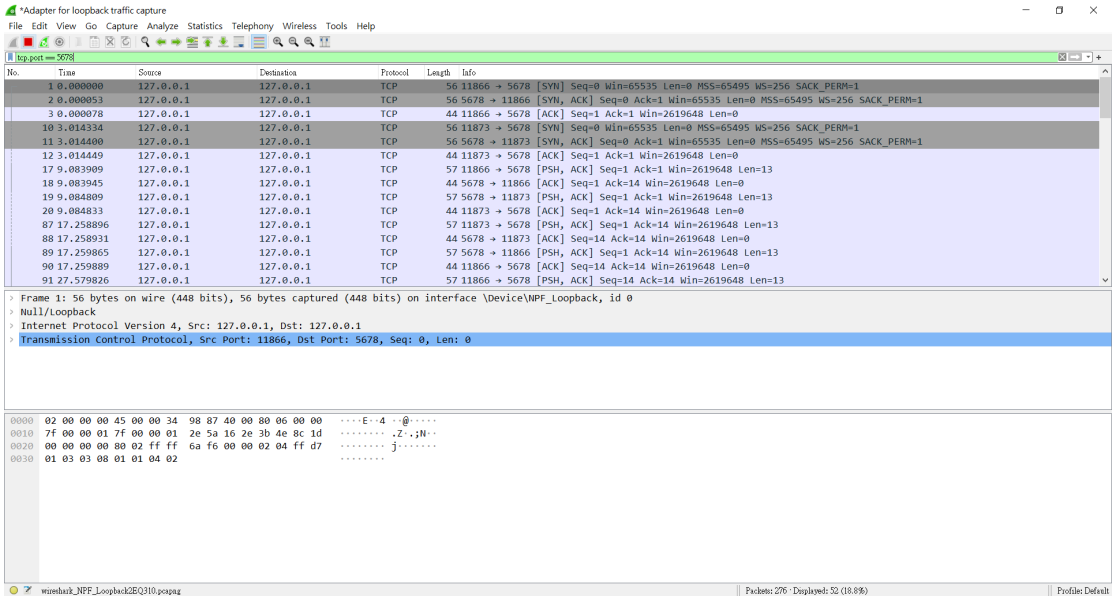
下圖左方為伺服器端畫面，右方為 1 號使用者端畫面，此為 2 號使用者犯規三次，程式結束的畫面。



下圖左方為伺服器端畫面，右方為 2 號使用者端畫面，此為 2 號使用者犯規三次，程式結束的畫面。



下圖伺服器端與兩個使用者端畫面互動時，使用 Wireshark 軟體擷取封包(TCP/IP 5678prot)的畫面。



三、 程式碼

環境

- Terminal Encode: UTF-8
- CMake 3.17.5
- MinGW w64 6.0
- GCC 8.1.0
- CLion 2021.1

[mid_1_client.cpp]

GitHub:

https://github.com/linwebs/network_programming/blob/main/mid_1_client.cpp

```
/*
 * NCYU 109 Network Programming mid 1 client
 * Created by linwebs on 2021/4/19.
 */
#include <stdio.h>
#include <winsock.h>
#define MAXSIZE 1024
int main(){
    WSADATA wsadata;
```

```
    struct sockaddr_in serv;
    SOCKET sd;
    char str[MAXSIZE];
    char str1[] = "time";
    char str2[] = "ask://8.8.8.8";
    char str3[] = "http://163.28.10.81";
    char str4[] = "end";
    WSAStartup(0x101, (WSADATA*) &wsadata); //
    sd = socket(AF_INET, SOCK_STREAM, 0);
    serv.sin_family = AF_INET;
    serv.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv.sin_port = htons(1234);

    connect(sd,(struct sockaddr*) &serv,sizeof(serv));

    send(sd,str1,strlen(str1)+1,0);
    recv(sd,str,MAXSIZE,0);
    printf("(1) time: %s",str);

    send(sd,str2,strlen(str2)+1,0);
    recv(sd,str,MAXSIZE,0);
    printf("(2) 8.8.8.8's domain name: %s\n",str);

    send(sd,str3,strlen(str3)+1,0);
    recv(sd,str,MAXSIZE,0);
    printf("(3) 163.28.10.81's webpage: \n %s \n",str);

    send(sd,str4,strlen(str4)+1,0);
    recv(sd,str,MAXSIZE,0);
    printf("(4) %s",str);
    closesocket(sd);
    WSACleanup();
}
```

[mid_1_server.cpp]
GitHub:
https://github.com/linwebs/network_programming/blob/main/mid_1_server.cpp

```
/*
 * NCYU 109 Network Programming mid 1 server
 * Created by linwebs on 2021/4/19.
 */
#include <iostream>
#include <chrono>
#include <cctype>
#include <cstring>
#include <ctime>
#include <winsock.h>

#define MAXLINE 1024

using namespace std;

/*
 * 執行內容
 * int status            => 執行狀態
 * int recv_head_len     => 接收到的標頭長度
 * int recv_content_len  => 接收到的內容長度
 * string page           => 取得的頁面
 * string recv_head      => 接收到的標頭長度
 * string recv_content => 接收到的內容長度
 */
struct content {
    int status = -1;
    int recv_head_len = 0;
    int recv_content_len = 0;
    string page;
    string recv_head;
    string recv_content;
};

content *get_page(const string &page);
```

```
int main() {
    SOCKET serv_sd;
    SOCKET cli_sd;
    SOCKET http_sd;
    int cli_len;
    char str[MAXLINE];
    char str_r[MAXLINE];

    struct sockaddr_in serv{};
    struct sockaddr_in cli{};
    struct sockaddr_in http{};
    WSADATA wsadata;

    // server's ip address
    const char server_ip[16] = "127.0.0.1";

    // server's port number
    u_short server_port = 1234;

    // receive bytes
    int rec_len;

    // send bytes
    int send_len;

    // bind status
    int bind_status;

    // Call WSAStartup() to Register "WinSock DLL"
    WSAStartup(0x101, (LPWSADATA) &wsadata);

    // Open a TCP socket
    serv_sd = socket(AF_INET, SOCK_STREAM, 0);

    // Prepare for connect.
    // Include sockaddr_ing struct (serv)
    serv.sin_family = AF_INET;
```

```
// server's ip address
serv.sin_addr.s_addr = inet_addr(server_ip);

// server's port number
// htons: host to network
serv.sin_port = htons(server_port);

// bind
bind_status = bind(serv_sd, (LPSOCKADDR) &serv, sizeof(serv));

if (bind_status == SOCKET_ERROR) {
    cout << "bind function failed with error: " << WSAGetLastError() << endl;
    closesocket(serv_sd);
    WSACleanup();
    return 1;
}

// call listen() function to let socket enter listen mode
listen(serv_sd, 5);

cli_len = sizeof(cli);

// accept connect
cout << "Server is waiting for client." << endl;
cli_sd = accept(serv_sd, (LPSOCKADDR) &cli, &cli_len);

if (cli_sd == SOCKET_ERROR) {
    cout << "can't accept: " << WSAGetLastError() << endl;
}

while (true) {
    // receive from client
    rec_len = recv(cli_sd, str, MAXLINE, 0);

    cout << "Recv:" << str << endl;
    //cout << "server recv: " << str << "(" << rec_len << " bytes)" << endl;

    if (rec_len == SOCKET_ERROR) {
```

```
        cout << "receive error: " << WSAGetLastError() << endl;
        break;
    }

    // (1) time
    if (!strcmp(str, "time")) {
        auto time = chrono::system_clock::now();

        time_t now_time = std::chrono::system_clock::to_time_t(time);

        string result_str = ctime(&now_time);

        strcpy(str_r, result_str.c_str());

        cout << "Send time to client." << endl;
    } else if (rec_len > 8) {
        string recv_str = str;

        //cout << "rec_len" << rec_len << endl;

        if (recv_str.substr(0, 6) == "ask://") {
            // (2) ask
            recv_str = recv_str.substr(6);
            strcpy(str_r, recv_str.c_str());


            struct in_addr sAddr{};
            LPHOSTENT hp;

            sAddr.s_addr = inet_addr(recv_str.c_str());

            hp = gethostbyaddr((LPSTR) &sAddr, sizeof(sAddr), AF_INET);

            if (hp == nullptr) {
                // other
                cout << "get hp error, code:" << WSAGetLastError() << endl;
                strcpy(str_r, "");
```

```
        cout << "Send nothing." << endl;
    } else {
        //printf("host name:%s\n", hp->h_name);
        //printf("host nickname:%s\n", hp->h_aliases[0]);
        // printf("host IP:%s\n\n", inet_ntoa(*(LPIN_ADDR) (hp->h_addr)));
        strcpy(str_r, (hp->h_name));

        cout << "Send domain name to client." << endl;
    }
} else if (recv_str.substr(0, 7) == "http://") {
    // (3) http
    recv_str = recv_str.substr(7);
    strcpy(str_r, recv_str.c_str());

    // Open a TCP socket
    http_sd = socket(AF_INET, SOCK_STREAM, 0);

    // Prepare for connect.
    // Include sockaddr_ing struct (serv)
    http.sin_family = AF_INET;

    // server's ip address
    http.sin_addr.s_addr = inet_addr(recv_str.c_str());

    // server's port number
    // htons: host to network
    http.sin_port = htons(80);

    int conn_status = connect(http_sd, (LPSOCKADDR) &http, sizeof(http));

    if (conn_status == SOCKET_ERROR) {
        cout << "connect error: " << WSAGetLastError() << endl;
        closesocket(http_sd);
        WSACleanup();
        break;
    }

    string request = "GET / HTTP/1.1\r\nHost: " + recv_str + "\r\nConnection:
```

```
close\r\n\r\n";

                int send_status = send(http_sd, request.c_str(), int(request.size()) + 1, 0);

                if (send_status == SOCKET_ERROR) {
                    cout << "send error: " << WSAGetLastError() << endl;
                    break;
                }

                char str_http[MAXLINE] = "";

                int rec_http_len = recv(http_sd, str_http, MAXLINE, 0);

                if (rec_http_len == SOCKET_ERROR) {
                    cout << "receive error: " << WSAGetLastError() << endl;
                    break;
                }

                strcpy(str_r, str_http);



                cout << "Send web page to client." << endl;

            } else {
                // other

                strcpy(str_r, "");

                cout << "Send nothing." << endl;
            }
        } else if (!strcmp(str, "end")) {
            // (4) end

            strcpy(str_r, "Test completes.");

            cout << "Complete service for the client." << endl;
        } else {
            // other
```

```
            strcpy(str_r, "");

            cout << "Send nothing." << endl;
        }

        // send from server
        send_len = send(cli_sd, str_r, int(strlen(str_r) + 1), 0);

        if (send_len == SOCKET_ERROR) {
            cout << "send error: " << WSAGetLastError() << endl;
            break;
        }

        //cout << "server reply: " << str_r << "(" << send_len << " bytes)" << endl;

        if (!strcmp(str, "end")) {
            closesocket(cli_sd);

            // call listen() function to let socket enter listen mode
            listen(serv_sd, 5);

            cli_len = sizeof(cli);

            // accept connect
            cout << endl << "Server is waiting for client." << endl;
            cli_sd = accept(serv_sd, (LPSOCKADDR) &cli, &cli_len);

            if (cli_sd == SOCKET_ERROR) {
                cout << "can't accept: " << WSAGetLastError() << endl;
            }
        }
    }

    // close TCP socket
    closesocket(serv_sd);

    // finish "WinSock DLL"
```

```
    WSACleanup();

    //system("pause");

    return 0;
}
```

[mid_2_client1.cpp]
GitHub:
https://github.com/linwebs/network_programming/blob/main/mid_2_client1.cpp

```
/*
 * NCYU 109 Network Programming mid 2 client 1
 * Created by linwebs on 2021/4/19.
 */
#include <iostream>
#include <winsock.h>

#define MAXLINE 1024

using namespace std;

int main() {
    SOCKET sd;
    struct sockaddr_in serv{};

    string input;
    char str[MAXLINE] = "";
    char str_r[MAXLINE];

    WSADATA wsadata;

    // server's ip address
    const char server_ip[16] = "127.0.0.1";

    // server's port number
    u_short server_port = 5678;

    // receive bytes
    int rec_len;

    // connect status
    int conn_status;

    // Call WSAStartup() to Register "WinSock DLL"
    WSAStartup(0x101, (LPWSADATA) &wsadata);
```

```
// Open a TCP socket
sd = socket(AF_INET, SOCK_STREAM, 0);

// Prepare for connect.
// Include sockaddr_ing struct (serv)
serv.sin_family = AF_INET;

// server's ip address
serv.sin_addr.s_addr = inet_addr(server_ip);

// server's port number
// htons: host to network
serv.sin_port = htons(server_port);

// connect to server
conn_status = connect(sd, (LPSOCKADDR) &serv, sizeof(serv));

if (conn_status == SOCKET_ERROR) {
    cout << "connect function failed with error: " << WSAGetLastError() << endl;
    closesocket(sd);
    WSACleanup();
    return 1;
}

cout << "Connect to server." << endl;

while (true) {
    cout << "[Client 1]:";

    getline(cin, input);

    strcpy(str, input.c_str());

    // send to server
    send(sd, str, int(strlen(str) + 1), 0);
    //cout << "client 1  傳送: " << str << "(" << strlen(str) + 1 << " bytes)" << endl;
```

```
        // receive from server
        rec_len = recv(sd, str_r, MAXLINE, 0);
        cout << "[Client 2]:" << str_r << endl;

        string recv_str = str_r;

        if(recv_str.substr(0, 6) == "client") {
            break;
        }
    }

    // close TCP socket
    closesocket(sd);

    // finish "WinSock DLL"
    WSACleanup();

    //system("pause");

    return 0;
}
```

[mid_2_client2.cpp]
GitHub:
https://github.com/linwebs/network_programming/blob/main/mid_2_client2.cpp

```
/*
 * NCYU 109 Network Programming mid 2 client 2
 * Created by linwebs on 2021/4/19.
 */
#include <iostream>
#include <winsock.h>

#define MAXLINE 1024

using namespace std;

int main() {
    SOCKET sd;
    struct sockaddr_in serv{};

    string input = "";
    char str[MAXLINE] = "";
    char str_r[MAXLINE];

    WSADATA wsadata;

    // server's ip address
    const char server_ip[16] = "127.0.0.1";

    // server's port number
    u_short server_port = 5678;

    // receive bytes
    int rec_len;

    // connect status
    int conn_status;

    // Call WSAStartup() to Register "WinSock DLL"
    WSAStartup(0x101, (LPWSADATA) &wsadata);
```

```
// Open a TCP socket
sd = socket(AF_INET, SOCK_STREAM, 0);

// Prepare for connect.
// Include sockaddr_ing struct (serv)
serv.sin_family = AF_INET;

// server's ip address
serv.sin_addr.s_addr = inet_addr(server_ip);

// server's port number
// htons: host to network
serv.sin_port = htons(server_port);

// connect to server
conn_status = connect(sd, (LPSOCKADDR) &serv, sizeof(serv));

if (conn_status == SOCKET_ERROR) {
    cout << "connect function failed with error: " << WSAGetLastError() << endl;
    closesocket(sd);
    WSACleanup();
    return 1;
}

cout << "Connect to server." << endl;

while (true) {
    // receive from server
    rec_len = recv(sd, str_r, MAXLINE, 0);
    cout << "[Client 1]:" << str_r << endl;

    string recv_str = str_r;

    if(recv_str.substr(0, 6) == "client") {
        break;
    }
```

```
        cout << "[client 2]:";

        getline(cin, input);

        strcpy(str, input.c_str());

        // send to server
        send(sd, str, int(strlen(str) + 1), 0);
        //cout << "client 2 傳送: " << str << "(" << strlen(str) + 1 << " bytes)" << endl;
    }

    // close TCP socket
    closesocket(sd);

    // finish "WinSock DLL"
    WSACleanup();

    //system("pause");

    return 0;
}
```

[mid_2_server.cpp]
GitHub:
https://github.com/linwebs/network_programming/blob/main/mid_2_server.cpp

```
/*
 * NCYU 109 Network Programming mid 2 server
 * Created by linwebs on 2021/4/19.
 */
#include <iostream>
#include <cstring>
#include <winsock.h>

#define MAXLINE 1024

using namespace std;

int main() {
    SOCKET serv_sd;
    SOCKET cli1_sd;
    SOCKET cli2_sd;
    int cli1_len;
    int cli2_len;

    char str[MAXLINE];

    string last_char = "";

    bool is_init = true;

    int client1_error_time = 0;
    int client2_error_time = 0;

    struct sockaddr_in serv{};
    struct sockaddr_in cli1{};
    struct sockaddr_in cli2{};

    WSADATA wsadata;

    // server's ip address
```

```
const char server_ip[16] = "127.0.0.1";

// server's port number
u_short server_port = 5678;

// receive bytes
int rec_len;

// send bytes
int send_len;

// bind status
int bind_status;

// Call WSAStartup() to Register "WinSock DLL"
WSAStartup(0x101, (LPWSADATA) &wsadata);

// Open a TCP socket
serv_sd = socket(AF_INET, SOCK_STREAM, 0);

// Prepare for connect.
// Include sockaddr_ing struct (serv)
serv.sin_family = AF_INET;

// server's ip address
serv.sin_addr.s_addr = inet_addr(server_ip);

// server's port number
// htons: host to network
serv.sin_port = htons(server_port);

// bind
bind_status = bind(serv_sd, (LPSOCKADDR) &serv, sizeof(serv));

if (bind_status == SOCKET_ERROR) {
    cout << "bind function failed with error: " << WSAGetLastError() << endl;
    closesocket(serv_sd);
    WSACleanup();
```

```
    return 1;
}

// call listen() function to let socket enter listen mode
listen(serv_sd, 5);

cli1_len = sizeof(cli1);
cli2_len = sizeof(cli2);

cout << "Server waits." << endl;

// accept connect
//cout << "等待 client 1 連線" << endl;
cli1_sd = accept(serv_sd, (LPSOCKADDR) &cli1, &cli1_len);
cout << "One client connects." << endl;

//cout << "等待 client 2 連線" << endl;
cli2_sd = accept(serv_sd, (LPSOCKADDR) &cli2, &cli2_len);
cout << "Two clients connect." << endl;

while (true) {
    // receive from client 1
    while (true) {
        rec_len = recv(cli1_sd, str, MAXLINE, 0);

        if (rec_len <= 0) {
            break;
        }

        cout << "1->2:" << str << endl;

        if (is_init) {
            last_char = str;

            last_char = last_char.substr(last_char.size() - 3, 3);

            string output = "正確，尾字是";
            output += last_char;
```

```
        cout << output << endl;

        is_init = false;

        // send msg from server to client 2
        send_len = send(cli2_sd, str, int(strlen(str) + 1), 0);

        if (send_len == SOCKET_ERROR) {
            cout << "[1->2] send error" << WSAGetLastError() << endl;
        }

        break;
    } else {
        string first_char = str;
        first_char = first_char.substr(0, 3);

        if (last_char == first_char) {
            last_char = str;

            last_char = last_char.substr(last_char.size() - 3, 3);

            string output = "正確，尾字是";
            output += last_char;

            cout << output << endl;

            // send msg from server to client 2
            send_len = send(cli2_sd, str, int(strlen(str) + 1), 0);

            if (send_len == SOCKET_ERROR) {
                cout << "[1->2] send error" << WSAGetLastError() << endl;
            }

            break;
        } else {
            string output = "沒有對正確，client1 犯規一次";
```

```
            if (++client1_error_time >= 3) {
                output = "client1 犯規超過三次，client1 輸了";
            }

            // send msg from server to client 2
            send_len = send(cli1_sd, output.c_str(), int(output.size() + 1), 0);

            if (send_len == SOCKET_ERROR) {
                cout << "[client->1] send error" << WSAGetLastError() << endl;
            }

            cout << output << endl;

            if (client1_error_time >= 3) {
                break;
            }
        }
    }
}

if (client1_error_time >= 3) {
    //cout << "client1 犯規超過三次，client2 輸了" << endl;

    strcpy(str, "client1 犯規超過三次，client1 輸了");

    // send msg from server to client 2
    send_len = send(cli2_sd, str, int(strlen(str) + 1), 0);

    if (send_len == SOCKET_ERROR) {
        cout << "[1->2] send error" << WSAGetLastError() << endl;
    }

    break;
}


// receive from client 2
while (true) {
```

```
rec_len = recv(cli2_sd, str, MAXLINE, 0);

if (rec_len <= 0) {
    break;
}

cout << "2->1:" << str << endl;

string first_char = str;
first_char = first_char.substr(0, 3);

if (last_char == first_char) {
    last_char = str;

    last_char = last_char.substr(last_char.size() - 3, 3);

    string output = "正確，尾字是";
    output += last_char;

    cout << output << endl;

    // send msg from server to client 1
    send_len = send(cli1_sd, str, int(strlen(str) + 1), 0);

    if (send_len == SOCKET_ERROR) {
        cout << "[2->1] send error" << WSAGetLastError() << endl;
    }

    break;
} else {
    string output = "沒有對正確，client2 犯規一次";

    if (++client2_error_time >= 3) {
        output = "client2 犯規超過三次，client2 輸了";
    }

    // send msg from server to client 2
    send_len = send(cli2_sd, output.c_str(), int(output.size() + 1), 0);
```

```
            if (send_len == SOCKET_ERROR) {
                cout << "[client->2] send error" << WSAGetLastError() << endl;
            }

            cout << output << endl;

            if (client2_error_time >= 3) {
                break;
            }
        }
    }

    if (client2_error_time >= 3) {
        //cout << "client2 犯規超過三次，client2 輸了" << endl;

        strcpy(str, "client2 犯規超過三次，client2 輸了");

        // send msg from server to client 2
        send_len = send(cli1_sd, str, int(strlen(str) + 1), 0);

        if (send_len == SOCKET_ERROR) {
            cout << "[2->1] send error" << WSAGetLastError() << endl;
        }

        break;
    }
}

// close TCP socket
closesocket(serv_sd);
closesocket(cli1_sd);
closesocket(cli2_sd);

// finish "WinSock DLL"
WSACleanup();

//system("pause");
```

```
    return 0;
}
```