

# 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。並將上次實驗的 IPv4 TCP echo server 和 client 改成 IPv6。

## 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：

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include <errno.h>
5
6  //socket
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <arpa/inet.h>
10 #include <unistd.h>
11 #include <netdb.h>
12
13 void error_msg(char*msg);
14
15 int setup_socket(int IP_ver);
16
17 int getLink_local_addrSCOPEID(char* LINK_LOCAL_ADDR);
18
19 void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in6*storeAddr);
20
21 void interact_with_client(int serverSock);
```

1~11 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。

12~21 行：宣告函式，分別為 error\_msg、setup\_socket、setup\_address、getLink\_local\_addrSCOPEID、interact\_with\_client。

```
23 int main(int argc, char*argv[]){
24     //arguments
25     char SERVER_IP[50] = {0};
26     int SERVER_PORT = 0;
27
28     //socket
29     int serverSock;
30     struct sockaddr_in6 serverAddr;
31
32     if(argc!=3){
33         error_msg("[Usage] TCP_server SERVER_IP SERVER_PORT");
34     }
35     strcpy(SERVER_IP, argv[1]);
36     SERVER_PORT = atoi(argv[2]);
37
38     serverSock = setup_socket(6);
39     setup_address(SERVER_IP, SERVER_PORT, &serverAddr);
40     if(bind(serverSock, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0){
41         error_msg("[ERROR] Failed to bind.");
42     }
43     listen(serverSock, 2);
44     printf("server setup\n");
45
46     interact_with_client(serverSock);
47
48     close(serverSock);
49     return 0;
50 }
```

23~50 行：

這段程式碼是主程式 main，首先宣告了變數 SERVER\_IP 和 SERVER\_PORT，並初始化為 0。接下來宣告了一個整數型別的變數 serverSock 和一個結構體型別的變數 serverAddr。透過 argc 和 argv 判斷傳入的參數是否符合程式所需的兩個參數，如果不符合則輸出使用說明。之後，將 argv[1] 指向的字串複製到 SERVER\_IP 陣列中，並將 argv[2] 轉換成整數型別的 SERVER\_PORT。接著，呼叫 setup\_socket 函數，傳入 6 作為 IP 版本的參數，取得一個 socket 描述符並指派給 serverSock 變數。之後呼叫 setup\_address 函數，傳入 SERVER\_IP、SERVER\_PORT 和 &serverAddr 三個參數，該函數設置了 serverAddr 的成員變數。最後，呼叫 bind 函數，綁定 socket 描述符和 serverAddr 結構體，如果綁定失敗，輸出錯誤信息並退出程式。接著呼叫 listen 函數，告訴操作系統這是一個服務器端的 socket 並設置可以同時連接的客戶端的最大數量為 2。最後輸出 "server setup"，表示服務器準備就緒，準備開始接收客戶端的連接請求。最後呼叫 interact\_with\_client 函數，傳入 serverSock 描述符，開始和客戶端進行交互。交互結束後，關閉 socket 描述符並退出程式。

```

52 void error_msg(char*msg){
53     fprintf(stderr,"%s %s\n", msg, strerror(errno));
54     exit(1);
55 }

```

52~55 行：

用來輸出錯誤訊息到標準錯誤輸出 `stderr`。該函式接受一個指向字串的指標 `msg` 作為參數，代表需要輸出的錯誤訊息。

在函式內部，使用 `fprintf` 函式將 `msg` 和使用 `strerror` 函式獲取的錯誤訊息（由 `errno` 變數表示）格式化輸出到標準錯誤輸出中，然後使用 `exit` 函式退出程式並返回錯誤碼 1，以便告知呼叫該函式的程式出現了錯誤。

```

57 int setup_socket(int IP_ver){
58     int socketFd;
59     int sockopt = 1;
60     socketFd = socket(IP_ver==4 ? PF_INET : PF_INET6 , SOCK_STREAM, 0);
61     setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, &sockopt, sizeof(sockopt));
62     return socketFd;
63 }

```

57~63 行：

這個函數是用來創建一個 socket 的，接受一個整數參數 `IP_ver`，用於指定要創建的 socket 的 IP 地址版本。如果 `IP_ver` 為 4，則使用 `PF_INET` 創建一個 IPv4 socket；如果 `IP_ver` 為 6，則使用 `PF_INET6` 創建一個 IPv6。

接下來，創建一個名為 `sockopt` 的整數變量，並將其設置為 1。然後使用 `setsockopt()` 函數將 `SO_REUSEADDR` 選項設置為 socket。這將使操作系統在 socket 關閉後立即釋放 socket 的本地地址，以便在同一地址上重新綁定 socket 時不會發生地址已被使用的錯誤。最後，返回 `socketFd`。

```

67 void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in6*storeAddr
68     char SERVER_IP_CHR[50]={0};
69     strncpy(SERVER_IP_CHR,SERVER_IP,(size_t)(strchr(SERVER_IP,'%')-SERVER_IP));
70
71     storeAddr->sin6_family = AF_INET6; // AF_INET6
72     inet_pton(AF_INET6,SERVER_IP_CHR,&storeAddr->sin6_addr);
73     storeAddr->sin6_port =htons(SERVER_PORT);
74     storeAddr->sin6_scope_id=getLink_local_addrSCOPEID(SERVER_IP);
75
76     return;
77 }

```

67~77 行：

這段程式碼的功能是為了設定伺服器端的位址資訊，其中需要傳入 SERVER\_IP（伺服器 IP 位址）與 SERVER\_PORT（伺服器端的連接埠號碼），以及一個存儲設定好的位址的結構指標 storeAddr。

首先，程式碼會先宣告一個大小為 50 的字元陣列 SERVER\_IP\_CHR 並初始化為 0。接著，使用 strncpy() 函式將 SERVER\_IP 中的字元複製到 SERVER\_IP\_CHR 中，直到遇到 '%' 符號為止，這是為了去除 IP 位址中可能存在的範圍識別碼 (scope identifier)。

然後，程式碼會設定存儲位址的結構指標 storeAddr 中的成員。storeAddr->sin6\_family 被設定為 AF\_INET6 表示使用 IPv6 協議。inet\_pton() 函式可以將字串格式的 IPv6 位址轉換成網路字節序的二進制形式，將其儲存到 storeAddr->sin6\_addr 中。storeAddr->sin6\_port 被設定為 SERVER\_PORT 的值，並且使用 htons() 函式進行大小端轉換，將其轉換為網路字節序。

最後，使用 getLink\_local\_addrSCOPEID() 函式來取得該位址的範圍識別碼，將其存儲到 storeAddr->sin6\_scope\_id 中。最終設定好的位址結構指標 storeAddr 就可以被用來綁定到 socket 上了。

```

79 void interact_with_client(int serverSock){
80     int clientSock = 0;
81     struct sockaddr_in6 clientAddr;
82     int clientAddrLength = sizeof(clientAddr);
83
84     char clientAddr_str[50]={0};
85
86     char msg[BUFSIZ] = {0};
87
88     while(1){
89
90         clientSock = accept(serverSock, (struct sockaddr *)&clientAddr, &clientAddrLength);
91         printf("[INFO] Connection from %s[%d]\n", inet_ntop(AF_INET6,&clientAddr.sin6_addr,clientAddr_str,sizeof(clientAddr_str)), ntohs(clientAddr.sin6_port));
92
93         while(1){
94             memset(msg, 0, sizeof(msg));
95             if(recv(clientSock, &msg, sizeof(msg),0) <= 0){
96                 printf("[INFO] Client disconnected.\n");
97                 break;
98             }
99             printf("[CLIENT] %s\n", msg);
100
101             strcat(msg, "[server readed]");
102             send(clientSock, msg, strlen(msg), 0);
103         }
104     }
105     return;
106 }

```

79~106 行:

這個函式 `interact_with_client` 是用來處理與客戶端之間的互動。在函式開始時，會接受一個參數 `serverSock`，這個參數是代表伺服器端的 socket，也就是在主函式中所建立的那個。接著，宣告一個 `clientSock` 變數來儲存與客戶端連線的 socket，並且也宣告一個 `clientAddr` 變數，用來儲存客戶端的地址。透過 `accept` 函式可以接受客戶端的連線，同時取得客戶端的地址，並且建立一個新的 socket 來處理和這個客戶端的連線。

接著進入無窮迴圈，直到接受到客戶端的斷線訊息為止。在這個迴圈中，會先宣告一個 `msg` 變數來儲存從客戶端傳來的訊息，然後使用 `recv` 函式來接收客戶端的訊息。如果接收到的長度小於等於 0，表示客戶端已經斷線了，此時會顯示一個訊息，並且跳出迴圈。否則就會顯示客戶端傳來的訊息，並且在訊息後面加上一個字串 "[server readed]"，然後透過 `send` 函式把這個訊息送回給客戶端。

總結來說，這個函式主要是用來處理和客戶端的連線，接收客戶端的訊息，並且回應一些訊息給客戶端。



```

109 int getLink_local_addrSCOPEID(char* LINK_LOCAL_ADDR){
110     int scope_ID=-1;
111
112     struct addrinfo hints,*info;
113
114     memset(&hints,0,sizeof(hints));
115
116     hints.ai_flags=AI_NUMERICHOST;
117
118     if(getaddrinfo(LINK_LOCAL_ADDR,NULL,&hints,&info)==0){
119         struct sockaddr_in6 *sin6_info=(struct sockaddr_in6*)info->ai_addr;
120
121         scope_ID=sin6_info->sin6_scope_id;
122         freeaddrinfo(info);
123     }
124     return scope_ID;
125 }

```

109~125 行：

這個函式是用來從給定的 Link-local IPv6 位址中提取其對應的 Scope ID。在 IPv6 網路中，Link-local IPv6 位址是一種在單個鏈路範圍內使用的位址，它的格式是 fe80::/10。Scope ID 則是一個用於區分不同網路界面（例如網卡）的整數值。

函式接收一個 Link-local IPv6 位址的字串作為參數，然後使用 getaddrinfo() 函式將其轉換為一個 addrinfo 結構，然後從中提取 Scope ID。該函式使用了 struct sockaddr\_in6 類型，該類型表示 IPv6 地址和端口號。在這個函式中，為了讓 getaddrinfo() 函式可以識別 Link-local IPv6 位址，使用了 ai\_flags 參數設置 AI\_NUMERICHOST 標誌。最後，該函式返回提取的 Scope ID。

---

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 void error_msg(char*msg);
13 int setup_socket(int IP_ver);
14 void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in6*storeAddr);
15 void interact_with_server(int serverSock);
16 int getLink_local_addrSCOPEID(char* LINK_LOCAL_ADDR);
```

1~16 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，並宣告函式，分別為 `error_msg`、`setup_socket`、`setup_address`、`getLink_local_addrSCOPEID`、`interact_with_server`。

```
17 //error message
18 void error_msg(char*msg){
19     fprintf(stderr, "%s %s\n",msg,strerror(errno));
20     exit(1);
21 }
```

```
23 //setup socket with specified SERVER IP, port
24 int setup_socket(int IP_ver){
25     int serverSock;
26     serverSock = socket(IP_ver==4 ? PF_INET : PF_INET6 , SOCK_STREAM, 0);
27     return serverSock;
28 }
```

```
29
30 //setup server address
31 void setup_address(char*SERVER_IP, int SERVER_PORT, struct sockaddr_in6*storeAddr){
32     char SERVER_IP_CHR[50]={0};
33     strncpy(SERVER_IP_CHR,SERVER_IP,(size_t)(strchr(SERVER_IP,'%')-SERVER_IP));
34     storeAddr->sin6_family = AF_INET6;
35     inet_pton(AF_INET6,SERVER_IP_CHR,&storeAddr->sin6_addr);
36     storeAddr->sin6_port = htons(SERVER_PORT);
37     storeAddr->sin6_scope_id= getLink_local_addrSCOPEID(SERVER_IP);
38     return;
39 }
40
```

17~40 行：這幾行都與 **server** 端的部分一模一樣，這裡就不再重複敘述一次。

```

41 //interact with server
42 void interact_with_server(int serverSock){
43     char data[100] = {0};
44     while(1){
45         memset(data,0,sizeof(data));
46
47         //read message from user
48         printf("Please enter message : ");
49         fgets(data,sizeof(data)-1, stdin);
50         data[strlen(data)-1]='\0';
51
52         //send message to server
53         send(serverSock, data, strlen(data), 0);
54         if(recv(serverSock, data, sizeof(data), 0)<=0){
55             break;
56         }
57         printf("[SERVER] %s\n", data);
58     }
59
60     return;
61 }

```

41~61 行：也與 server 端的 `interact_with_client` 很類似，這個函數 `interact_with_server` 是用來和服務器進行交互的。函數會一直進入一個無限迴圈，並且在每次迴圈中等待用戶輸入信息，然後將信息發送給服務器。如果成功發送信息，則函數會等待接收服務器回傳的信息，並在控制台上印出服務器回傳的信息。如果接收失敗，函數就會跳出迴圈並結束函數。

```

87 int getlink_local_addrSCOPEID(char* LINK_LOCAL_ADDR){
88     int scope_ID= -1;
89     struct addrinfo hints,*info;
90     memset(&hints,0,sizeof(hints));
91     hints.ai_flags=AI_NUMERICHOST;
92     if(getaddrinfo(LINK_LOCAL_ADDR,NULL,&hints,&info)==0){
93         struct sockaddr_in6 *sin6_info = (struct sockaddr_in6*)info->ai_addr;
94         scope_ID = sin6_info->sin6_scope_id;
95         freeaddrinfo(info);
96     }
97     return scope_ID;
98 }
99

```

87~99 行：

這個函數主要是用來從傳入的 `LINK_LOCAL_ADDR` 參數中獲取該 IPv6 位址的 `SCOPE_ID`，即該位址對應的網路接口編號。在函數中，首先使用 `getaddrinfo` 函數來獲取 `LINK_LOCAL_ADDR` 的地址資訊，接著從該地址資訊中取出 `sockaddr_in6` 結構體指針 `sin6_info`，透過 `sin6_scope_id` 成員讀取 `SCOPE_ID`。最後使用 `freeaddrinfo` 釋放獲取的地址資訊。如果獲取 `SCOPE_ID` 成功，該值會被返回，否則返回預設值 `-1`。



```

63  int main(int argc, char*argv[]){
64      char SERVER_IP[50]={0};
65      int SERVER_PORT = 0;
66
67      int serverSock;
68      struct sockaddr_in6 serverAddr;
69      if(argc!=3){
70          error_msg("[Usage] TCP_client SERVER_IP SERVER_PORT");
71      }
72      strcpy(SERVER_IP, argv[1]);
73      SERVER_PORT = atoi(argv[2]);
74
75      serverSock = setup_socket(6);
76      setup_address(SERVER_IP,SERVER_PORT,&serverAddr);
77      if(connect(serverSock,(struct sockaddr*)&serverAddr, sizeof(serverAddr))<0){
78          error_msg("[ERROR] Failed to connetc to server. ");
79      }
80
81      interact_with_server(serverSock);
82
83      close(serverSock);
84
85      return 0;
86  }

```

63~86 行:

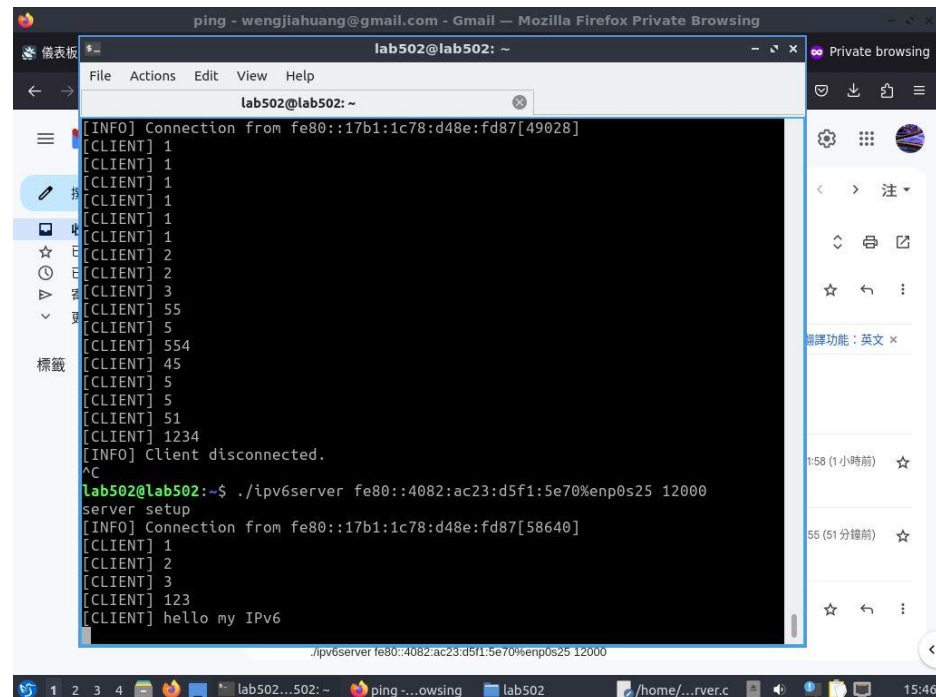
這裡是 client 的 main 函式，會從命令列參數中讀取使用者所提供的 SERVER\_IP 和 SERVER\_PORT。然後使用 setup\_socket() 函式建立一個 IPv6 的 socket，使用 setup\_address() 函式初始化 sockaddr\_in6 結構變數，包含 SERVER\_IP 和 SERVER\_PORT，並設定 scope\_id。最後，使用 connect() 函式連接到 server，如果連接失敗則呼叫 error\_msg() 函式顯示錯誤訊息。

接著，呼叫 interact\_with\_server() 函式，與 server 進行互動，將使用者輸入的訊息傳送到 server，並顯示 server 回傳的訊息。當與 server 的連線中斷時，透過 close() 函式關閉 socket。最後回傳 0 表示程式執行成功結束。

---

以下是測試結果：

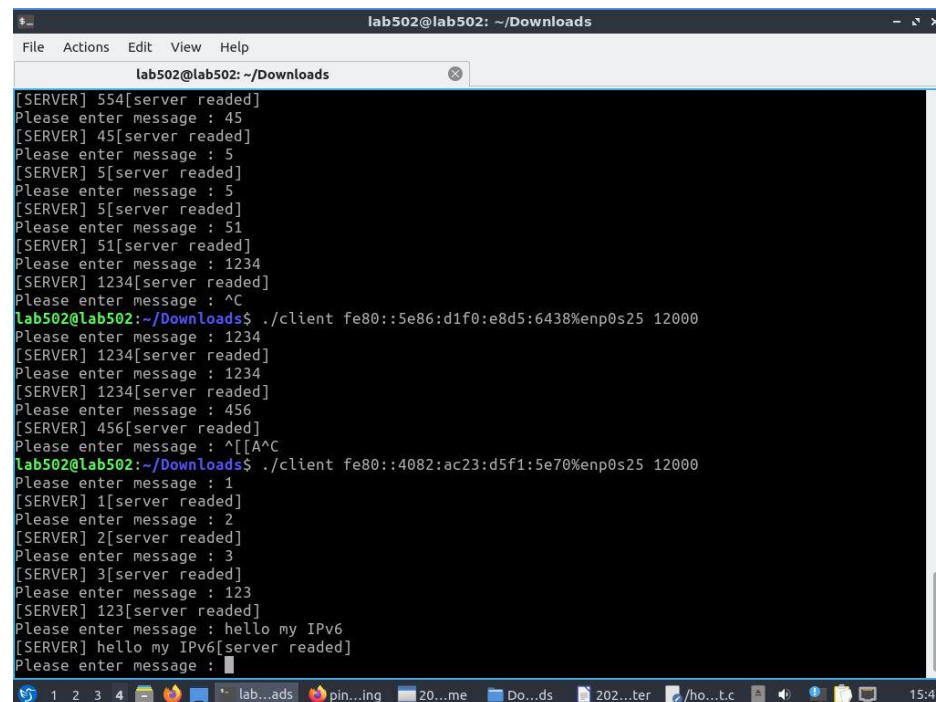
Server:



The screenshot shows a terminal window titled 'lab502@lab502: ~' with a menu bar (File, Actions, Edit, View, Help). The terminal output shows the server setup process. It starts with an INFO message: '[INFO] Connection from fe80::17b1:1c78:d48e:fd87[49028]'. This is followed by a series of CLIENT messages with IDs 1, 2, 3, 5, 55, 5, 554, 45, 5, 51, 1234. Then, an INFO message states '[INFO] Client disconnected.'. The user then runs the command './ipvs server fe80::4082:ac23:d5f1:5e70%enp0s25 12000'. The terminal shows 'server setup' and another INFO message: '[INFO] Connection from fe80::17b1:1c78:d48e:fd87[58640]'. This is followed by CLIENT messages with IDs 1, 2, 3, 123, and a message 'hello my IPv6'. The taskbar at the bottom shows icons for a file manager, terminal, browser, and other applications, with the time 15:46.

```
lab502@lab502: ~  
[INFO] Connection from fe80::17b1:1c78:d48e:fd87[49028]  
[CLIENT] 1  
[CLIENT] 1  
[CLIENT] 1  
[CLIENT] 1  
[CLIENT] 1  
[CLIENT] 1  
[CLIENT] 2  
[CLIENT] 2  
[CLIENT] 3  
[CLIENT] 55  
[CLIENT] 5  
[CLIENT] 554  
[CLIENT] 45  
[CLIENT] 5  
[CLIENT] 51  
[CLIENT] 1234  
[INFO] Client disconnected.  
^C  
lab502@lab502:~$ ./ipvs server fe80::4082:ac23:d5f1:5e70%enp0s25 12000  
server setup  
[INFO] Connection from fe80::17b1:1c78:d48e:fd87[58640]  
[CLIENT] 1  
[CLIENT] 2  
[CLIENT] 3  
[CLIENT] 123  
[CLIENT] hello my IPv6  
./ipvs server fe80::4082:ac23:d5f1:5e70%enp0s25 12000
```

Client:



The screenshot shows a terminal window titled 'lab502@lab502: ~/Downloads' with a menu bar (File, Actions, Edit, View, Help). The terminal output shows the client-side interactions. It starts with a SERVER message: '[SERVER] 554[server readed]'. This is followed by a series of 'Please enter message' prompts and corresponding SERVER messages with IDs 45, 45, 5, 5, 5, 51, 1234, 1234, 1234, 456, 456, 1, 2, 3, 123, and a message 'hello my IPv6'. The user then runs the command './client fe80::5e86:d1f0:e8d5:6438%enp0s25 12000'. The terminal shows the client sending messages and the server responding. The taskbar at the bottom shows icons for a file manager, terminal, browser, and other applications, with the time 15:46.

```
lab502@lab502: ~/Downloads  
[SERVER] 554[server readed]  
Please enter message : 45  
[SERVER] 45[server readed]  
Please enter message : 5  
[SERVER] 5[server readed]  
Please enter message : 5  
[SERVER] 5[server readed]  
Please enter message : 51  
[SERVER] 51[server readed]  
Please enter message : 1234  
[SERVER] 1234[server readed]  
Please enter message : ^C  
lab502@lab502:~/Downloads$ ./client fe80::5e86:d1f0:e8d5:6438%enp0s25 12000  
Please enter message : 1234  
[SERVER] 1234[server readed]  
Please enter message : 1234  
[SERVER] 1234[server readed]  
Please enter message : 456  
[SERVER] 456[server readed]  
Please enter message : ^[[A^C  
lab502@lab502:~/Downloads$ ./client fe80::4082:ac23:d5f1:5e70%enp0s25 12000  
Please enter message : 1  
[SERVER] 1[server readed]  
Please enter message : 2  
[SERVER] 2[server readed]  
Please enter message : 3  
[SERVER] 3[server readed]  
Please enter message : 123  
[SERVER] 123[server readed]  
Please enter message : hello my IPv6  
[SERVER] hello my IPv6[server readed]  
Please enter message : 
```

## 5. 問題與討論

1. 有許多 ipv6 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。
4. IPv6 的部署比 IPv4 慢，並且一些網路還不支持 IPv6。這意味著，在某些情況下，使用 IPv4 可能更為可靠或更具成本效益。

## 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，這次使用了 IPv6 的 Socket Programming，相比 IPv4，IPv6 具有更大的地址空間、更好的安全性和更高的效率等優點。但是，IPv6 在實際使用中仍然存在一些問題。

首先，IPv6 的普及率相對較低，因為目前仍有很多應用程序和硬件設備只支持 IPv4。這可能會使得 IPv6 應用的普及受到限制。其次，IPv6 的地址格式相對複雜，需要進行較為複雜的設置和配置，對於一些不熟悉 IPv6 的用戶來說，可能會造成一定的困擾。此外，在實際應用中，IPv6 網路環境的構建和配置也需要進行相應的調整和優化，以提高網路的可靠性和效率。

總體而言，IPv6 作為一種新型網際網路協議，具有諸多優勢，但在實際應用中仍面臨著一些挑戰和問題。通過本次實驗，我們能更深入地了解 IPv6 的基礎知識和原理，也能更好地理解 IPv6 的應用前景和發展趨勢。

## 7. 參考文獻

<https://zh.wikipedia.org/zh-tw/IPv6>

[https://test-ipv6.com/index.html.zh\\_TW](https://test-ipv6.com/index.html.zh_TW)

<https://www.ithome.com.tw/tech/92046>

<https://ipv6.twnic.tw/about-inro.html>

[https://hackmd.io/@ki\\_2CYV1QfCsYx3X8THqiw/ptyl11111](https://hackmd.io/@ki_2CYV1QfCsYx3X8THqiw/ptyl11111)

<https://nordvpn.com/zh-tw/blog/ipv4-ipv6-qubie/>