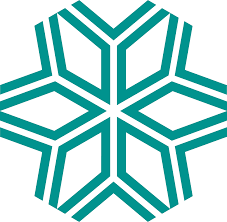
雲林科技大學

網路最佳化: Non-blocking socket多人聊天室

學生:薛華慶

學號: M10917055

目錄

[程式流程 0](#_Toc69566966)

[程式解說 3](#_Toc69566967)

[Server 3](#_Toc69566968)

[Client 3](#_Toc69566969)

[結論 4](#_Toc69566970)

[參考文獻 4](#_Toc69566971)

[Code 5](#_Toc69566972)

[Server 5](#_Toc69566973)

[Client Writer 13](#_Toc69566974)

[Client Reader 16](#_Toc69566975)

# 程式流程

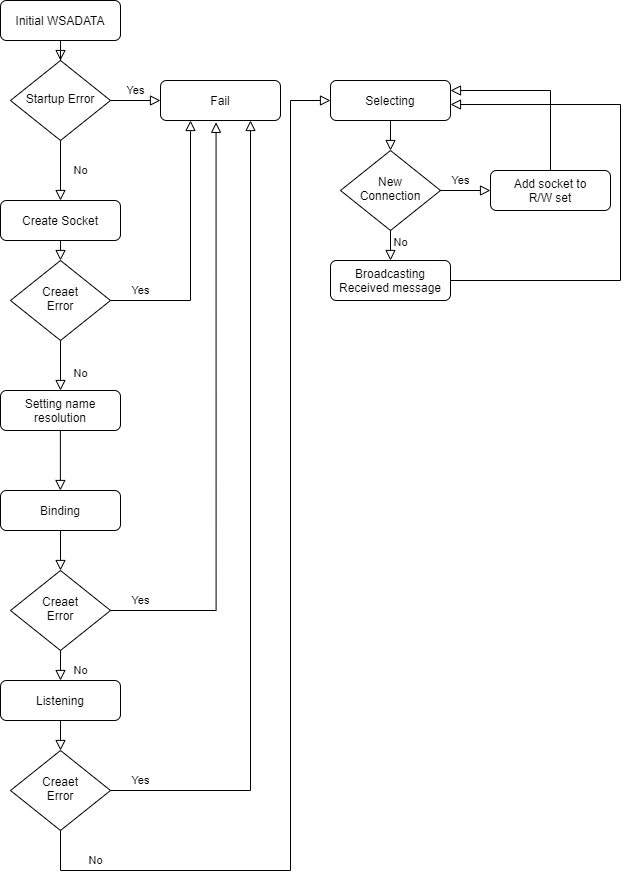


圖 1 Server program flow

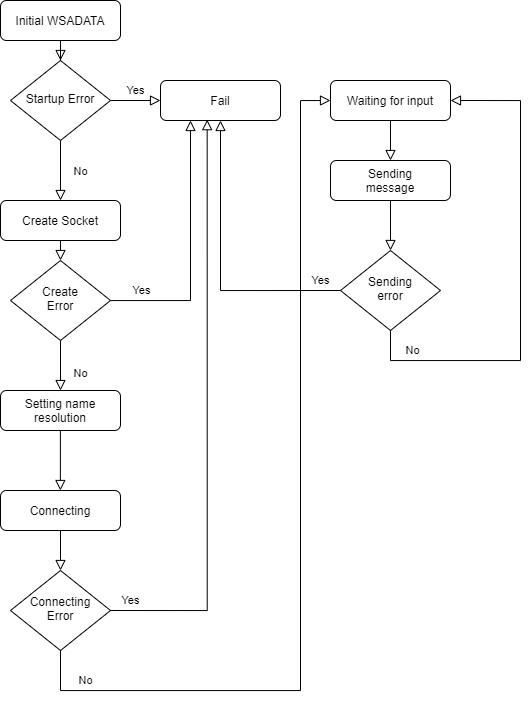


圖 2 Client Writer flow

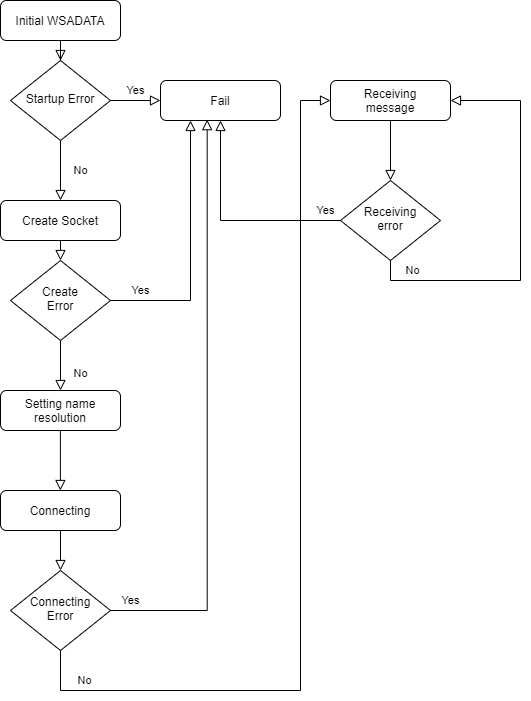


圖 3 Client Reader flow

# 程式解說

### Server

如圖一所示，首先必須將Windows WSDATA物件初始化，接著將WSDAT作為參數建立一socket物件(這裡的socket物件僅僅作為連線使用，為表示不同此處稱之為nFd)，建立完成後進入設定環節，在此步驟programmer須將server資訊，如: 連線協定、IP、Port…等資訊設定完成。設定完成後便進入綁定程序，所謂綁定就是透過上述建立完成之server設定使socket物件套用此設定。接著將socket設定成監聽模式，如此一來每當有使用者與server端建立連線，server端在該模式下即可透過accept()接受使用者之連線並進行相對作業。

上述流程為單一作業之server，稱之為阻塞式socket。由於本次實驗目標為一多人聊天室，而阻塞式socket本身在同一時間只能接受一位使用者之連線，因此本次實驗使用非阻塞式socket作為server架構。所謂非阻塞式socket就是將socket透過陣列宣告建立一定數量之socket，此處宣告之socket陣列與上nFd不同，在此處宣告之socket是用來記錄存放每位使用者之socket而nFd是為了暫時存放使用者連線之socket。而對於讀取接收使用者訊息，非阻塞式socket與阻塞式socket不同在於，非阻塞式socket運用select作為訪問是否有訊息傳入之行為，而select會將使用者socket存放在於set上，set又可細分為讀取set以及寫入set ， 因此在以下源碼中可看見，server在運行迴圈內會不斷呼叫select函數，用以偵測有無訊息傳入。假設此時有一使用者傳遞訊息指server，server可透過select得知有訊息傳入，即可透過原先宣告之socket陣列中提取當前使用者socket並透過recv函數接收訊息，並對陣列內所有socket進行廣播使所有使用者皆可收到訊息。

### Client

不論是非阻塞式socket或是阻塞是socket，client通常不會有太多變化。在此次實驗中，因為每當使用者等待輸入時，console畫面會因scanf ( ) 卡住而無法刷新聊天訊息，為此本次實驗為使用者提供兩支程式分別為，傳輸用及讀取用，

程式流程如圖二及圖三。Client端建立連線前的初始化作業與server端相同，直至創建socket物件後，client使用connect函數與server要求建立連線(與server不同，在server中使用binding進行socket綁定)。 在讀取用程式中client會進入迴圈中不斷透過recv函數接收server傳送之資訊，而傳輸用程式則是不段透過scanf函數讀取使用者訊息輸入並將該輸入字串透過send函數傳送至server。

# 結論

透過此次實驗成功做出一非阻塞式socket多人連線聊天系統，但由於client等待輸入過程中會因為輸入函數造成畫面卡住無法更新聊天紀錄，因此本次實驗提供兩支程式供使用者使用此為方法一，方法二則為利用圖形化介面將client端接收及傳送兩種動作分別寫作成API形式，如此一來使用者介面中即可透過按鈕或是圖形化介面將兩種動作平行化處理。相關Demo及程式碼請連結至以下相關連結。

Demo影片: <https://www.youtube.com/watch?v=DpEPXVtFCgs>

程式原始碼: <https://github.com/CodeMachine0121/Non-blocking-socket-multi-chatRoom>

# 參考文獻

Window Socket 筆記: <http://www.cchsu.com/arthur/prg_bg5/winsock.htm>

WinSock tutorial : <https://www.binarytides.com/winsock-socket-programming-tutorial/>

# Code

### Server

#include <stdio.h>

#include<winsock2.h>

#pragma comment(lib, "ws2\_32") // windows socket library

#define WSA\_VERSION MAKEWORD(2, 2)

#define SOCK\_SIZE 20

struct client\_informations{

    char \*host;

    int port;

};

int SocketWrite(SOCKET sock, fd\_set writerset, char \*message, int len){

    if(FD\_ISSET(sock, &writerset)){

         // cut fraction of buffer

        //char bufsend[] = "say: Hello from server\n",);

        // 會一直跑回圈 要看

        printf("msg: %s\n", message);

        send(sock, message, len,0);

        return 1;

    }message="";

    return 0;

}

int for\_SocketWrite(SOCKET socks[], fd\_set writerset, char \*message, int len,int times){

    for (int i = 0; i < times; i++)

    {

        SocketWrite(socks[i+1],writerset,message,len);

        printf("Send message to sock[%d]\n", i+1);

    }

    return 0;

}

int main(void){

    char \*message="";

    WSADATA wsa={0};

    // initialize

    printf("\nInitializing.....\n");

    if(WSAStartup(MAKEWORD(2,2),&wsa) != 0){

        printf("Failed. Error Code: %d\n", WSAGetLastError());

        return 1;

    }

    printf("Initialized\n");

// creating socket

    int nFd = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    if(nFd==INVALID\_SOCKET){

        printf("Error creating socket, ec:%d\n", WSAGetLastError());

        return -1;

    }

// name resolution

    struct sockaddr\_in oAddr;

    oAddr.sin\_family=AF\_INET;

    oAddr.sin\_addr.s\_addr = INADDR\_ANY;

    oAddr.sin\_port= htons(8881);

// binding

    if(bind(nFd, (struct sockaddr\*)&oAddr, sizeof(oAddr)) == SOCKET\_ERROR){

        printf("Error binding sokcet, ec:%d\n", WSAGetLastError());

        closesocket(nFd);

        return -1;

    }

// listening

    if(listen(nFd, SOCK\_SIZE) == SOCKET\_ERROR){

        printf("Error listening, ec: %d\n", WSAGetLastError());

        closesocket(nFd);

        return -1;

    }

// select

    u\_long  unblock=1;

    if(ioctlsocket(nFd, FIONBIO, &unblock) == SOCKET\_ERROR){

        printf("ioctlsocket(acp) erroe, ec:%d\n", WSAGetLastError());

        return -1;

    }

    fd\_set ORSet;

    fd\_set OWSet;

    FD\_ZERO(&ORSet);

    FD\_ZERO(&OWSet);

    FD\_SET(nFd,&ORSet);

    FD\_SET(nFd,&OWSet);

    SOCKET socks[64];

    struct client\_informations clis[64];

    socks[0]=nFd;

    clis[0].host="127.0.0.1";

    clis[0].port=80;

    int TotalSockets=1;

    int c,j=0;

    while(1){

        fd\_set readerset = ORSet;

        fd\_set writerset = OWSet;

        struct timeval tv;

        tv.tv\_sec = 1;

        tv.tv\_usec = 0;

        int res = select(0,&readerset,&writerset,NULL,&tv);

        if(res == SOCKET\_ERROR){

            printf("select error, ec:%d\n", WSAGetLastError());

            break;

        }

        if(res==0){

            continue; // no connect

        }

        int temTotalSockets = TotalSockets;

         // receive message

        int i;

        for(i=0;i<TotalSockets;i++){

            if(FD\_ISSET(socks[i], &readerset)){

                // listensocket 可讀

                if(socks[i]==nFd){ // 第一次連線

                    struct sockaddr\_in  client;

                    SOCKET acp = accept(nFd, (struct sockaddr \*)&client, &c);

                    if(acp==INVALID\_SOCKET){

                        printf("accept error, ec:%d\n", WSAGetLastError());

                        break;

                    }

                    if(ioctlsocket(acp, FIONBIO, &unblock) == SOCKET\_ERROR){

                        printf("ioctlsocket(acp) erroe, ec:%d\n", WSAGetLastError());

                        break;

                    }

                    FD\_SET(acp, &ORSet);

                    FD\_SET(acp, &OWSet);

                    //printf("temTotalSockets: %d\n", temTotalSockets);

                    socks[temTotalSockets++]=acp;

                    // get client address information and save it into struct client\_informations

                    //char\* addr\_cli = ;

                    clis[j].host = inet\_ntoa(client.sin\_addr);

                    clis[j].port = client.sin\_port;

                    printf("one connection %s:%d by socket[%d] create clis[%d]\n", clis[j].host, clis[j].port,i,j);

                    j++;

                }else{

                    // recv message

                    char buf[1024];

                    int len = recv(socks[i], buf, 1024, 0);

                    if(len==0){

                        printf("No Sleep\n");

                        //break;

                    }else if( len==SOCKET\_ERROR){;

                        //printf("Recv error, ec:%d\n", WSAGetLastError());

                        //break;

                    }else{

                        //char outbuf[len+1];

                        //memcpy(outbuf,buf,len);

                        //outbuf[len] = 0;

                        // '[',']',': '=4

                        int len = strlen(clis[i-1].host) + strlen(buf) + 1 + 4;

                        char msg[len];

                        memset(msg, '\0', len);

                        strcat(msg,"[");

                        strcat(msg,clis[i-1].host);

                        strcat(msg,"]: ");

                        strcat(msg,buf);

                        message = msg;

                        printf("recv: '%s' by socks[%d]\n",msg, i);

                        printf("from %s:%d  \n",clis[i-1].host, clis[i-1].port);

                        for\_SocketWrite(socks, writerset, message, strlen(message),j);

                        //free(outbuf);

                    }

                }

            }

        }

        TotalSockets=temTotalSockets;

    }

    closesocket(nFd);

    WSACleanup();

    return 0;

}

### Client Writer

#include <stdio.h>

#include<winsock2.h>

#pragma commnet(lib, "ws2\_32.lib") // windows socket library

int main(void){

    WSADATA wsa;

    SOCKET s;

    int rc;

    struct sockaddr\_in server;

    printf("\nWinsock initializing....\n");

    if(WSAStartup(MAKEWORD(2,2),&wsa)!=0){

        printf("Failes. Error Code: %d\n",WSAGetLastError());

        return 1;

    }

    printf("Initialized\n");

    // Create socket

    if((s=socket(AF\_INET, SOCK\_STREAM, 0))==INVALID\_SOCKET){

        printf("Could not create socket: %d\n", WSAGetLastError());

    }

    printf("Socket created\n");

    //設定連線

    server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    server.sin\_family = AF\_INET;

    server.sin\_port = htons(8881);

    // Connect to remote server

    rc = connect(s,(struct sockaddr \*)&server, sizeof(server));

    if(rc<0){

        printf("connect error\n");

        return 1;

    }

    printf("Connected\n");

    while(1){

        char \*str;

        str = (char \*)malloc(sizeof(char)\*1024);

        printf("> ");

        scanf("%s", str);

        if(send(s, str, 1024, 0)<=0){

            printf("send error\n");

            return 1;

        }

        printf("\"%s\" sent\n",str);

    }

    closesocket(s);

    WSACleanup();

    return 0;

}

### Client Reader

#include <stdio.h>

#include<winsock2.h>

#pragma commnet(lib, "ws2\_32.lib") // windows socket library

int main(void){

    WSADATA wsa;

    SOCKET s;

    int rc;

    struct sockaddr\_in server;

    printf("\nWinsock initializing....\n");

    if(WSAStartup(MAKEWORD(2,2),&wsa)!=0){

        printf("Failes. Error Code: %d\n",WSAGetLastError());

        return 1;

    }

    printf("Initialized\n");

    // Create socket

    if((s=socket(AF\_INET, SOCK\_STREAM, 0))==INVALID\_SOCKET){

        printf("Could not create socket: %d\n", WSAGetLastError());

    }

    printf("Socket created\n");

    //設定連線

    server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    server.sin\_family = AF\_INET;

    server.sin\_port = htons(8881);

    // Connect to remote server

    rc = connect(s,(struct sockaddr \*)&server, sizeof(server));

    if(rc<0){

        printf("connect error\n");

        return 1;

    }

    printf("Connected\n");

    while(1){

         // message to send

        char \*server\_reply; // message to receive

        server\_reply = (char \*) malloc(sizeof(char)\*1024);

        if( (rc = recv(s,server\_reply, 1024, 0)) == SOCKET\_ERROR){

            printf("recv error\n");

        }

        //server\_reply[1024] = ' ';

        printf(">> %s\n", server\_reply);

       // free(server\_reply);

    }

    closesocket(s);

    WSACleanup();

    return 0;

}