OS final project(team21)

組員：E94081107 范姜揚/ E24086242/黃柏叡 / E24086496 陳柏翰 / E24084172 謝亞城

Our Github Repo (程式碼以github 為主，有微調!謝謝助教)

GitHub - coding-ray/OS-project-2

Contribute to coding-ray/OS-project-2 development by creating an account on GitHub.

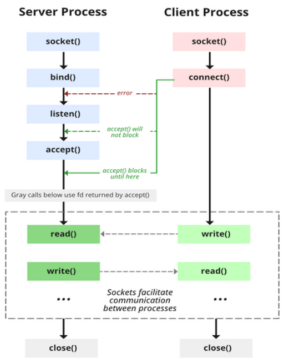
https://github.com/coding-ray/OS-project-2?fbclid=IwAR1OU4sSM2N5N7dsSPm3zLxrGy6MoOYCH0BhcZlXnb-GE9eG9XHHt9730SU

coding-ray/**OS-project-2**

2 Contributors 0 Issues 0 Stars 2 Forks

Socket Basics

Socket programming is a way of connecting two nodes on a network to communicate with each other.



State diagram for server and client model of Socket

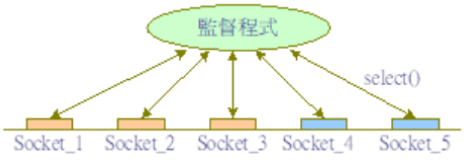
用socket 設計聊天室有以下兩種大方向:

(1) 當 Socket 被執行 listen() 系統呼叫（虛擬電路方式），而進入聆聽狀態之後，不要即時執行 read() 呼叫（或 recvfrom() 呼叫）。而先利用 select() 去詢問連線是否有資料進來，如果資料已進入，在執行 read() 呼叫來讀取該資料，如此執行 read() 呼叫就不會無窮的等待著。因此，一個主程式就可以監視多個 Socket 端點要求連線，或處理多個端點的傳輸資料。

(2) 另一種方法是 Socket 隨時監視是否有訊號進入，也就是直接執行 read() 系統呼叫等待接收訊號。但當收到遠端連接訊號後，立即利用 fork() 系統呼叫產生子程序（Child Process），由子程序負責連線處理的工作，而原來主程序（Main Process）再回去監視是否有其它遠端要求連線，如此，便可以達到一個 Socket 的通訊端點，可以接受多個使用者要求連線。一般 Internet 上的 Client/Server 架構上的伺服器都可同時接受多個使用者連線，下一節在詳加介紹其製作方法。

寫聊天室要注意的觀念：

- enable **“multiple I/O “ :**



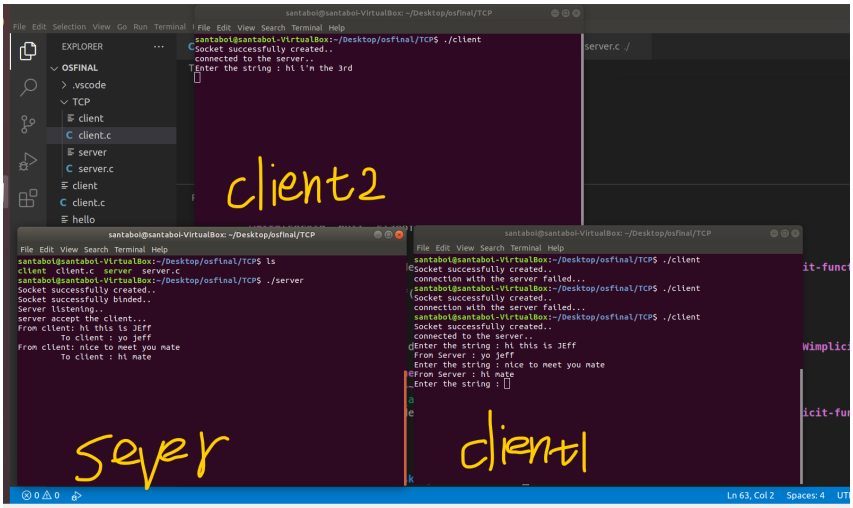
Socket 『**多工輸入/輸出**』（**Multiple I/O**）
主程式要允許多個 Socket 端資料的進出。可以使用**poll()，select()**。現在多使用 **select()，用多個socket公用一個監視埠，**用 fd_set 來存放被監視的 Socket 端點的識別碼：fd_set = (socket_1, socket_2, socket_3, socket_4, socket_5)，而其中任何一個 Socket 有事件發生（Read, Write, Exception），回應給主程式後再由主程式來處理該事件。

One_to_One_ChatRoom(實作)

- TCP is suited for applications that require high reliability, and transmission time is relatively less critical
- TCP handles reliability and congestion control. It also does error checking and error recovery

以下是用最簡單的TCP library 去寫出來的，可以實現一對一的聊天室。一個server只能對應到一位client，以下圖為例server跟client1 可以正常收發訊息，但client2產生deadlock。

註：一對一就不用擔心到不同thread 動到相同的buffer位置的問題 ，可以看到當開到第二個client連到server時他就無法去處理



Many_to_Many_ChatRoom(實作)

(使用C語言 ，無法用class)，故使用**nested struct 去存放聊天室的傳輸溝通的必要資料**

這些資料結構放在header : **CHATHEADER.h**

- server端重要資料結構
  - char_queue 用來存放要傳送的messages ，有設定預設大小(要夠大)，預防一堆人同時傳訊息queue會報掉，給reader writer 當buffer

```c
typedef struct
{
    pthread_mutex_t *mutex;
    pthread_cond_t *notFull, *notEmpty;
    int head, tail;
    int full, empty;
    char *buffer[MAX_BUFFER];
} chat_queue;
```

- server 的重要資料 (不包成struct也可以)，但全部設在global會非常亂
  - 內含有list of socket , mutex list , message queue

```c
typedef struct
{
    pthread_mutex_t *clientListMutex;
    fd_set serverReadFds;

    int socketFd;
    int clientSockets[MAX_BUFFER];
    int numClients;

    chat_queue *queue;
} Chat_datastruct;
```

- client端資料結構
  - 取用server端資料 Chat_datastruct , 還有各自的socket_fd

```c
typedef struct
{
    Chat_datastruct *data;
    int clientSocketFd;
} clientHandlerVars;
```

multithreading 機制

multithreading 參考經典的 read write problem 設計，包含consumer 跟 producer。

- **consumer** — **consume data and “dequeue”**
- **producer** — **generate data and “enqueue”**
- **mutex lock** — 保護buffer
- **condition variable** — 判斷buffer “**空**” 或 ”**滿**” 的狀態
- 若多個threads 去完成 read write problem，則資料結構queue 共用需要 mutex lock 保護機制
- **avoid deadlock : 一個client 連進來就讓clientnum加1，mutex list 跟client相關結構都會變大，故理論上不會有deadlock，除非超過socket系統負荷**

註 : 程式實作，用read write problem 去處理messages storage，傳遞溝通的部分用web socket programming + TCP 的概念去做。

multithreading 程式碼

- 使用 **pthread (在compile時候要加入 -phtread 或 -lpthread 讓 compiler gcc 知道)**
- **server.c 共開啟兩種thread**
  - thread1**(處理connection)**

```
pthread_t connectionThread;
if ((pthread_create(&connectionThread, NULL, (void *)&newClientHandler, (void *)&data)) == 0)
{
    fprintf(stderr, "******************NCKU OS_ChatRoom started******************\n");
}
```

- thread2 **(處理messege read write)**
  - client 連到 server 時即配給一個thread

```
pthread_t messagesThread;
if ((pthread_create(&messagesThread, NULL, (void *)&messageHandler, (void *)&data)) == 0)
{
    fprintf(stderr, "allow clients now.........\n");
}
```

- **critical section**
  - mutex lock 是設計給threads之間共用的，一次只讓一個人進入**critical section**
  - 有使用一般 mutex lock
  - 也有使用condition variable ，幫助判斷condition且達成critical section 保護的功能，

    If you just need mutual exclusion, then condition variables don't do anything for you. However, if you need to know when something happens, then condition variables can help.

    - condition variable 在此處初始化

```
pthread_cond_init(q->notFull, NULL);
q->notEmpty = (pthread_cond_t *)malloc(sizeof(pthread_cond_t));
if (q->notEmpty == NULL)
{
    perror("Couldn't allocate anymore memory!");
    exit(EXIT_FAILURE);
}
pthread_cond_init(q->notEmpty, NULL);
```

    - 在動到chat data struct(內部client連接狀態跟數量時)，也用**mutex lock** 保護住
      - 圖一(remove client，確保threads不會同時改動data)

```
pthread_mutex_lock(data->clientListMutex);
for (int i = 0; i < MAX_BUFFER; i++)
{
    if (data->clientSockets[i] == clientSocketFd)
    {
        data->clientSockets[i] = 0;
        close(clientSocketFd);
        data->numClients--;
        i = MAX_BUFFER;
    }
}
pthread_mutex_unlock(data->clientListMutex);
```

      - 圖二 (add client ，確保threads不會同時改動**chatdata**)

```
// Obtain lock on clients list and add new client in
pthread_mutex_lock(chatData->clientListMutex);
if (chatData->numClients < MAX_BUFFER)
{
    // Add new client to list
    for (int i = 0; i < MAX_BUFFER; i++)
    {
        if (!FD_ISSET(chatData->clientSockets[i], &(chatData->serverReadFds)))
        {
            chatData->clientSockets[i] = clientSocketFd;
            i = MAX_BUFFER;
        }
    }

    FD_SET(clientSocketFd, &(chatData->serverReadFds));

    // Spawn new thread to handle client's messages
    clientHandlerVars chv;
    chv.clientSocketFd = clientSocketFd;
    chv.data = chatData;

    pthread_t clientThread;
    if ((pthread_create(&clientThread, NULL, (void *)&clientHandler, (void *)&chv)) == 0)
    {
        chatData->numClients++;
        fprintf(stderr, "new client is joined successfully!! It's Socket_fd : %d\n", clientSocketFd);
    }
    else
        close(clientSocketFd);
}
pthread_mutex_unlock(chatData->clientListMutex);
```

- **pthread_cond_wait + pthread_cond_signal (用mutex 保護住)**
  - 處理message時用以保護 (**busy waiting + mutex lock**)

```
// Obtain lock and pop message from queue when not empty
pthread_mutex_lock(q->mutex);
while (q->empty)
{
    pthread_cond_wait(q->notEmpty, q->mutex);
}
char *msg = queuePop(q);
pthread_mutex_unlock(q->mutex);
pthread_cond_signal(q->notFull);
```

  - 將message推到聊天室的資料結構時候也用到 (**busy waiting + mutex lock**)

```
// Wait for queue to not be full before pushing message
while (q->full)
{
    pthread_cond_wait(q->notFull, q->mutex);
}

// Obtain lock, push message to queue, unlock, set condition variable
pthread_mutex_lock(q->mutex);
fprintf(stderr, "message is pushed to queue (from client-> %s) \n", msgBuffer);
queuePush(q, msgBuffer);
pthread_mutex_unlock(q->mutex);
pthread_cond_signal(q->notEmpty);
```

程式執行結果

a. **測試環境 Ubuntu 18.04**

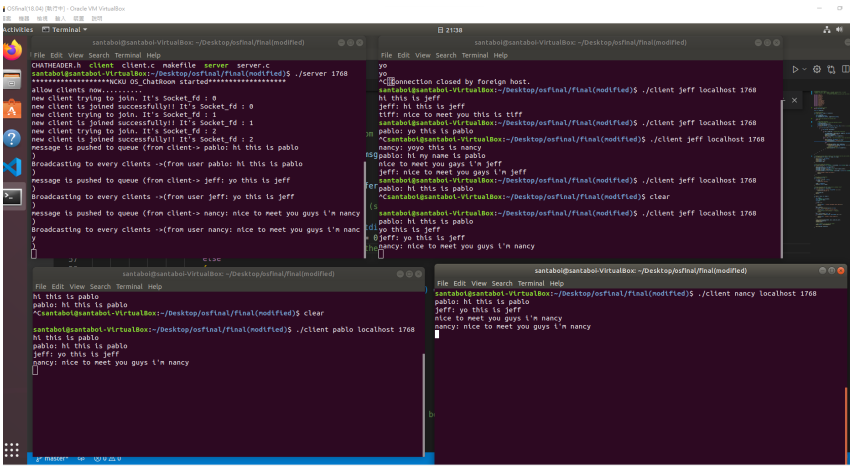
b. **都在同一個machine上面跑，開多個terminal ，也許可以用putty 連 (可能會遇到防火牆問題，沒時間測試)**

c. **以下詳細執行可參考readme**

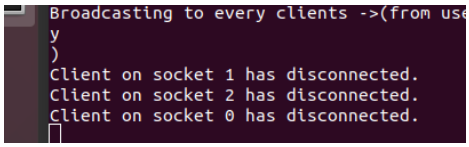
```
//run server
$./server [port]
//run client
$./client <host> <port>
```

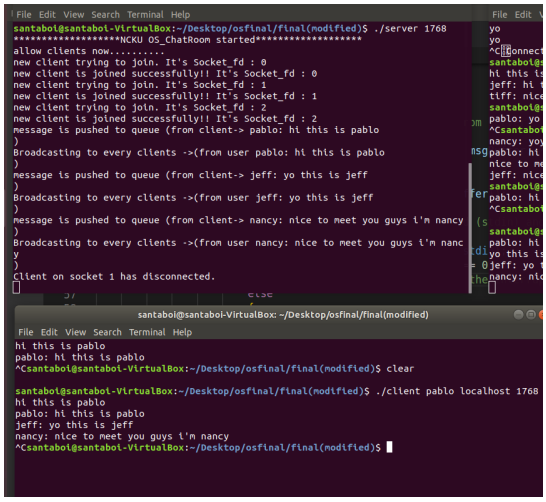
d. **client端 也可以直接使用符合TCP協定的 telnet 去跑**

執行結果一 (**client server**)

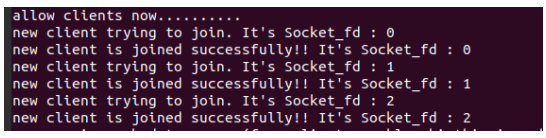


- 左上角視窗是server
  - server 顯示 (**client 退出**)

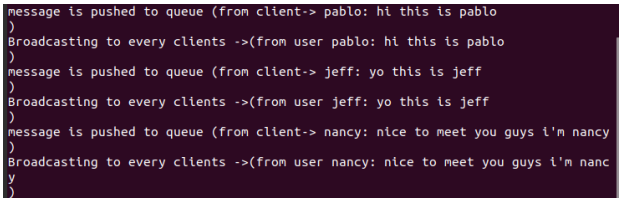




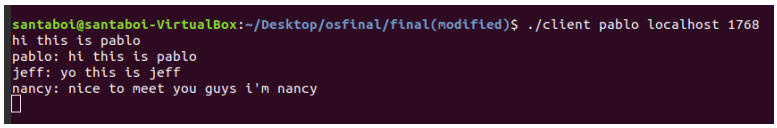
  - **Server** 顯示 client 加入

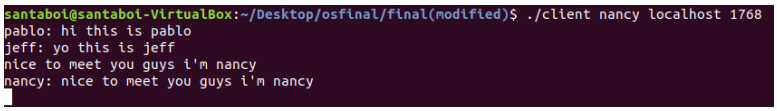


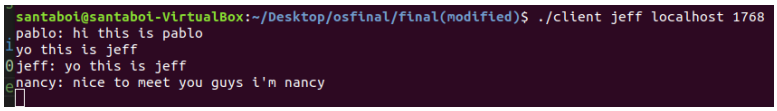
  - 顯示收發的訊息(多對多由server當中繼點)
    - **enqueue , broadcasting**



- 其他三個視窗是加入chatroom 的三個 client (user 的聊天窗口)



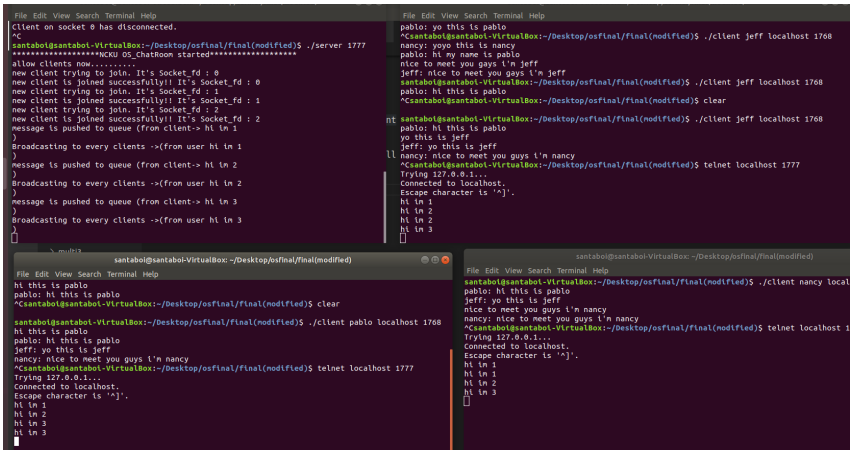


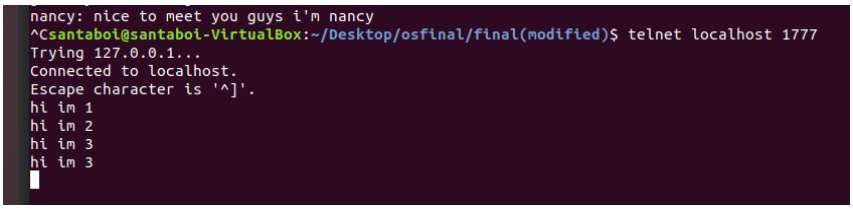


執行結果二(**server telnet**)

若沒寫client 的替代方案

- 因為是直接用telnet 去連線 ， 故server 無法抓到是誰傳訊息 **但訊息傳遞是正常的**





進階探討與分析(主要對比select()跟multithreading)

multi-threading 難寫而且若寫不好容易有deadlock，或是整個系統崩潰的狀況。若不想用multithreading 又要實作many to many chatroom。可以使用select() function。

select() 優點：

a. **好寫!!!**

b. select() is that your server will only require a single process to handle all requests. Thus, your server will **not need shared memory** or **synchronization primitives** for different 'tasks' to communicate.

c. **select() 授予同時監視多個 sockets 的權力，它會告訴你哪些 sockets 已經有資料可以讀取、寫入、execption**

d. 並非真正的多線程，而是藉由library 的函數去設定timeout timeinterval，達到多人asynchronous

select() 缺點：

a. with a fork()'ing solution, after the server fork()s, the child process works with the client as if there was only one client in the universe -- the child does not have to worry about new incoming connections or the existence of other sockets. **With select()**
, the programming isn't as transparent.

b. 對cpu 好費較大，要一直循環確認

c. 沒有利用到現在cpu 多核心的功能，理論上效能不會到multithreading 那麼好

使用multi-threading感想：

為何使用multi-threading：
一個process中常會使用fork產生child process來處理client，但fork需要付出的代價是很大的，需要將整個parent的memory複製到child，儘管有copy-on-write（child需要的時候才copy）可以優化，代價還是很大，且還有一點是要將資訊回傳給parent需要費一些功夫，相較於threads可以避免這兩個問題，因為我們知道threads又被稱作lightweight processes，他會比process還輕，create一個thread至少比process快上10倍，這是因為threads共享global memory這讓資訊的傳遞在threads 中更加容易，但需要注意synchronization的問題。

Troubleshooting

- 1 遺漏pthread lib

```
santaboi@santaboi-VirtualBox:~/Desktop/osfinal/multi$ gcc m_client.c -o m_client
/tmp/cctpl67V.o: In function `main':
m_client.c:(.text+0x14e): undefined reference to `pthread_create'
m_client.c:(.text+0x181): undefined reference to `pthread_create'
m_client.c:(.text+0x1ac): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status
```

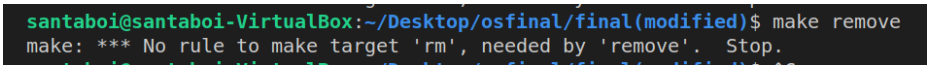
solve : should manually add -phread in cmd

```
santaboi@santaboi-VirtualBox:~/Desktop/osfinal/multi$ gcc m_client.c -o m_client -pthread
```

- 2 (When you run the client from a different Linux/Unix server, please consider the firewall issues.)
- 3 若在傳送訊息時候沒有flush掉print buffer 在多人同時傳訊息時有機會亂掉

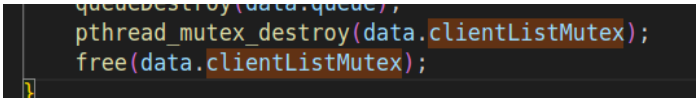
```
//在每次要印出的訊息結束後
fflush( stdout );
//才不會有東西一直卡在buffer印不出來造成訊息處理混亂
```

- 4 makefile error

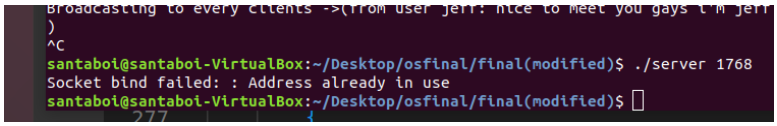


formatting

- 5 忘記free 掉mutex
  - 切記創立mutext結束後要free



- 6 port is already used



  - **server**加入**interrupt handler** 才能安全跟穩定的把client server端關掉，若沒加入handle interrupt機制容易出現以下狀況，直接關閉terminal時造成port實際上沒有關掉，造成短時間連不到相同port狀況

reference : https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

https://www.geeksforgeeks.org/socket-programming-cc/

https://stackoverflow.com/questions/45457764/how-to-handle-multiple-clients-in-socket-programming

https://blog.csdn.net/chenglibin1988/article/details/8812506?spm=1001.2101.3001.6650.7&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~default-7-8812506-blog-91357225.pc_relevant_multi_platform_whitelistv1&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~default-7-8812506-blog-91357225.pc_relevant_multi_platform_whitelistv1&utm_relevant_index=13

http://www.tsnien.idv.tw/Internet_WebBook/chap8/8-8 Socket 多工方式.html

https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/

https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/?ref=rp