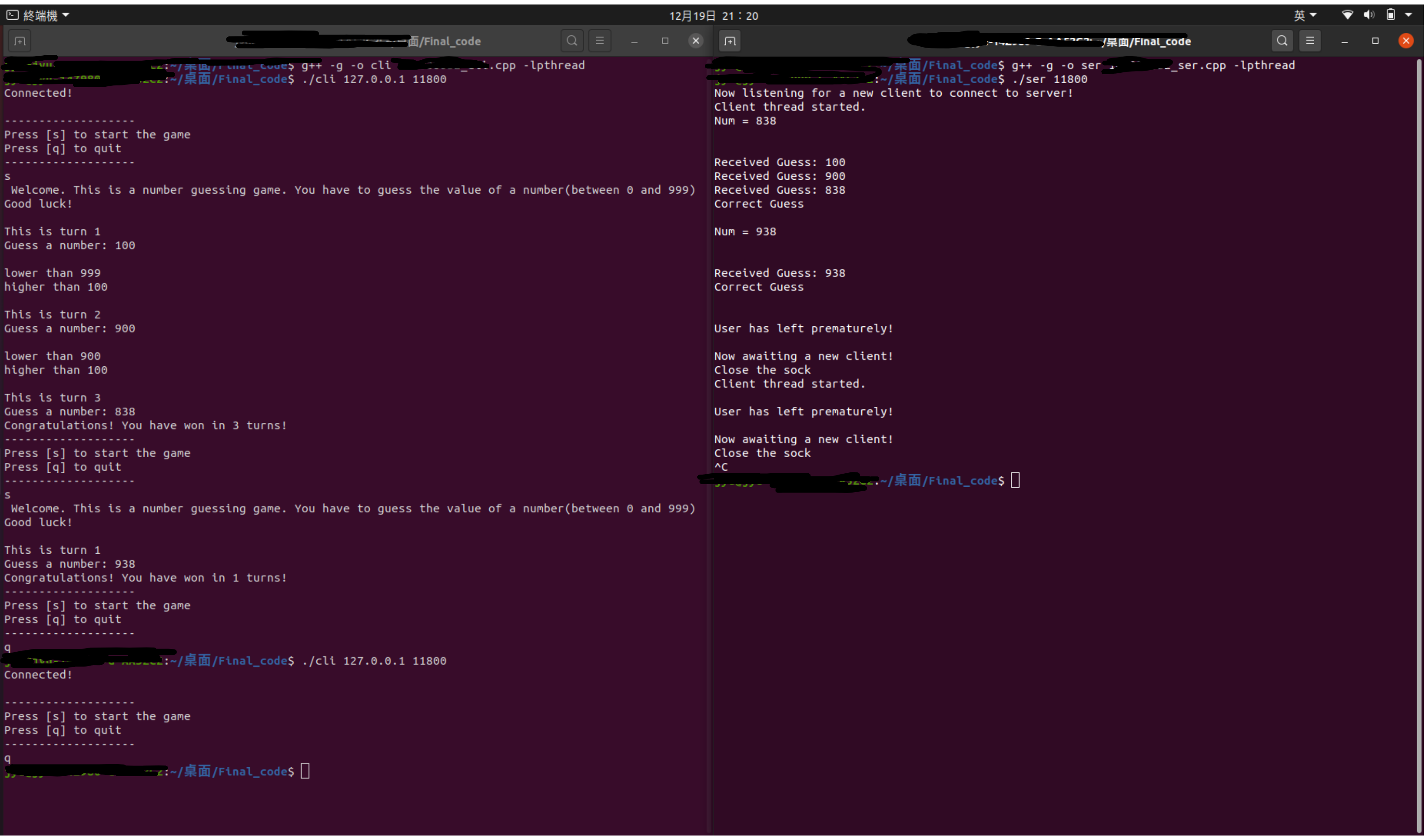
Introduction to Computer Networks

Final Project

Report

壹、執行結果



終端機 12月19日 21：20 英

[redacted]/Final_code

```
[redacted]:~/桌面/Final_code$ g++ -g -o cli [redacted].cpp -lpthread
[redacted]:~/桌面/Final_code$ ./cli 127.0.0.1 11800
Connected!

------------------
Press [s] to start the game
Press [q] to quit
------------------
s
 Welcome. This is a number guessing game. You have to guess the value of a number(between 0 and 999)
Good luck!

This is turn 1
Guess a number: 100

lower than 999
higher than 100

This is turn 2
Guess a number: 900

lower than 900
higher than 100

This is turn 3
Guess a number: 838
Congratulations! You have won in 3 turns!
------------------
Press [s] to start the game
Press [q] to quit
------------------
s
 Welcome. This is a number guessing game. You have to guess the value of a number(between 0 and 999)
Good luck!

This is turn 1
Guess a number: 938
Congratulations! You have won in 1 turns!
------------------
Press [s] to start the game
Press [q] to quit
------------------
q
[redacted]:~/桌面/Final_code$ ./cli 127.0.0.1 11800
Connected!

------------------
Press [s] to start the game
Press [q] to quit
------------------
q
[redacted]:~/桌面/Final_code$
```

[redacted]/桌面/Final_code

```
[redacted]~/桌面/Final_code$ g++ -g -o ser [redacted]_ser.cpp -lpthread
[redacted]:~/桌面/Final_code$ ./ser 11800
Now listening for a new client to connect to server!
Client thread started.
Num = 838


Received Guess: 100
Received Guess: 900
Received Guess: 838
Correct Guess

Num = 938


Received Guess: 938
Correct Guess


User has left prematurely!

Now awaiting a new client!
Close the sock
Client thread started.

User has left prematurely!

Now awaiting a new client!
Close the sock
^C
[redacted].~/桌面/Final_code$
```

貳、實作

一、Server

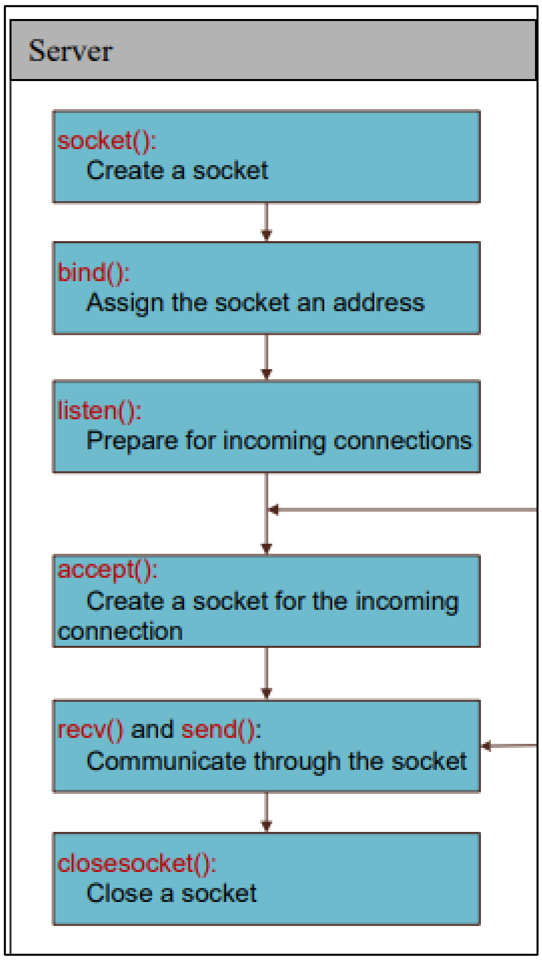
(一) 全域變數

為了講解方便，這裡先用註解說明底下會用到的變數和一些 struct。

```
const int MAX_ARGS = 2; // 用來存最大可輸入變數
const int PORT_ARG = 1; // 用來存 port 的 id
const int MAX_PENDING = 5; //用來存最大 pending incoming requests
const int MAXPORT = 11899; // 最大 port
const int MINPORT = 11800; // 最小 port

//用來存連線的訊息，像是 sock、還有進行回合數
struct arg_t
{
    int sock;
    int roundCount;
};

//server 判定結果之後，會回傳此結構作為判定結果給 client
struct roundResult
{
    int tooHigh;
    int tooLow;
    int equal;
};
```

(二)　main

在進行 Server 的實作時，先參考助教所提供的圖來知道實作流程：


Server
socket(): Create a socket
bind(): Assign the socket an address
listen(): Prepare for incoming connections
accept(): Create a socket for the incoming connection
recv() and send(): Communicate through the socket
closesocket(): Close a socket

剛開始會需要判斷 port 和 IP 是否輸入正確，使用以下程式碼判斷：

```
    if (argc != MAX_ARGS)
    {
        cerr << "Invalid number of arguments. Please input IP address 
for first arg, then port # for "
```

```
            << " second one. Now exiting program.";
        exit(-1);
    }

    unsigned short portNum = (unsigned short)strtoul(argv[PORT_ARG],
NULL, 0);
    if (portNum > MAXPORT || portNum < MINPORT)
    {
        cerr << "This port is not assigned to this program. Please try
again with " << endl
            << "numbers that are between 11800 & 11899. Now exiting. ";

        exit(-1);
    }
```

判斷 port 和 IP 的方法蠻直覺的，在這裡就不多加贅述。接著就要開始建立新的 socket：

```
    int status; // 用來檢查 TCP 函式
    int clientSock; // 用來存 client

    int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
    {
        cerr << "Error with socket. Now exiting program. " << endl;
        close(sock);
        exit(-1);
    }
```

宣告一個整數 sock 來存 socket()回傳的數值。若有 error 的話會回傳-1，並印出錯誤訊息，關閉 sock。在 socket()裡面，AF_INET 代表 IPv4 網路協定，SOCK_STREAM 代表 TCP，IPPROTO_TCP 是指定實際使用的傳輸協定(TCP)。接著，會進到 bind()，也就是要把設定的 address 綁在 socket 身上，實際實作如下：

```
    //設定 port
    struct sockaddr_in servAddr;
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(portNum);

    status = bind(sock, (struct sockaddr *)&servAddr,
                 sizeof(servAddr));
    if (status < 0)
    {
        cerr << "Error with bind. Now exiting program. " << endl;
        close(sock);
        exit(-1);
    }
```

因為 bind 會用到一些 struct，用來指出這個 socket 所要監聽的 address 和 port number 等其他資訊，內容如下：

```
include <netinet/in.h>

struct sockaddr {
    unsigned short    sa_family;       // 2 bytes address family,
AF_xxx
    char              sa_data[14];     // 14 bytes of protocol address
};

// IPv4 AF_INET sockets:

struct sockaddr_in {
    short            sin_family;       // 2 bytes e.g. AF_INET,
AF_INET6
    unsigned short   sin_port;         // 2 bytes e.g. htons(3490)
    struct in_addr   sin_addr;         // 4 bytes see struct in_addr,
below
    char             sin_zero[8];      // 8 bytes zero this if you want
to
};

struct in_addr {
    unsigned long s_addr;              // 4 bytes load with
inet_pton()
};
```

struct sockaddr 和 struct sockaddr_in 相似，sockaddr_in 將 sockaddr 中的 `char sa_data[14];` ，長度 14 bytes 轉為三個變數，一般寫成是我們使用 sockaddr_in 對其中的變數賦值，再將其轉型為 sockaddr。s_addr 是用 unsigned long int 來表示 host address number。

回到 bind()的實作，這邊使用的是 INADDR_ANY，是指任何連上來的

address，如果要接受來自 internet 的 connection 可使用。然後用 status 存 bind() 回傳的數值，0 為成功，-1 為失敗。再接著，我們來到 listen()：

```
    status = listen(sock, MAX_PENDING);
    cerr << "Now listening for a new client to connect to server!" <<
endl;
    if (status < 0)
    {
        cerr << "Error with listen. Now exiting program. " << endl;
        close(sock);
        exit(-1);
    }
```

listen()是要等待請求，sock 是上面的 socket file descriptor，MAX_PENDING 是監聽佇列大小，當有連線請求到來會進入此監聽佇列，連線請求被 accept() 後會離開監聽佇列，當佇列滿時，新的連線請求會返回錯誤。一樣，如果有錯誤會傳回-1。接著，監聽到 client 的 connect()請求後，我們需要 accept()，實作如下：

```
    while (true)
    {
        pthread_t tid;

        struct sockaddr_in clientAddr;
        socklen_t addrLen = sizeof(clientAddr);
        clientSock = accept(sock, (struct sockaddr *)&clientAddr,
&addrLen);
        if (clientSock < 0)
        {
            cerr << "Error with accept. Now exiting program. " << endl;
            close(clientSock);
            exit(-1);
        }

        arg_t *args_p = new arg_t;
        args_p->sock = clientSock;

        status = pthread_create(&tid, NULL, func, (void *)args_p);
        if (status)
        {
            cerr << "Error creating threads, return code is " << status 
<< ". Now exiting " << endl;
            close(clientSock);
            exit(-1);
        }
        cerr << "Client thread started." << endl;
    }
```

這裡會用 clientSock 去接收 accept()回傳值。看是否有成功 accept 到。若有的話，會傳回另一個包含 client 資訊的新 socket descriptor，作為傳送資料用。若一切都順利的話就會開始進行猜數字了。

(三) 猜數字(*func())

一樣先從一些變數開始，如下：

```
    //reclaiming variables from args_pa
    arg_t *args_p;
    args_p = (arg_t *)args_pa;

    srand(time(NULL));
    args_p->roundCount = 0;
    long roundCount = 0;
    long actualNums;
    long numsGuess;
    long numHigh, numOn, numLow;
    bool won = false;
    roundResult result;
    roundResult *rPointer;
    bool exit = false;
    long wantToPlay;

    pthread_detach(pthread_self());
```

這些變數的意思到底下再解釋。接下來，會進到一個 while(true)迴圈裡面，首先會確認玩家的遊玩意願：

```
        wantToPlay = receiveLong(*args_p, exit);
        if (wantToPlay == 1)
        {
            exit = true;
            break;
        }
```

我用 wantToPlay 去 receive 一個 long number，receiveLong()的實作會在下面提到。如果玩家已經不想玩的話，就會跳出這個迴圈。等待下一個使用者。如果玩家還想玩，就會產生一亂數：

```
            exit = false;
            won = false;
            roundCount = 0;

            actualNums = (rand() % 1000);
            cerr << "Num = " << actualNums << endl;
```

實作方法應該淺顯易懂，exit 用來看玩家是否已經離開，won 用來看玩家是否已經猜到數字了，roundCount 則是目前玩家已經猜幾次了。接著，當玩家沒有贏也沒有離開，會不斷執行以下程式：

```
                cerr << endl
                     << endl;
                numHigh = 999;
                numOn = 0;
                numLow = 0;
                do
                {
                    numsGuess = receiveLong(*args_p, exit);
                    if (!exit)
                    {
                        cerr << "Received Guess: " << numsGuess << endl;
                        if (numsGuess < actualNums)
                            numLow = numsGuess;
                        else if (numsGuess > actualNums)
                            numHigh = numsGuess;
                        else
                            numOn = 1;
                    }

                    result.tooHigh = numHigh;
                    result.tooLow = numLow;
                    result.equal = numOn;

                    sendResult(result, *args_p);

                    roundCount++;
```

```
            if (numOn == 1)
            {
                won = true;
                cerr << "Correct Guess" << endl<<endl;
            }

        } while (!won);

        if (!exit)
        {
            sendLong(roundCount, *args_p);
        }
```

numHigh 是目前有可能的最大值，numLow 是目前有可能的最小值，numOn 則是用來看玩家是否有猜對。然後我用 numsGuess 去接受 receiveLong()回傳的數值，當玩家沒有離開的時候會印出玩家猜的號碼，並更新 numOn、numHigh、numLow。接下來存入 result 裡面，用 sendResult()回傳給 client，sendResult()的實作在下方會補充。回傳後 roundCount 加一。如果玩家猜到了(numOn = 1)，就設定 won 是 true，跳出迴圈，否則繼續執行。如果玩家沒有離開，就回傳玩家用了幾個回合。

實際上 client 端也會記錄回合數，但為了避免玩家私自竄改，因此還是使用 server 回傳的方式。接下來，如果玩家離開了，就執行：

```
    if (exit)
    {
        cerr << endl
             << "User has left prematurely! " << endl;
        cerr << endl
             << "Now awaiting a new client!" << endl;
    }
    cerr << "Close the sock" << endl;
    close(args_p->sock);

    pthread_exit(NULL);
```

(四) sendLong

```
void sendLong(long num, arg_t connInfo)
{
    long temp = htonl(num);
    int bytesSent = send(connInfo.sock, (void *)&temp, sizeof(long), 0);
    if (bytesSent != sizeof(long))
    {
        cerr << "Error sending long! Now exiting. ";
        close(connInfo.sock);
        exit(-1);
    }
}
```

首先用 htonl()將 unsigned integer host long 從 host byte order 轉為 network byte order。接著用 send()去傳資料，並用 bytesSent 儲存回傳資訊來 debug。

(五) reveiveLong

```
long receiveLong(arg_t connInfo, bool &abort)
{
    int bytesLeft = sizeof(long);
    long networkInt;
    char *bp = (char *)&networkInt;

    while (bytesLeft > 0)
    {
        int bytesRecv = recv(connInfo.sock, (void *)bp, bytesLeft, 0);
        if (bytesRecv <= 0)
        {
            abort = true;
            break;
        }
        else
        {
            bytesLeft = bytesLeft - bytesRecv;
            bp = bp + bytesRecv;
        }
    }
    if (!abort)
    {
        networkInt = ntohl(networkInt);
        return networkInt;
    }
    else
        return 0;
}
```

首先會宣告一個 bytesLeft，表示目前還可以收多少。然後宣告 long netWorkInt，把 bp 指到 netWorkInt。當目前還可以收(bytesLeft>0)，就 invoke recv()來收，並判斷是否有 error。如果有 error 就 abort，沒有的話就更新 bytesLeft 和 bp。

最後再用 ntohl()將 unsigned integer netlong 從 network byte order 轉為 host byte order。

(六) sendResult()

```
void sendResult(roundResult result, arg_t connInfo)
{
    result = toNet(result);
    roundResult *rPointer;
    rPointer = &result;
    int bytesSent = send(connInfo.sock, (void *)rPointer,
sizeof(result), 0);
    if (bytesSent != sizeof(result))
    {
        cerr << "Error sending results! Now exiting program.";
        close(connInfo.sock);
        exit(-1);
    }
}
```

在這裡我會用 toNet()這個函式把 result 這個 struct 裡面的東西都轉為 network byte order。然後把 rPointer 指到 result。接著會呼叫 send()來送資料。

至此 Server 已經實作完畢，下面開始說明 Client 的實作。

二、Client

(一) 全域變數

在 client 部分一樣先來看一些下面會用到的全域變數：

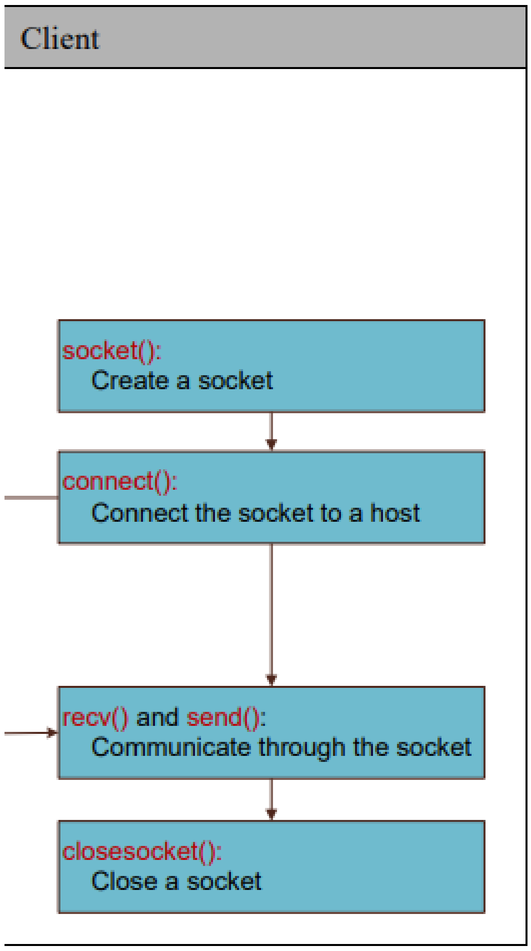
```
const int MAX_ARGS = 3;
const int PORT_ARG = 2;
const int IP_ARG = 1;
const int MAXPORT = 11899;
const int MINPORT = 11800;

struct roundResult
```

```
{
    int tooHigh;
    int tooLow;
    int equal;
};
```

其實和 server 端的基本上雷同，struct roundResult 用來存每一次猜數字的結果。

(二) main：連線

在實作之前先看一下講義的 TCP flow chart 了解一下 client 的實作流程：



Client
socket(): Create a socket
connect(): Connect the socket to a host
recv() and send(): Communicate through the socket
closesocket(): Close a socket

剛開始收 IP 和 port number，我用以下程式碼實作：

```
    int status;
    bool won = false;
    long roundCount = 0;
    long numGuess;
    int numActual;
    bool goodInput;
    long tooHigh, tooLow, equal;
    if (argc != MAX_ARGS)
    {
        cerr << "Invalid number of arguments. Please input port # for
first arg. Now exiting program.";
        exit(-1);
    }

    unsigned short portNum = (unsigned short)strtoul(argv[PORT_ARG],
NULL, 0);
    if (portNum > MAXPORT || portNum < MINPORT)
    {
        cerr << "This port is not assigned to this program. Please try
again with " << endl
             << "numbers that are between 11800 & 11899. Now exiting. ";
        exit(-1);
    }

    unsigned long servIP;
    status = inet_pton(AF_INET, argv[IP_ARG], &servIP);
    if (status <= 0)
        exit(-1);
    struct sockaddr_in servAddr;
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = servIP;
    servAddr.sin_port = htons(portNum);
```

這邊的實作其實和 server 端的幾乎雷同，只是多了 IP 需要處理，就不多作說明。再來，根據流程圖，我需要 create a socket，因此我們用以下程式碼實作：

```
    int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
    {
        cerr << "Error with socket" << endl;
        exit(-1);
    }
```

這裡和 server 端也一樣，接著要實作 connect()，向指定的 server 進行直接通訊：

```
    status = connect(sock, (struct sockaddr *)&servAddr,
sizeof(servAddr));
    if (status < 0)
    {
        cerr << "Error with connect" << endl;
        exit(-1);
    }

    cerr << "Connected! " << endl
         << endl;
```

用 status 來存 connect()的回傳值以偵錯。connect()內部輸入的值其實和 bind()差不多。接下來，client 應該就可以順利連上 server，開始進行猜數字。

(三) main：猜數字

猜數字的會先用一個 while(true)迴圈包起來，然後剛開始會詢問使用者遊玩的意願：

```
        cerr << "--------------------" << endl;
        cerr << "Press [s] to start the game" << endl;
        cerr << "Press [q] to quit" << endl;
        cerr << "--------------------" << endl;
        char signal;
        cin >> signal;
        if (signal == 'q')
        {
            sendLong(1, sock);
            break;
        }
        else
            sendLong(0, sock);
        roundCount = 0;
        won = false;

        cerr << " Welcome. This is a number guessing game. You have to
guess the value of a number(between 0 and 999)" << endl;
        cerr << "Good luck!" << endl
             << endl;
```

當使用者輸入 q 的時候，就會傳一個值給 server，跟他說使用者沒有要玩，需等待下一個 client，並跳出程式。反之，就會傳一個 long 給 server 說現在可以開始遊戲了，Server 就會產生一亂數。sendLond()的實作會在下方說明。然

後設定 roundCount 和 won 分別為 0 和 false，並印出歡迎訊息。接著，當使用者沒有猜中的情況下(while(!won))，會不斷執行以下程式：

```
            cerr << "This is turn " << (roundCount + 1) << endl;

            cerr << "Guess a number"
                 << ": ";
            goodInput = false;

            while (!goodInput)
            {
                if (cin >> numGuess && !(numGuess > 999 || numGuess <
0))
                    goodInput = true;
                else
                {
                    cerr << "Invalid Input. Input must be between 0-
999(only integers)." << endl;
                    cerr << "Try again: ";
                    cin.clear();
                    cin.ignore();
                }
            }

            sendLong(numGuess, sock);

            roundResult tempRes;
            tempRes.equal = 0;
            tempRes = recResult(sock);

            if (tempRes.equal == 1)
            {
                won = true;
            }

            if (!won)
            {
                cerr << endl
                     << "lower than " << tempRes.tooHigh << endl;
                cerr << "higher than " << tempRes.tooLow << endl;
                cerr << endl;
            }

            roundCount++;
```

首先會印出現在是第幾次猜了，然後會請使用者輸入一個號碼。下面會先檢查使用者輸入的是否是合法(0<=value<=999)的，如果不合法就清掉然後請使用者再重新輸入。如果合法的話，就 send 給 server，接著 server 會進行判斷，然後回傳 result，client 這邊會用 tempRes 來存 result。如果 tempRes 的 equal 是 0 的話，就判定玩家還未獲勝，印出目前可能最大值和最小值。如果 equal 是 1 的話，玩家獲勝，跳出迴圈，然後執行以下程式：

```
        roundCount = receiveLong(sock);

        cerr << "Congratulations! You have won " << "in " << roundCount
<< " turns!" << endl;
```

會從 server 端收 roundCount，然後顯示給玩家知道自己猜了幾輪才猜到。這裡會回到讓玩家選擇要不要繼續遊玩的地方，玩家輸入 s 的話就會重新一輪遊戲，輸入 q 的話就會跳出迴圈，執行以下程式：

```
    status = close(sock);
    if (status < 0)
    {
        cerr << "Error with close" << endl;
        exit(-1);
    }
```

這邊就會把 socket 關掉。

(四) sendLong

Client 的 sendLong 和 server 基本上大同小異：

```
void sendLong(long num, int sock)
{
    long temp = htonl(num);
    int bytesSent = send(sock, (void *)&temp, sizeof(long), 0);
    if (bytesSent != sizeof(long))
        exit(-1);
}
```

(五) recResult

```
roundResult recResult(int sock)
{
    roundResult tempRes;
    roundResult *rPointer = &tempRes;
    int bytesLeft = sizeof(tempRes);
    while (bytesLeft > 0)
    {
        int bytesRecv = recv(sock, (void *)rPointer, sizeof(tempRes),
0);
        if (bytesRecv <= 0)
        {
            cerr << "Error receiving results. Now exiting program.";
            cin.get();
            exit(-1);
        }
        bytesLeft = bytesLeft - bytesRecv;
    }
    tempRes = *rPointer;
    tempRes = notNet(tempRes);
    return tempRes;
}
```

recResult()的實作和 sendResult()蠻像的，先宣告一個 tempRes，然後用一個*rpointer 指到 tempRes。bytesLeft 用來確認目前剩下多少空間。接者，當 bytesLeft>0 的時候，就會用 recv()來收接受到資料的長度(byte)。如果 recv()回傳-1，就代表有錯誤，要通知使用者。然後更新 bytesLeft 數值。最後用 notNet()函式把 tempRes 裡面的數值將 unsigned integer net long 從 network byte order 轉為 host byte order。

(六) receiveLong

```
long receiveLong(int sock)
{
    int bytesLeft = sizeof(long);
    long networkInt;
    char *bp = (char *)&networkInt;

    while (bytesLeft > 0)
    {
        int bytesRecv = recv(sock, (void *)bp, bytesLeft, 0);
        if (bytesRecv <= 0)
        {
            break;
        }
        else
        {
            bytesLeft = bytesLeft - bytesRecv;
            bp = bp +
                 bytesRecv;
        }
    }
    networkInt = ntohl(networkInt);
    return networkInt;
}
```

receiveLong()和 server 的也都相同，就不多作解釋。

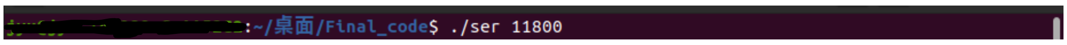
至此 client 端的實作也已經完畢，下面我們來看實際執行的狀況。

參、Execution of each function

首先，server 端會執行以下程式：

```
    if (argc != MAX_ARGS)
    {
        cerr << "Invalid number of arguments. Please input IP address
for first arg, then port # for "
             << " second one. Now exiting program.";
        exit(-1);
    }

    unsigned short portNum = (unsigned short)strtoul(argv[PORT_ARG],
NULL, 0);
    if (portNum > MAXPORT || portNum < MINPORT)
    {
        cerr << "This port is not assigned to this program. Please try
again with " << endl
             << "numbers that are between 11800 & 11899. Now exiting. ";

        exit(-1);
    }
```

使用者必須要輸入正確的 port number，如下圖：

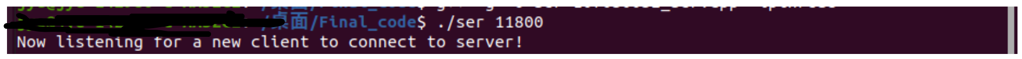


```
[redacted]:~/桌面/Final_code$ ./ser 11800
```

輸入 port number 後，就會執行以下程式，嘗試開啟 socket、bind 和 listen。

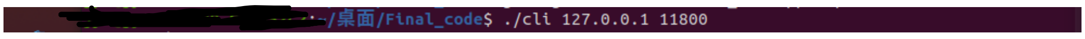
```cpp
    int status;
    int clientSock;

    int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
    {
        cerr << "Error with socket. Now exiting program. " << endl;
        close(sock);
        exit(-1);
    }

    struct sockaddr_in servAddr;
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(portNum);

    status = bind(sock, (struct sockaddr *)&servAddr,
                  sizeof(servAddr));
    if (status < 0)
    {
        cerr << "Error with bind. Now exiting program. " << endl;
        close(sock);
        exit(-1);
    }

    status = listen(sock, MAX_PENDING);
    cerr << "Now listening for a new client to connect to server!" <<
endl;
    if (status < 0)
    {
        cerr << "Error with listen. Now exiting program. " << endl;
        close(sock);
        exit(-1);
    }
```

若都有成功的話，會印出：



```
[redacted]/桌面/Final_code$ ./ser 11800
Now listening for a new client to connect to server!
```

這時再執行 client，輸入 IP address 和 port number，如圖：



```
[redacted]~/桌面/Final_code$ ./cli 127.0.0.1 11800
```

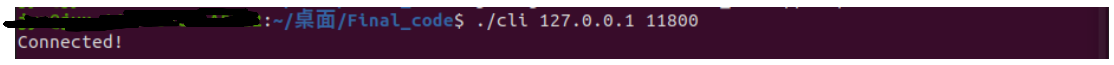
Client 會執行以下程式碼，用來確認輸入的 input 是否有符合規定：

```
    if (argc != MAX_ARGS)
    {
        cerr << "Invalid number of arguments. Please input port # for
first arg. Now exiting program.";
        exit(-1);
    }

    unsigned short portNum = (unsigned short)strtoul(argv[PORT_ARG],
NULL, 0);
    if (portNum > MAXPORT || portNum < MINPORT)
    {
        cerr << "This port is not assigned to this program. Please try
again with " << endl
             << "numbers that are between 11800 & 11899. Now exiting. ";
        exit(-1);
    }
```

若符合規定，client 會嘗試 create socket 並連上 server：

```
    int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
    {
        cerr << "Error with socket" << endl;
        exit(-1);
    }

    status = connect(sock, (struct sockaddr *)&servAddr,
sizeof(servAddr));
    if (status < 0)
    {
        cerr << "Error with connect" << endl;
        exit(-1);
    }

    cerr << "Connected! " << endl
         << endl;
```

成功 connect 上的話，會印出以下結果：

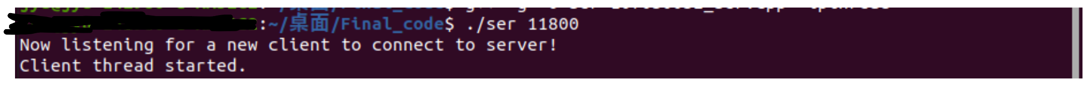


```
[redacted]:~/桌面/Final_code$ ./cli 127.0.0.1 11800
Connected!
```

此時 server 端會執行以下程式來 accept client 端的 connect：

```
    while (true)
    {

        pthread_t tid;

        struct sockaddr_in clientAddr;
        socklen_t addrLen = sizeof(clientAddr);
        clientSock = accept(sock, (struct sockaddr *)&clientAddr,
&addrLen);
        if (clientSock < 0)
        {
            cerr << "Error with accept. Now exiting program. " << endl;
            close(clientSock);
            exit(-1);
        }

        arg_t *args_p = new arg_t;
        args_p->sock = clientSock;

        status = pthread_create(&tid, NULL, func, (void *)args_p);
        if (status)
        {
            cerr << "Error creating threads, return code is " << status
<< ". Now exiting " << endl;
            close(clientSock);
            exit(-1);
        }
        cerr << "Client thread started." << endl;
    }
```

若沒有問題，會印出：

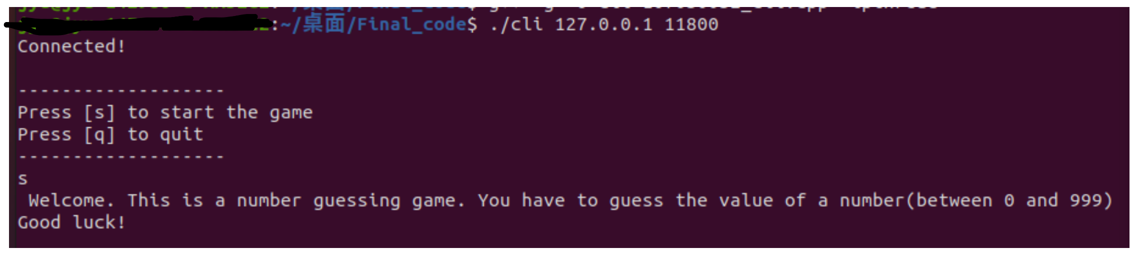


```
[redacted]:~/桌面/Final_code$ ./ser 11800
Now listening for a new client to connect to server!
Client thread started.
```

接著，client 會執行以下程式確認玩家的參與意願，然後傳給 server：

```
        cerr << "-------------------" << endl;
        cerr << "Press [s] to start the game" << endl;
        cerr << "Press [q] to quit" << endl;
        cerr << "-------------------" << endl;
        char signal;
        cin >> signal;
        if (signal == 'q')
        {
            sendLong(1, sock);
            break;
        }
        else
            sendLong(0, sock);
        roundCount = 0;
        won = false;

        cerr << " Welcome. This is a number guessing game. You have to guess the value of a number(between 0 and 999)" << endl;
        cerr << "Good luck!" << endl
             << endl;
```

執行結果如圖：



```
[redacted]:~/桌面/Final_code$ ./cli 127.0.0.1 11800
Connected!

------------------
Press [s] to start the game
Press [q] to quit
------------------
s
 Welcome. This is a number guessing game. You have to guess the value of a number(between 0 and 999)
Good luck!
```

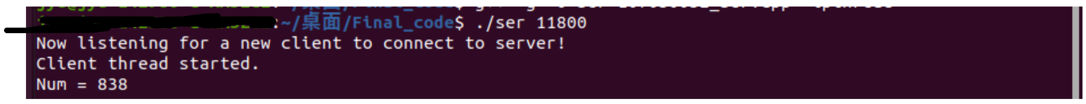
server 端會執行以下程式確認玩家的意願：

```
        wantToPlay = receiveLong(*args_p, exit);
        if (wantToPlay == 1)
        {
            exit = true;
            break;
        }
```

如果玩家確定要玩，server 會執行以下程式：

```
            actualNums = (rand() % 1000);
            cerr << "Num = " << actualNums << endl;
```

server 印出現在產生的數字在 terminal 上：

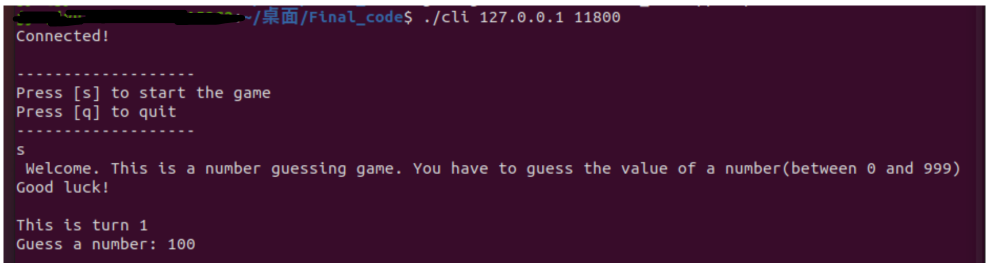


```
[redacted]:~/桌面/Final_code$ ./ser 11800
Now listening for a new client to connect to server!
Client thread started.
Num = 838
```

這時 client 就要開始猜數字了，會執行以下程式碼：

```
            cerr << "This is turn " << (roundCount + 1) << endl;
            cerr << "Guess a number"
                 << ": ";
            goodInput = false;

            while (!goodInput)
            {
                if (cin >> numGuess && !(numGuess > 999 || numGuess <
0))
                    goodInput = true;
                else
                {
                    cerr << "Invalid Input. Input must be between 0-
999(only integers)." << endl;
                    cerr << "Try again: ";
                    cin.clear();
                    cin.ignore();
                }
            }

            sendLong(numGuess, sock);
```

會稍微檢查一下數字有沒有問題，然後送給 server，執行結果如下圖：



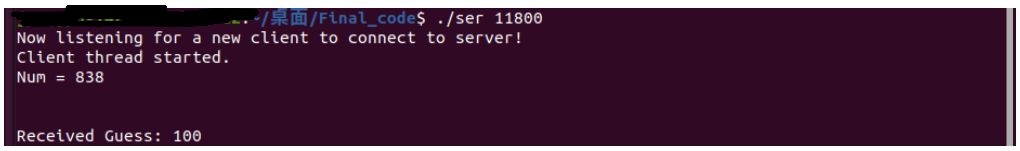
```
[redacted]~/桌面/Final_code$ ./cli 127.0.0.1 11800
Connected!

------------------
Press [s] to start the game
Press [q] to quit
------------------
s
 Welcome. This is a number guessing game. You have to guess the value of a number(between 0 and 999)
Good luck!

This is turn 1
Guess a number: 100
```

server 會收這個數值，並檢查有沒有問題，然後回傳判斷結果給 client，實作如下：

```
                numHigh = 999;
                numOn = 0;
                numLow = 0;
                do
                {
                    numsGuess = receiveLong(*args_p, exit);
                    if (!exit)
                    {
                        cerr << "Received Guess: " << numsGuess << endl;
```

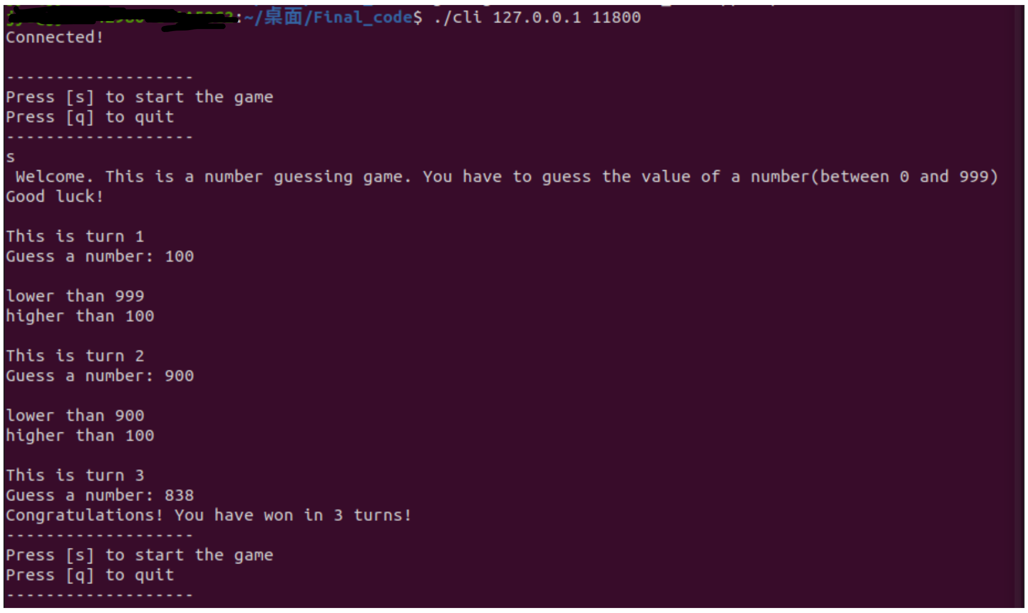
```
            if (numsGuess < actualNums)
                numLow = numsGuess;
            else if (numsGuess > actualNums)
                numHigh = numsGuess;
            else
                numOn = 1;
        }

        result.tooHigh = numHigh;
        result.tooLow = numLow;
        result.equal = numOn;

        sendResult(result, *args_p);

        roundCount++;
```

執行結果如圖：



```
[redacted]:~/桌面/Final_code$ ./ser 11800
Now listening for a new client to connect to server!
Client thread started.
Num = 838


Received Guess: 100
```

Client 收到後，會看是不是對的，如果是對的就跳出迴圈，然後問玩家要不要進行下一輪，若是錯的，就顯示目前可能最大值和可能最小值，如圖：



```
[redacted]:~/桌面/Final_code$ ./cli 127.0.0.1 11800
Connected!

------------------
Press [s] to start the game
Press [q] to quit
------------------
s
 Welcome. This is a number guessing game. You have to guess the value of a number(between 0 and 999)
Good luck!

This is turn 1
Guess a number: 100

lower than 999
higher than 100

This is turn 2
Guess a number: 900

lower than 900
higher than 100

This is turn 3
Guess a number: 838
Congratulations! You have won in 3 turns!
------------------
Press [s] to start the game
Press [q] to quit
------------------
```

到這邊基本上就說明完畢。如果玩家有想要退出就按 q 即可，或按 s 進行下一輪。

參考資料

1. C++ Socket 資料整理
https://dangerlover9403.pixnet.net/blog/post/212391408-%5B%E6%95%99%E5%AD%B8%5Dc++-socket%E8%B3%87%E6%96%99%E6%95%B4%E7%90%86

2. 資訊人筆記 – Socket
https://www.kshuang.xyz/doku.php/programming:c:socket

3. C++ Socket 通訊總結（附 C++實現）
https://www.itread01.com/content/1549920996.html

4. Introduction to Computer Networks - Socket Tutorial
https://elearn.nthu.edu.tw/

5. 到底 ntohl() 與 htonl() 做了什麼?
https://jyhshin.pixnet.net/blog/post/26587993

6. ordonak/number-guessing-game
https://github.com/ordonak/number-guessing-game

7. Introduction to Computer Networks – Lab spec 2021
https://elearn.nthu.edu.tw/