# 電腦網路實驗實驗報告 < Socket Programming >

姓名: 翁佳煌　　學號: 409430030

## 1. 實驗名稱

網路程式設計，寫一個會標記「已讀」訊息的 echo server，並寫一支 client 程式跟 echo server 對話。

## 2. 實驗目的

這次實驗將讓我們理解通訊協議、網路設備、網路架構中的 client/server 模型以及 TCP socket programming 的基本概念，讓我們對網路程式設計有整體的認識，並通過 getaddrinfo 和 TCP client/server 的示範演示來演示如何實現這些技術。通過本實驗的學習，將學會如何建立基於 TCP 協議的客戶端/伺服器應用程式，並掌握如何使用網路編程的基本工具和 API。

## 3. 實驗設備

Linux 作業系統之電腦。

## 4. 實驗步驟

首先看到 server 端，它會持續等待客戶端的連線，並接收、處理客戶端的訊息並回覆已讀給 client，以下是詳細 code 介紹:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

//socket
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

void error_msg(char*msg);

int setup_socket(void);

void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in*storeAddr);

void interact_with_client(int serverSock);

```

1~10 行: 引用需要用到的標頭檔，除了 c 語言常見的標頭以外，還包含 sys/socket.h、netinet/in.h、arpa/inet.h、unistd.h 等等。
12~18 行: 宣告函式，分別為 error_msg、setup_socket、setup_address、interact_with_client。

```
20 int main(int argc, char*argv[]){
21
22         char SERVER_IP[18] = {0};
23         int SERVER_PORT = 0;
24
25         int serverSock;
26         struct sockaddr_in serverAddr;
27
28         if(argc!=3){
29                 error_msg("[Usage] TCP_server SERVER_IP SERVER_PORT");
30         }
31         strcpy(SERVER_IP, argv[1]);
32         SERVER_PORT = atoi(argv[2]);
33
34         serverSock = setup_socket();
35         setup_address(SERVER_IP, SERVER_PORT, &serverAddr);
36         if(bind(serverSock, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) <0){
37                 error_msg("[ERROR] Failed to bind.");
38         }
39         listen(serverSock, 2);
40         printf("server setup\n");
41
42         interact_with_client(serverSock);
43
44         close(serverSock);
45         return 0;
46 }
```

接下來看到 main 函式。
22~26 行:定義了一些變數，包括了 SERVER_IP、SERVER_PORT、serverSock 和 serverAddr。

28~32 行:用來判斷命令行參數是否正確，如果命令行參數不是 3 個，則調用 error_msg 函數輸出錯誤信息並退出程序。否則，將 SERVER_IP 和 SERVER_PORT 的值賦值為命令行參數中指定的值。

34~46 行:呼叫 setup_socket 函數來建立一個 socket，並且將描述符存放到 serverSock 中。呼叫 setup_address 函數來設置 serverAddr 結構體的 IP 地址和端口號。使用 bind 函數將 socket 描述符和 IP 地址與端口號綁定起來，如果失敗則執行 error_msg 函數。接著使用 listen 函數開始監聽伺服器端的 socket 描述符，最多可以接受 2 個客戶端連線請求。再來呼叫 interact_with_client 函數來開始與客戶端進行互動，最後的 close(serverSock)的功能為關閉伺服器端的 socket。

```
48 void error_msg(char*msg){
49         fprintf(stderr,"%s %s\n", msg, strerror(errno));
50         exit(1);
51 }
```

48~51 行:
出現錯誤時輸出錯誤訊息到標準錯誤輸出(stderr)，然後直接結束程式執行(exit(1))。

```
53 int setup_socket(void){
54         int socketFd;
55         int sockopt= 1;
56         socketFd = socket(PF_INET, SOCK_STREAM, 0);
57         setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, &sockopt, sizeof(sockopt));
58         return socketFd;
59 }
```

再來看到 setup_socket 函數。
53~59 行:
函數內部先宣告了一個 socketFd 變數，用來存放 socket 的檔案描述符。接著，宣告一個 sockopt 變數，並將它初始化為 1。
接著，使用 socket()函數建立一個 TCP socket，並將它的檔案描述符存放在 socketFd 變數中。socket 函數的第一個參數指定 socket 的 domain，這裡使用了 PF_INET，表示使用 IPv4 協議；第二個參數指定 socket 的類型，這裡使用了 SOCK_STREAM，表示使用 TCP 協議；第三個參數指定協議，這裡使用了 0，表示使用預設協議。
接著，使用 setsockopt()函數設定 socket 的屬性。setsockopt 函數的第一個參數指定要設定的 socket；第二個參數指定要設定的屬性，這裡使用 SOL_SOCKET，表示設定的是 socket 級別的屬性；第三個參數設定了 SO_REUSEADDR ，使得當 socket 關閉時，端口可以立刻被重用。第四個參數是 sockopt 的位址；第五個參數指定 sockopt 的大小。

```
61 void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in*storeAddr){
62         storeAddr->sin_family = AF_INET;
63         storeAddr->sin_addr.s_addr = inet_addr(SERVER_IP);
64         storeAddr->sin_port =htons(SERVER_PORT);
65         return;
66 }
```

接下來看到 setup_address 函數。
61~66 行:
storeAddr->sin_family = AF_INET; 將伺服器地址的協定設為 IPv4。
storeAddr->sin_addr.s_addr = inet_addr(SERVER_IP); 用 inet_addr() 函

數將伺服器的 IP 位址轉換為網絡字節序（network byte order），然後將其儲存到伺服器地址的 sin_addr 欄位中。
storeAddr->sin_port =htons(SERVER_PORT); 用 htons() 函數將伺服器的連接埠號碼轉換為網絡字節序，然後將其儲存到伺服器地址的 sin_port 欄位中。

```
68 void interact_with_client(int serverSock){
69         int clientSock = 0;
70         struct sockaddr_in clientAddr;
71         int clientAddrLength = sizeof(clientAddr);
72 
73         char msg[BUFSIZ] = {0};
74 
75         while(1){
76 
77                 clientSock = accept(serverSock, (struct sockaddr *)&clientAddr, &clientAddrLength);
78                 printf("[INFO] Connection from %s[%d]\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
79 
80                 while(1){
81                         memset(msg, 0, sizeof(msg));
82                         if(recv(clientSock, &msg, sizeof(msg),0) <= 0){
83                                 printf("[INFO] Client disconnected.\n");
84                                 break;
85                         }
86                         printf("[CLIENT] %s\n", msg);
87 
88                         strcat(msg, "[server readed]");
89                         send(clientSock, msg, strlen(msg), 0);
90                 }
91         }
92         return;
93 }
```

最後看到 interact_with_client，負責接收 client 端的訊息以及回復。
68~93 行:
一開始宣告了一個變數 clientSock 並初始化為 0，用來接收與客戶端的連接。宣告了一個變數 msg 來接收從客戶端傳來的訊息。在 while 迴圈中，使用 accept 函數來接受來自客戶端的連接，並將客戶端的 IP 位址和連接埠號印出。接著使用一個內層 while 迴圈來持續接收客戶端傳來的訊息。使用 memset 函數將 msg 陣列中的元素全部初始化為 0，以確保接收到的訊息不會錯誤。接著使用 recv 函數來接收來自客戶端的訊息，並判斷接收到的長度是否小於等於 0，如果是則認為客戶端已經斷線，印出相關訊息後跳出內層 while 迴圈。否則，印出客戶端傳來的訊息。將固定的回應訊息串接在 msg 陣列後面，使用 send 函數將回應訊息傳回給客戶端。在內層 while 迴圈結束後，外層 while 迴圈會繼續等待下一個客戶端的連接。

---

接下來是介紹 **client 端**，它需要輸入訊息並傳送到 server 端後，要接收到 server 端送來的已讀訊息，以下是詳細 code 介紹:

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <errno.h>
5 #include <unistd.h>
6
7 //socket
8 #include <sys/socket.h>
9 #include <netinet/in.h>
10 #include <arpa/inet.h>
11 #include <netdb.h>
12
```

首先 1~12 行和上方 server 介紹的一樣 include 相關的標頭檔。

```
13 //error message
14 void error_msg(char*msg){
15         fprintf(stderr, "%s %s\n",msg,strerror(errno));
16         exit(1);
17 }
18
```

再來第 13~17 行:

出現錯誤時輸出錯誤訊息到標準錯誤輸出(stderr)，然後直接結束程式執行(exit(1))。

```
19 //setup socket with specified SERVER IP, port
20 int setup_socket(void){
21         int serverSock;
22         serverSock = socket(PF_INET, SOCK_STREAM, 0);
23         return serverSock;
24 }
25
```

19~25 行:

它使用了 socket() 函數來創建 serverSocket。socket() 函數的第一個參數是要使用的地址族（AF_INET 表示 IPv4 地址族），第二個參數是要使用的 socket 類型（SOCK_STREAM 表示 TCP 協議），第三個參數是要使用的協議（0 表示自動選擇）。

```
26 //setup server address
27 void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in*storeAddr){
28         storeAddr->sin_family = AF_INET;
29         inet_pton(AF_INET,SERVER_IP,&storeAddr->sin_addr);
30         storeAddr->sin_port = htons(SERVER_PORT);
31         return;
32 }
33
```

26~32 行:

這個函數用來設置指定 SERVER_IP 和 SERVER_PORT 的伺服器地址。它會將伺服器地址存儲到傳遞的指針 storeAddr 所指向的 sockaddr_in 結構中。

在函數中，sin_family 屬性被設置為 AF_INET，表示使用 IPv4 網絡協議。sin_addr 屬性使用 inet_pton() 函數將 SERVER_IP 轉換成網絡字節順序，並將其存儲到結構體中。sin_port 屬性使用 htons() 函數將 SERVER_PORT 轉換成網絡字節順序，並將其存儲到結構體中。

```
34 //interact with server
35 void interact_with_server(int serverSock){
36         char data[100] = {0};
37         while(1){
38                 memset(data,0,sizeof(data));
39
40                 //read message from user
41                 printf("Please enter message : ");
42                 fgets(data,sizeof(data)-1, stdin);
43                 data[strlen(data)-1]='\0';
44
45                 //send message to server
46                 send(serverSock, data, strlen(data), 0);
47                 if(recv(serverSock, data, sizeof(data), 0)<=0){
48                         break;
49                 }
50                 printf("[SERVER] %s\n", data);
51         }
52
53         return;
54 }
55
```

34~54 行:

這個函式是用來與伺服器進行互動的。它使用了一個無限迴圈，不斷讀取使用者從終端輸入的訊息，然後將該訊息傳送到伺服器。接著它使用 recv 函式來等待從伺服器回傳的訊息，一旦收到，就會在終端輸出 "[SERVER] " 和接收到的訊息。如果 recv 函式返回的是小於或等於零的值，就代表與伺服器的連接已經斷開了，此時該函式會跳出無限迴圈並結束。

```c
int main(int argc, char*argv[]){
        char SERVER_IP[18]={0};
        int SERVER_PORT = 0;

        int serverSock;
        struct sockaddr_in serverAddr;
        if(argc!=3){
                error_msg("[Usage] TCP_client SERVER_IP SERVER_PORT");
        }
        strcpy(SERVER_IP, argv[1]);
        SERVER_PORT = atoi(argv[2]);

        serverSock = setup_socket();
        setup_address(SERVER_IP,SERVER_PORT,&serverAddr);
        if(connect(serverSock,(struct sockaddr*)&serverAddr, sizeof(serverAddr))<0){
                error_msg("[ERROR] Failed to connetc to server. ");
        }

        interact_with_server(serverSock);

        close(serverSock);

        return 0;
}
```

最後是主函式:

使用者必須提供要連線的伺服器的 IP 位址和埠號。接下來，主程式會使用 setup_socket() 函式建立一個 TCP socket，並使用 setup_address() 函式設定要連線的伺服器的位址。最後，主程式使用 connect() 函式建立與伺服器的 TCP 連線，並進入 interact_with_server() 函式與伺服器進行資料交換。

在主程式中，如果命令列參數的數量不是 3 個，則會顯示一個使用說明並結束程式。如果連線失敗，主程式會顯示錯誤訊息並結束程式。如果正常建立連線，主程式會進入資料交換的迴圈。

## 5. 問題與討論

1. 有許多 socket 的函數仍然需要再更進一步理解，包括其用途以及需要的參數。
2. 如果改使用 ipv6 有許多地方需要更動，例如將 PF_INET 改成 PF_INET6，以支援 IPv6 等等。這問題在下禮拜的實驗課將會實作。
3. Socket Programming 在 Linux 和 Windows 上的差異，包括 Socket 創建方式不同、IP 位址與埠號表示方式不同、Socket 的清除方式不同等等。

## 6. 心得與感想

這次的實驗讓我更深入地了解了 Socket Programming 的概念，也學習到如何使用 TCP 協定建立 client-server 連線。在這個實驗中，我學會了如何在 C 語言中使用 Socket API，包括建立 Socket、綁定 Socket 到特定地址和埠口、聆聽連線請求、接受連線請求、建立連線、傳送和接收資料等。另外，我也學會了如何使用簡單的命令行參數來讓使用者指定 Server 的 IP 地址和埠口，以及如何在程式中動態分配和釋放資源。這些都是 socket programming 中非常重要的技能。

總而言之，這次的實驗讓我對 Socket 的概念以及使用方法有了更深入的了解，我相信這些知識對我未來的程式開發之路會非常有幫助。

## 7. 參考文獻


http://zake7749.github.io/2015/03/17/SocketProgramming/
https://www.kshuang.xyz/doku.php/programming:c:socket
https://github.com/davidleitw/socket