資訊通訊概論

實驗：Simple Network Programming Experiment

組員：1103309 王聖允、1103318 詹承義

Experiment 6：

(a)：

一、UDP

1. socket()：建立一個 socket，並回傳一個 File Descripter 供後續使用。有三個參數，要設定分別是：

Domain: AF_UNIX, AF_LOCAL 用於本機的溝通 AF_INET 用於 IPv4 協定而 AF_INET6 則用於 IPv6 協定。

Type: SOCK_STREAM 對應到 tcp、SOCK_DGRAM 對應到 udp 協定。

Protocol:只要設為 0，就會依照 Domain 與 Type 自動選擇。

2. bind()：綁定伺服器 Socket 和 Address 描述結構，設定要監聽的位址及 Port。

3. recvfrom()：接收 Datagram，包含傳送端 Socket 位址。

4. sendto()：傳送 Datagram，必須描述對方 Socket 位址。

5. close()：關閉先前建立的 socket。

二、TCP

1. socket():建立一個 socket，並設定其參數有三個參數要設定分別是

Domain: AF_UNIX, AF_LOCAL 用於本機的溝通 AF_INET 用於 IPv4 協定而 AF_INET6 則用於 IPv6 協定。

Type: SOCK_STREAM 對應到 tcp 協定 SOCK_DGRAM 對應到 udp 協定。

Protocol:只要設為 0，就會依照 Domain 與 Type 自動選擇。

2. bind()：綁定伺服器 Socket 和 Address 描述結構，設定要監聽的位址及 Port。

3. listen()：用於伺服器，將 socket 設定成可連線狀態，當有多個請求時放入連線佇列。

4. connect()：用於客戶端，在傳輸訊息前先與伺服器連接。

5. accept()：從連線佇列中取出第一個進行接收，回傳一個代表傳送端的 File Descripter。

6. write()：透過先前建立的連線傳輸訊息。

7. read()：透過先前建立的連線接收訊息。

5. close()：關閉先前建立的 socket。

(b)：

一、UDP

因為 UDP 不需事先建立連線即可傳輸資料，所以他的運作模式也相對簡單。

1. 客戶端傳送要求至伺服器端。

2. 伺服器則回傳回覆至客戶端。

Why is the model used:

因為 UDP 協定比較在乎傳送資料時的速度，較不注重可靠性，所以封包容易

丟失，傳送過程中也容易出錯。串流媒體、網路遊戲等服務經常使用 UDP 協定，這些網路應用不太需要可靠性機制，封包遺失不會導致服務中斷，但可以有更好的傳輸效率。DNS、DHCP 也是使用 udp 進行傳輸。

二、TCP

伺服器端先以 listen()將 socket 設定成可連線狀態並進行監聽，客戶端以 connect()送出連線請求，伺服器會將請求放入監聽佇列中。再來伺服器中以 accept()函式取得佇列中的第一個進行連線並產生一個 child socket，並在 child socket 中使用 write()和 read()進行一對一的資料傳輸。

Why the model is used?

因為 TCP 會事先確認連線，所以經由 TCP 發送的資料能完整到達目的地，但這也提高了 TCP 傳送資料時所需要的時間。通常要求速度的服務不會使用 TCP，除非極度重視資料的正確性或完整性，例如 http、ftp 或 smtp。

(c)：

一、執行環境：

1、作業系統：Ubuntu 22.04 Linux

2、Compiler：g++ 11.3.0

二、程式碼

1、UDP Server

server.cpp

```cpp
#include <iostream>
#include <string>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <unistd.h>

using namespace std;

int main()
{
    char buffer[50] = {0};
    struct sockaddr_in serverAddr = {0};

    int endpoint = socket( domain: AF_INET,  type: SOCK_DGRAM,  protocol: 0);
    if(endpoint == -1)
    {
        perror( s: "failed to create socket");
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket created on file descriptor " << endpoint << endl;
    }
```

```
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons( hostshort: 6666);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    int rc = bind( fd: endpoint,  addr: (const struct sockaddr *)&serverAddr,  len: sizeof(serverAddr));

    if(rc == -1)
    {
        perror( s: "failed to bind");
        close( fd: endpoint);
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket binded with address " << endl;
    }

    socklen_t len = 0;
    int mLength = recvfrom( fd: endpoint,  buf: (char *)buffer,  n: 50,  flags: MSG_WAITALL,  addr: 0,  addr_len: &len);

    buffer[mLength] = '\n';
    cout << endl << "\e[0;32m[Success]\e[1;37m Data received : " << buffer;

    cout << endl << "\e[1;33m[close]\e[1;37m closing socket"<< endl;
    close( fd: endpoint);
```

補充說明：

1、cout 結合 Linux Bash 輸出顏色設定（例如：\e[0;32m 為綠色）

2、UDP Client

```
#include <iostream>
#include <string>
#include <sys/types.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
using namespace std;

int main()
{
    string message = "Hello World";

    //create the server address description object
    struct sockaddr_in serverAddr = {0};

    //create a socket (endpoint) for the client
    int endpoint = socket(AF_INET, SOCK_DGRAM, 0);

    if(endpoint == -1)
    {
        perror("Failed to create socket");
        exit(EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket created on file descriptor " << endpoint << endl;
    }
```

```
    string ipv4 = "140.138.242.143";
    //set the receiving server address
    serverAddr.sin_family = AF_INET; //IPV4
    serverAddr.sin_port = htons( hostshort: 6666);
    serverAddr.sin_addr.s_addr = inet_addr( cp: ipv4.c_str());

    cout << endl << "\e[0;32m[Success]\e[1;37m Set server address to " << ipv4 << endl;

    int len = sendto( fd: endpoint,  buf: (const char *)message.c_str(),  n: message.size(),
                      flags: 0,  addr: (const struct sockaddr *)&serverAddr,  addr_len: sizeof(serverAddr) );

    if(len == -1)
    {
        perror( s: "Fail to send message");
    }
    else
    {
        cout << endl << "\e[0;32m[Finished]\e[1;37m Message sent" << endl;
    }

    cout << endl << "\e[1;33m[close]\e[1;37m closing socket"<< endl << endl;
    close( fd: endpoint);
}
```

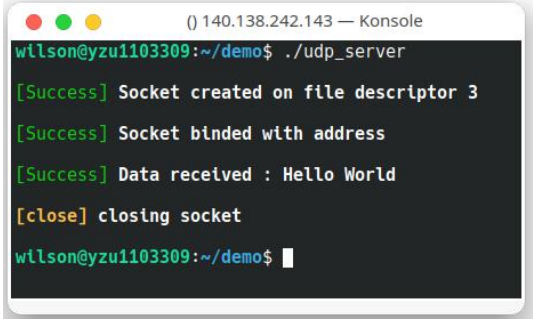
封包擷取：

Source	Destination	Protocol	Length	Info
140.138.50.131	140.138.242.143	UDP	53	49196 → 6666 Len=11

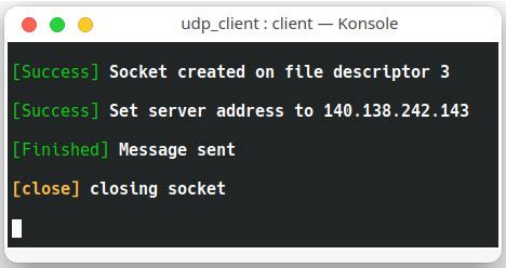
```
0000  f4 4e 05 08 42 c0 90 e8  68 f6 63 8b 08 00 45 00   ·N··B··· h·c···E·
0010  00 27 35 5b 40 00 40 11  c7 43 8c 8a 32 83 8c 8a   ·'5[@·@· ·C··2···
0020  f2 8f c0 2c 1a 0a 00 13  95 9b 48 65 6c 6c 6f 20   ···,···· ··Hello
0030  57 6f 72 6c 64                                    World
```

因為在 Client 程式中設定 s_addr 為 140.138.242.143、sin_port 為 6666，所以封包 Destination 為 140.138.242.143，Server 端以 Port 6666 接收。而 Socket type 設為 SOCK_DGRAM，所以封包 protocol 為 UDP。

執行視窗截圖：



Server 端



Client 端

3、TCP Server

```cpp
#include <iostream>
#include <string>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <unistd.h>

using namespace std;

int main()
{
    char buffer[50];
    memset( s: buffer,  c: '\0',  n: sizeof(buffer));
    struct sockaddr_in serverAddr = {0}, cli = {0};

    int endpoint = socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
    if(endpoint == -1)
    {
        perror( s: "failed o create socket");
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket created on file descriptor " << endpoint << endl;
    }
```

```cpp
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons( hostshort: 6666);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    int status = bind( fd: endpoint,  addr: (const struct sockaddr *)&serverAddr,  len: sizeof(serverAddr));

    if(status == -1)
    {
        perror( s: "failed to bind");
        close( fd: endpoint);
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket binded with address " << endl;
    }

    status = listen( fd: endpoint,  n: 5);
    if(status != 0)
    {
        perror( s: "listen failed");
        close( fd: endpoint);
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m listening on socket " << endl;
    }
```

```
    socklen_t length_cli = sizeof(cli);
    int client_endpoint = accept( fd: endpoint,  addr: (struct sockaddr *)&cli,  addr_len: &length_cli);

    if(status == -1)
    {
        perror( s: "failed to accept");
        close( fd: endpoint);
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m connection with client was set up on file descriptor " << client_endpoint << endl;
    }

    read( fd: client_endpoint,  buf: &buffer,  nbytes: sizeof(buffer) );
    cout << endl << "\e[0;32m[Success]\e[1;37m Data received : " << buffer << endl;

    write( fd: client_endpoint,  buf: buffer,  n: sizeof(buffer) );
    cout << endl << "\e[0;32m[Finished]\e[1;37m sent back check message "<< endl;

    cout << endl << "\e[1;33m[close]\e[1;37m closing socket"<< endl << endl;
    close( fd: endpoint);
}
```

4、TCP Client

```

int main()
{
    //string message = "Hello World";

    char message[50] = "Hello world";
    //create the server address description object
    struct sockaddr_in serverAddr = {0};

    //create a socket (endpoint) for the client
    int endpoint = socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);

    if(endpoint == -1)
    {
        perror( s: "Failed to create socket");
        exit( status: EXIT_FAILURE);
    }

```

```
    string ipv4 = "140.138.242.143";
    //set the receiving server address
    serverAddr.sin_family = AF_INET; //IPV4
    serverAddr.sin_port = htons( hostshort: 6666);
    serverAddr.sin_addr.s_addr = inet_addr( cp: ipv4.c_str());

    cout << endl << "\e[0;32m[Success]\e[1;37m Set server address to " << ipv4 << endl;
    //set up connection before sending message
    int status = connect( fd: endpoint,  addr: (const struct sockaddr *)&serverAddr,  len: sizeof(serverAddr));

    if(status != 0)
    {
        perror( s: "failed to connect server before ");
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket created on file descriptor " << endpoint << endl;
    }
```

```
48      write( fd: endpoint,  buf: message,  n: sizeof(message) );
49      cout << endl << "\e[0;32m[Finished]\e[1;37m Message sent, waiting for check reply" << endl;
50      char check[50];
51      memset( s: check,  c: '\0' , n: sizeof(check));
52      read( fd: endpoint,  buf: check,  nbytes: sizeof(check) );
53
54      //The server will send back the exact message
55      status = strcmp(check, message);
56
57      if( status == 0 )
58          cout << endl << "\e[0;32m[Success]\e[1;37m Message was sent properly and perfectly" << endl;
59      else
60          cout << endl << "\e[0;31m[Error]\e[1;37m Reply doesn't match with the original message" << endl;
61
62
63      cout << endl << "\e[1;33m[close]\e[1;37m Closing socket" << endl << endl;
64      close( fd: endpoint);
65  }
```

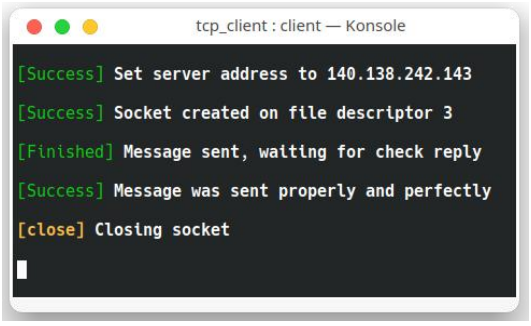
封包擷取：

Source	Destination	Protocol	Length	Info
140.138.50.131	140.138.242.143	TCP	74	55802 → 6666 [SYN] Seq=0 Win=64240
140.138.242.143	140.138.50.131	TCP	74	6666 → 55802 [SYN, ACK] Seq=0 Ack=1
140.138.50.131	140.138.242.143	TCP	66	55802 → 6666 [ACK] Seq=1 Ack=1 Win=
140.138.50.131	140.138.242.143	TCP	116	55802 → 6666 [PSH, ACK] Seq=1 Ack=1
140.138.242.143	140.138.50.131	TCP	66	6666 → 55802 [ACK] Seq=1 Ack=51 Win=
140.138.242.143	140.138.50.131	TCP	116	6666 → 55802 [PSH, ACK] Seq=1 Ack=51
140.138.50.131	140.138.242.143	TCP	66	55802 → 6666 [ACK] Seq=51 Ack=51 Win
140.138.50.131	140.138.242.143	TCP	66	55802 → 6666 [FIN, ACK] Seq=51 Ack=
140.138.242.143	140.138.50.131	TCP	66	6666 → 55802 [FIN, ACK] Seq=51 Ack=
140.138.50.131	140.138.242.143	TCP	66	55802 → 6666 [ACK] Seq=52 Ack=52 Win
140.138.242.143	140.138.50.131	TCP	66	6666 → 55802 [ACK] Seq=52 Ack=52 Win

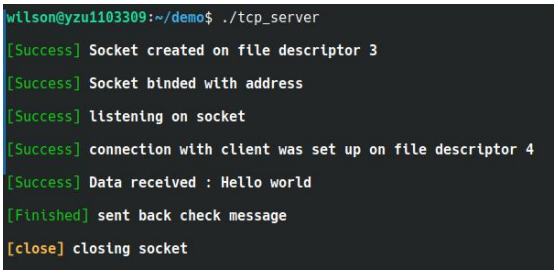
Socket type 設為 SOCK_STREAM，所以封包 protocol 為 TCP。可以看到經過程序比 UDP 多了許多，在 Info 欄位也可以看到每個封包都有不同的 Flag，其中 SYN 代表連接請求、ACK 代表確認接收之回傳、PSH 為訊息內容、FIN 代表關閉連接。例如以下看到 Flag 為「PSH, ACK」的封包內容，可見訊息內容「Hello World」:

```
0000  90 e8 68 f6 63 8b f4 4e  05 08 42 c0 08 00 45 00   ··h·c··N ··B···E·
0010  00 66 99 61 40 00 3d 06  66 09 8c 8a f2 8f 8c 8a   ·f·a@·=· f·······
0020  32 83 1a 0a d9 fa 59 a8  88 a4 19 75 50 b3 80 18   2·····Y· ···uP···
0030  01 fd 95 11 00 00 01 01  08 0a 34 31 d4 0e f2 21   ········ ··41···!
0040  f4 a2 48 65 6c 6c 6f 20  77 6f 72 6c 64 00 00 00   ··Hello  world···
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00                                       ····
```

執行視窗截圖：



TCP Client 端



TCP Server 端

Experiment 6：

1、Stop and Wait Server

```
#define ACK 0
#define NORMAL 1
#define CLOSE 2

typedef struct packet{
    char data[256];
}Packet;

typedef struct frame{
    int type; //ACK:0, NORMAL:1
    int id;
    Packet packet;
}Frame;

//keep count of the frames for checking ack ID
int FrameID = 0;
//create the server address description object
struct sockaddr_in serverAddr = {0};
```

```
int main()
{
    //create input and output frames
    //each frame includes the packet (cstring message)
    Frame output;
    Frame input;

    //create a socket (endpoint) for the client
    //UDP uses "SOCK_DGRAM"
    int endpoint = socket( domain: AF_INET,  type: SOCK_DGRAM,  protocol: 0);
    if(endpoint == -1)
    {
        perror( s: "Failed to create socket");
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket created on file descriptor " << endpoint << endl;
    }

    //set the receiving server address
    serverAddr.sin_family = AF_INET; //IPV4
    serverAddr.sin_port = htons( hostshort: 6666);
    serverAddr.sin_addr.s_addr = INADDR_ANY;
```

```

    int status = bind( fd: endpoint,  addr: (struct sockaddr *)&serverAddr,  len: sizeof(serverAddr) );
    if(status == -1)
    {
        perror( s: "Failed to bind socket with address");
        close( fd: endpoint);
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket binded with address " << endl;
    }

    //start the sending process
    start_process( &: output,  &: input, endpoint);

    close( fd: endpoint);
}
```

```
void start_process(Frame &output, Frame &input, int endpoint)
{
    while(true)
    {
        sockaddr_in replyAddr;
        socklen_t replyAddrSize;
        recvfrom( fd: endpoint,  buf: &input,  n: sizeof(Frame),  flags: 0,  addr: (struct sockaddr *)&replyAddr,  addr_len: &replyAddrSize);

        if(input.type == CLOSE)
        {
            cout << endl << "\e[1;33m[close]\e[1;37m close signal received, exiting" << endl << endl;
            break;
        }

        if(input.type == NORMAL && input.id == FrameID)
        {
            cout << endl << "\e[0;32m[Success]\e[1;37m Data received : " << input.packet.data << endl;

            output.type = ACK;
            output.id = ++FrameID;

            sendto( fd: endpoint,  buf: &output,  n: sizeof(output),  flags: 0,  addr: (struct sockaddr *)&replyAddr,  addr_len: replyAddrSize);
            cout << endl << "\e[0;32m[Finished]\e[1;37m ACK sent" << endl;
        }

    }
}
```

2、Stop and Wait Client

```
#define ACK 0
#define NORMAL 1
#define CLOSE 2

typedef struct packet{
    char data[256];
}Packet;

typedef struct frame{
    int type; //ACK:0, NORMAL:1
    int id;
    Packet packet;
}Frame;

//keep count of the frames for checking ack ID
int FrameID = 0;
//set the state to ready, in order to start sending new frames
bool ready = true;
//create the server address description object
struct sockaddr_in serverAddr = {0};
```

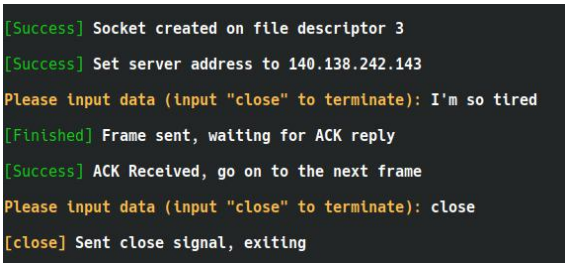
```
    string ipv4 = "140.138.242.143";
    //set the receiving server address
    serverAddr.sin_family = AF_INET; //IPV4
    serverAddr.sin_port = htons( hostshort: 6666);
    serverAddr.sin_addr.s_addr = inet_addr( cp: ipv4.c_str());

    cout << endl << "\e[0;32m[Success]\e[1;37m Set server address to " << ipv4 << endl;

    //start the sending process
    start_process( &: output,  &: input, endpoint);

    close( fd: endpoint);
}
```

```cpp
int main()
{
    //create input and output frames
    //each frame includes the packet (cstring message)
    Frame output;
    Frame input;

    //create a socket (endpoint) for the client
    //UDP uses "SOCK_DGRAM"
    int endpoint = socket( domain: AF_INET,  type: SOCK_DGRAM,  protocol: 0);
    if(endpoint == -1)
    {
        perror( s: "Failed to create socket");
        exit( status: EXIT_FAILURE);
    }
    else
    {
        cout << endl << "\e[0;32m[Success]\e[1;37m Socket created on file descriptor " << endpoint << endl;
    }
```

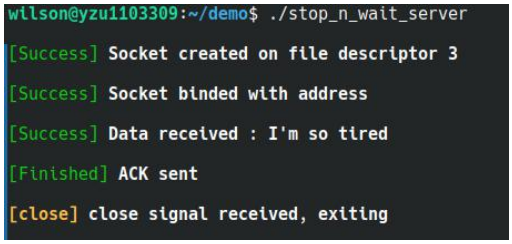
```cpp
void start_process(Frame &output, Frame &input, int endpoint)
{
    while(true)
    {
        if(ready == true)
        {
            output.id = FrameID;
            output.type = NORMAL;
            memset( s: output.packet.data,  c: '\0',  n: sizeof(output.packet.data) );

            //wait for user input
            cout << endl << "\e[1;33mPlease input data (input \"close\" to terminate): \e[1;37m";
            cin.getline( s: output.packet.data,  n: 256);

            if(strcmp(output.packet.data, "close") == 0)
            {
                output.type = CLOSE;
                sendto( fd: endpoint,  buf: &output,  n: sizeof(output),
                        flags: 0,  addr: (const struct sockaddr*)&serverAddr,  addr_len: sizeof(serverAddr));
                cout << endl << "\e[1;33m[close]\e[1;37m Sent close signal, exiting " << endl << endl;
                break;
            }

            //send to the server
            sendto( fd: endpoint,  buf: &output,  n: sizeof(output),
                    flags: 0,  addr: (const struct sockaddr*)&serverAddr,  addr_len: sizeof(serverAddr));
            cout << endl << "\e[0;32m[Finished]\e[1;37m Frame sent, waiting for ACK reply" <<endl;
        }
```

```cpp
        else if(ready == false)
        {
            //no ack received, resend the frame
            sendto( fd: endpoint,  buf: &output,  n: sizeof(output),
                    flags: 0,  addr: (const struct sockaddr*)&serverAddr,  addr_len: sizeof(serverAddr));
        }

        //start waiting for ack
        socklen_t address_size = sizeof(serverAddr);
        recvfrom( fd: endpoint,  buf: &input,  n: sizeof(input),
                  flags: 0,  addr: (struct sockaddr *)&serverAddr,  addr_len: &address_size);

        if(input.type == ACK && input.id == FrameID + 1)
        {
            cout << endl << "\e[0;32m[Success]\e[1;37m ACK Received, go on to the next frame" << endl;
            ready = true;
        }
        else
        {
            cout << endl << "\e[0;31m[Error]\e[1;37m ACK Lost, resending" << endl;
            ready = false;
        }

        ++FrameID;
    }
}
```

測試執行截圖：



Stop and Wait Client



Stop and Wait Server

由此可知，Client 先送了一則含有訊息「I'm so tired」的封包，Server 端回傳 ACK 確認封包，接著 Client 傳送 「close」 訊息，Client 和 Server 即關閉先前開啟的 Socket。以下是擷取的封包：

Source	Destination	Protocol	Length	Info
140.138.50.131	140.138.242.143	UDP	306	49504 → 6666 Len=264
140.138.242.143	140.138.50.131	UDP	306	6666 → 49504 Len=264
140.138.50.131	140.138.242.143	UDP	306	49504 → 6666 Len=264

可以看到有兩個封包送往 Server 端（140.138.242.143:6666），有一個封包（ACK）送回 Client，與執行情況相符。上圖第一個封包也包含「I'm so tired」的訊息：

```
f4 4e 05 08 42 c0 90 e8  68 f6 63 8b 08 00 45 00   ·N··B··· h·c···E·
01 24 70 e4 40 00 40 11  8a bd 8c 8a 32 83 8c 8a   ·$p·@·@· ····2···
f2 8f c1 60 1a 0a 01 10  ca 39 01 00 00 00 00 00   ···`···· ·9······
00 00 49 27 6d 20 73 6f  20 74 69 72 65 64 00 00   ··I'm so  tired··
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```