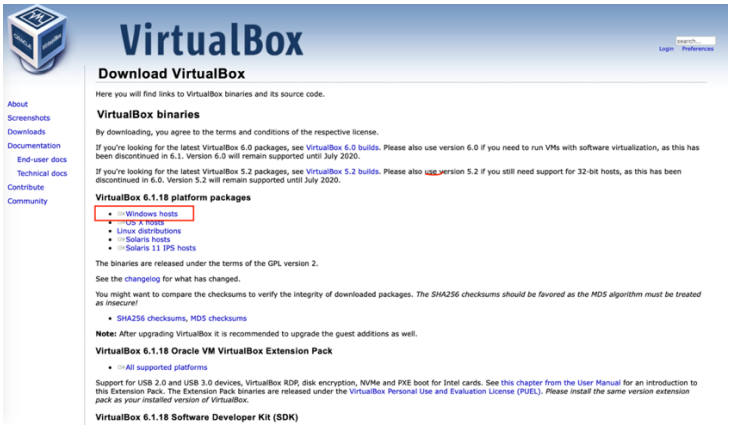
OS project2

E64071237	土木系	朱育萱
E64074120	電機系	吳育駿
E14076538	電機系	劉子溦
F14076130	電機系	陳冠盛

Set up virtual machine environment for Linux OS

安裝虛擬機步驟(VM)

(1)選擇紅框中的 windows hosts 安裝符合 windows 需求的虛擬機



(2)下載及安裝 VirtualBox 虛擬機器軟體

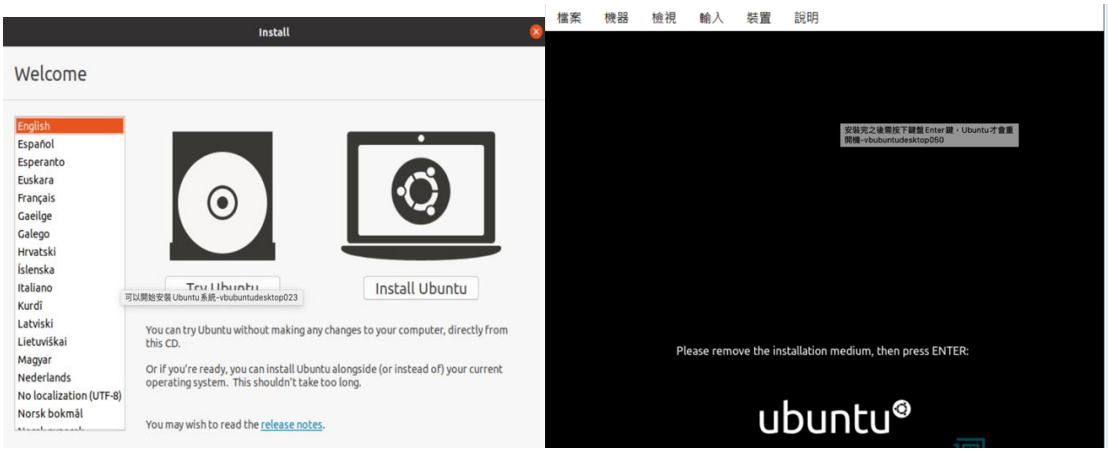
點選紅色的框框中的 windows hosts 安裝符合 windows 需求的虛擬機

(3)在虛擬機之中安裝 Ubuntu 桌面版

打開 Ubantu 後點選裝置並選取要的磁碟機用來安裝

成功選取好磁碟機後就可以看到左下的畫面進行安裝 Ubuntu，安裝完畢後就

可以看到右下的這個畫面，這們一來就完成了 VM 的安裝



README file is needed!

How to run your program

How to use your system

多人聊天室

real-time chatroom

- based on socket & multi-thread

Environment

- ubuntu 19.04

How to run this program

Run GNU make in the repository

```
make
```

Then start

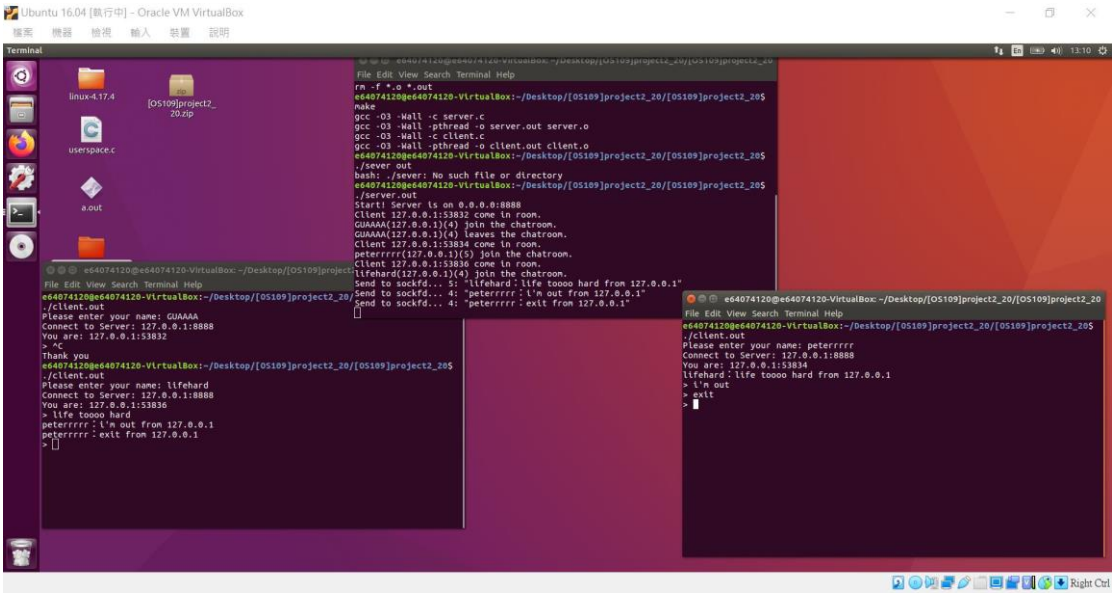
```
./server.out
./client.out
```

How to use this program

client :
(1) User needs to enter nickname first.
(2) If other users send a message, terminla shows who is talking and the message.
(3) If someone leaves chatroom, show [nickname][ip] leaves the chatroom.

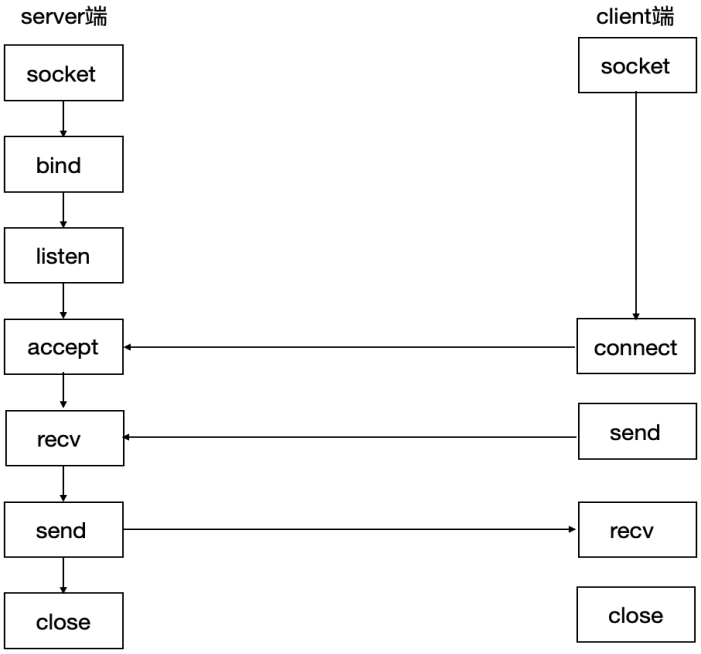
server :
(1) If someone joins chatroom, show its nickname & ip address.
(2) If someone sends a message, send this message to other clients with ip & nickname.
(3) If someone leaves chatroom, show [nickname][ip][sockedfd] leaves the chatroom.

Command	Describtion
/exit	Leave the chatroom
/connecting	Test connection, responds with socketfd
/help	Show this help



Briefly describe the workflow and architecture of the system.

Workflow:



我們所是依照 socket 的 TCP 架構去做設計的 workflow，如上圖

server 端

(1)socket 函數：

主要是用來生成一個通訊端點，使用者或應用程式只要連接到 socket， 便可以和網路上任何一個通訊端點連線

```
//建立一個socket
server_sockfd = socket(AF_INET , SOCK_STREAM , 0);
if (server_sockfd == -1) {
    printf("Fail to create a socket.");
    exit(EXIT_FAILURE);
}

// 建立Socket所用到的資訊
struct sockaddr_in server_info, client_info;
int s_addrlen = sizeof(server_info);
int c_addrlen = sizeof(client_info);
memset(&server_info, 0, s_addrlen);
memset(&client_info, 0, c_addrlen);
server_info.sin_family = PF_INET;
server_info.sin_addr.s_addr = INADDR_ANY;
server_info.sin_port = htons(8888);
```

(2)bind 函數：

用來綁定一個通訊端點和 IP 地址，使接口與指定的通訊端點和 IP 地址相關聯。

```
bind(server_sockfd, (struct sockaddr *)&server_info, s_addrlen);
```

(3)listen 函數：

使 server 的這個端口和 IP 處於監聽狀態，等待網路中某個 client 的連接請求，如果 client 端有連接請求，通訊端點就會接受這個連接。

```
listen(server_sockfd, 5);
```

(4)accept 函數：

遠程計算機的連接請求建立起與這個請求之間的通信。當出現監聽狀態時，如果某個時刻出現連接請求，則不是立即處理這個請求，可能將請求請求等待中，等到系統空閒的時候再處理的連接請求。當 accept 函數接受一個連接時，

會到一個新的 socket，之後的資料傳輸和讀去就要透過這個 socket，而原來的 socket 也可以繼續使用，繼續監聽其他連接請求。

```
client_sockfd = accept(server_sockfd, (struct sockaddr*) &client_info, (socklen_t*) &c_addrlen);
```

(5)recv 函數：

用新的套接字來接收遠端主機傳來的數據，並把數據存到由參數 buf 指向的內存空間。recv 這個函數用在兩個地方第一個是在接收名字的時候，如以下 code 如果名字長度等小於 2 或大於 30 則不執行,也就是可以解釋成沒有收到 input，經 socket 接收的名字小於等於 0 也不執行,也就是可以解釋成沒有收到 input

```
if (recv(np->data, nickname, LENGTH_NAME, 0) <= 0 || strlen(nickname) < 2 || strlen(nickname) >= LENGTH_NAME-1) {
    printf("%s didn't input name.\n", np->ip);
    leave_flag = 1;
} else {
    strncpy(np->name, nickname, LENGTH_NAME);
    printf("%s(%s)(%d) join the chatroom.\n", np->name, np->ip, np->data);
    sprintf(send_buffer, "%s(%s) join the chatroom.", np->name, np->ip);
}
```

第二個部分是在 conservation 的時候，如以下的 code 簡單來說就是如果 client 有傳 message 則 server 會顯示出把這個 message 傳到其餘每個 client socket，並且顯示出 IP 位置和 client 名稱，若有 client 離開聊天室則顯示出是哪個 client 離開聊天室。

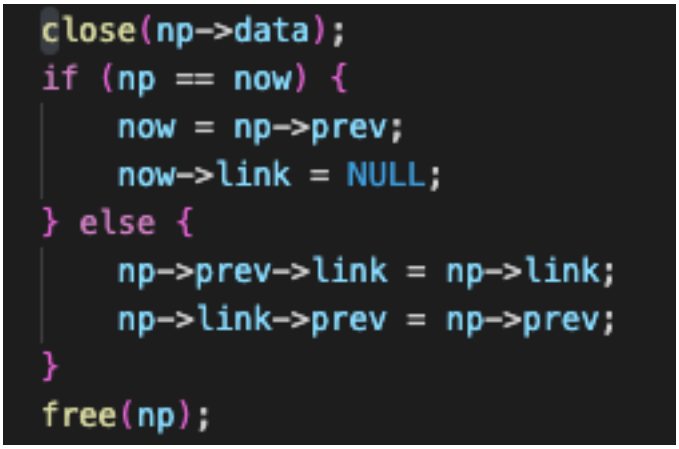
```
int receive = recv(np->data, recv_buffer, LENGTH_MSG, 0);
if (receive > 0) {
    if (strlen(recv_buffer) == 0) {
        continue;
    }
    sprintf(send_buffer, "%s：%s from %s", np->name, recv_buffer, np->ip);
} else if (receive == 0 || strcmp(recv_buffer, "/exit") == 0) {
    printf("%s(%s)(%d) leave the chatroom.\n", np->name, np->ip, np->data);
    sprintf(send_buffer, "%s(%s) leave the chatroom.", np->name, np->ip);
    leave_flag = 1;
} else {
    printf("Fatal Error: -1\n");
    leave_flag = 1;
}
sendAll(np, send_buffer);
```

(6)send 函數：

藉由 server 把接收到的訊息傳遞給每個 client 的 terminal 上

(7)close 函數：

當使用完資料後如果已經不再需要，則可使用 close()關閉該資料，並且 close() 會讓數據寫回 disk，並釋放該資料所佔有的資源。如以下的 code，這裡的功能是用來把負責連接 client 的 node 去做一個刪除的動作。



client 端

(1)socket 函數：

和 server 端的用意是差不多的一樣式用來生成一個通訊端點，讓使用者或應用程式只要連接到 socket 便可以和網路上任何一個通訊端點連線。

```c
sockfd = socket(AF_INET , SOCK_STREAM , 0);
if (sockfd == -1) {
    printf("Fail to create a socket.");
    exit(EXIT_FAILURE);
}
// 建立Socket所用到的資訊
struct sockaddr_in server_info, client_info;
int s_addrlen = sizeof(server_info);
int c_addrlen = sizeof(client_info);
memset(&server_info, 0, s_addrlen);
memset(&client_info, 0, c_addrlen);
server_info.sin_family = PF_INET;
server_info.sin_addr.s_addr = inet_addr("127.0.0.1");
server_info.sin_port = htons(8888);
```

(2)connect 函數：

用來請求連接遠程 server，將參數 sockfd 的 socket 連至參數 server_info 指定的服務器 IP 和接點上，s_addrlen 為變量長度。如同以下的 code 是負責與 sever 做連接，並用 0 跟-1 去判斷是否可執行。

```c
int err = connect(sockfd, (struct sockaddr *)&server_info, s_addrlen);
if (err == -1) {
    printf("Connection to Server error!\n");
    exit(EXIT_FAILURE);
}
//在client 跟 sever 中顯示
getsockname(sockfd, (struct sockaddr*) &client_info, (socklen_t*) &c_addrlen);
getpeername(sockfd, (struct sockaddr*) &server_info, (socklen_t*) &s_addrlen);
printf("Connect to Server: %s:%d\n", inet_ntoa(server_info.sin_addr), ntohs(server_info.sin_port));
printf("You are: %s:%d\n", inet_ntoa(client_info.sin_addr), ntohs(client_info.sin_port));
```

(3)send 函數：

把接收到的訊息傳遞給 server，如下圖的 code 是用來將 message 傳出去的

```c
void SendMsgController() {
    char message[LENGTH_MSG] = {};
    while (1) {
        printf("\r%s", "> ");
        fflush(stdout);
        while (fgets(message, LENGTH_MSG, stdin) != NULL) {
            str_trim_lf(message, LENGTH_MSG);
            if (strlen(message) == 0) {
                printf("\r%s", "> ");
                fflush(stdout);
            } else {
                break;
            }
        }
        send(sockfd, message, LENGTH_MSG, 0);
        if (strcmp(message, "/exit") == 0) {
            break;
        }

    }
    flag = 1;
}
```

(4)recv 函數：

同樣也是和 sever 的功能一樣，用新的套接字來接收遠端主機傳來的數據，並把數據存到由參數 buf 指向的內存空間。如下圖的 code 是用來接收 message 的

```c
void RecvMsgController() {
    char receiveMessage[LENGTH_SEND] = {};
    while (1) {
        int receive = recv(sockfd, receiveMessage, LENGTH_SEND, 0);
        if (receive > 0) {
            printf("\r%s\n", receiveMessage);
            printf("\r%s", "> ");
            fflush(stdout);
        } else if (receive == 0) {
            break;
        }
    }
}
```

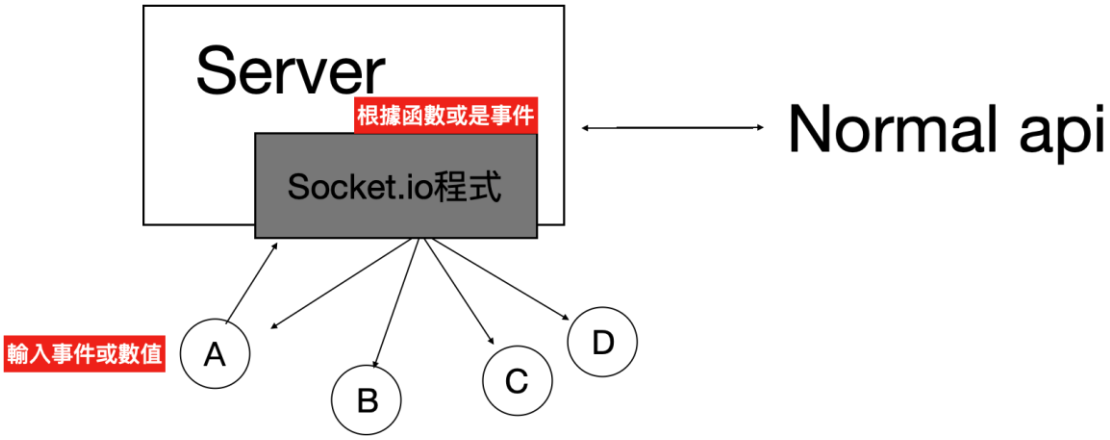
(5)close 函數：

使用完資料後如果已經不再需要，則可使用 close()關閉該資料

```c
close(sockfd);
```

Architecture:

terminal 執行的時候若要多人聊天則要開啟不同的視窗去做操作，假設如果有三個人要聊天則需要開四個 terminal，一個當做 server 另外三個當作是 client，這是我們將上述的 workflow 實現的結果，而下圖為我們的架構。



每個 client 要傳送訊息給彼此的時候都是先將資料傳送到 server 端，包含了 client 的 socket port、ip 以及想傳的 message body，而 server 端接收到 client 端傳來的 request，sever 再根據每一個 client 的(IP +port number = socket)傳送對應的訊息，讓每個 client 都可以看到傳送者的資訊及他想分享的訊息內容。

整體來說是類似廣播的概念，想講話的話並不是直接跟所有人講，而是先跟電台講，再透過廣播電台將聲音發送給每個聽眾知道。

Explain the solution of synchronization problems in this project.

程式設計中一共有三個 thread : send_msg_thread、recv_msg_thread、id

這三個 thread 不會出現 race condition，並且程式也沒有 shared memory 等會發生 race condition 的情況，此外 client 不能同時發送訊息到 server 所以也不會有 race condition 的問題發生，因此不會有 synchronization 的問題。其中第一個 mutual exclusion 因為上面講到的我們在進行操作時 client 無法同時發送訊息所以有 process 在 critical section 裡面其他 process 就不可以進去，這個條件是符合

的，第二個 progress 也因為這個原因 critical section 裡面沒有 process 時，可以透過 terminal 的操作讓下一個 process 進去，最後第三個 bounded waiting 在 client 跟 server 互相傳送的過程中不會讓 process 一直站在 critical section 裡面，會在有限的時間內換給下一個 process。

Discuss anything as possible.

(1)我們寫了三個.c 檔案，一個分別是 client.c、server.c，還有一個 server.h 用來作為放一些常用的函式如下圖，因為 sever 是用來控制 client 的畫面，所以必須要有 linklist 用來控制，而這個檔案是負責用 node 來存放總共有幾個 client 而另外初始化一個 linklist 來進行控制。

```c
#ifndef LIST
#define LIST

typedef struct ClientNode {
    int data;
    struct ClientNode* prev;
    struct ClientNode* link;
    char ip[16];
    char name[31];
} ClientList;
//
ClientList *newNode(int sockfd, char* ip) {
    ClientList *np = (ClientList *)malloc( sizeof(ClientList) );
    np->data = sockfd;
    np->prev = NULL;
    np->link = NULL;
    strncpy(np->ip, ip, 16);
    strncpy(np->name, "NULL", 5);
    return np;
}

#endif // LIST
```

(2)bind 和 listen 舉例解釋

因為在上面 server 端 workflow 中並沒有很詳細的介紹兩個函數的關係所以在這邊額外講解一下，可以將這兩者的運作模式比喻成港口跟客人的船的關係，在客人的船要從哪個港口要做登陸，必需要去反覆去查看船來了沒有，這個過程就叫做監聽，對應到了 socket 的 listen()。但是因為 server 一次只能服務一個，當港口出入較為頻繁的時候，我們要讓來船照拜訪的先後排成隊列，即是說每當一個請求送到 server，socket 就會把它丟到監聽隊列的尾端。