Socket programming HW2

資管五 徐遠志 b05705046

- **如何 Compile**

開啟 terminal 然後 cd 到 server.cpp 的目錄下，執行：

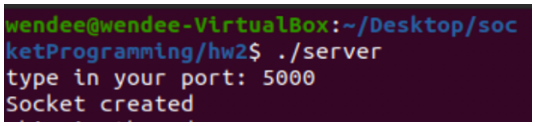
`make`

- **如何執行程式 (包含參數說明)**

在 terminal 執行：

`./server`

接著程式會問 server 要聽哪一個 port，就輸入 port:



看到 Socket created 就表示成功了

- **程式需求、執行需求或環境**

虛擬機器安裝 linux 20.04 版本的 ISO 檔

需要 g++ 進行編譯

- **程式邏輯說明與截圖搭配**

由於 server 需要可以服務多個 client，所以我們會先設定可以服務的 client 數量上限 MAX_CLIENT，並利用 pthread_create 開好足夠數量的 thread

```
void Host::Start()
{
    for (int i = 0; i < MAX_CLIENT; i++)
    {
        int ret = pthread_create(&my_thread[i], NULL,
        &Host::client_thread_helper, (void *)i);
        if (ret != 0)
        {
            cout << "Error: pthread_create() failed\n";
            exit(EXIT_FAILURE);
        }
    }

    pthread_join(my_thread[0], NULL);
}
```

針對每一個 thread ，我們讓它等著服務 client，只要一有連線 request 被 accept 之後，server 就會回傳 Connection accepted 的回應給 client，並開始接收 client 的訊息給回應

```
while (true)
{
    int client_sock;
    client_sock = accept(server_sock,      You, seconds ago • Uncommitted cha
    (sockaddr *)&newSocketAddr, &newSocketSize);
    if (client_sock < 0)
    {
        cout << "client_sock fail: " << client_sock << endl;
        cerr << "Can't accepting the request from client!" << endl;
        exit(0);
    }
    cout << "Connection accepted from " << inet_ntoa(newSocketAddr.sin_addr)
    send_data(client_sock, "Connection accepted!");

    while (true)
    {
        receive(client_sock);
    }
}
```

收到 client 傳來的 command 之後，我們會先進行字串的前處理（依據#字號進行斷字），再呼叫 handleEvent 來決定要回傳什麼資料。

```
void Host::receive(int client_sock)
{
    char buffer[2000] = {0};
    memset(buffer, '\0', sizeof(buffer));

    if (recv(client_sock, buffer, sizeof(buffer), 0) > 0)
    {
        cout << "client :" << buffer << "\n";
        string data = buffer;
        string response =
        handleEvent(client_sock, split_str(data));      You,
    }
}
```

在 handleEvent 裡面，會依照剛剛前處理切出來的關鍵字去進行比對，再來決定下一步需要呼叫哪個對應的函式

```
if (contains(tokens[0], "Exit") && tokens.size() == 1)…
else if (contains(tokens[0], "REGISTER") && tokens.size() == 3)
{
    response = RegisterAccount(tokens);
}
else if (contains(tokens[0], "List") && tokens.size() == 1)
{
    response = GetOnlineAccounts(client_sock);
}
else if (tokens.size() == 2)
{
    response = Login(client_sock, tokens[0]);
}
else
{
    response = "please check your command format \n";
}
```

在 RegisterAccount 這個函式裡，我們會先檢查想要新註冊的帳號是否已經註冊過，沒有的話，我們就新增一個 Account 物件，設定好帳號名稱、金額之後，加入總帳號名單裡

```
bool accountExist = false;
for (int i = 0; i < accounts.size(); i++)
{
    if (accounts[i].name == name)
    {
        accountExist = true;
    }
}

if (accountExist)…
else
{
    Account newUser(name, stoi(data[2]));
    accounts.push_back(newUser);
    return "100 OK\n";
}
```

登入的時候，我們首先會透過比對姓名來檢查這個帳號是否存在，並同時根據帳號的 sd 屬性來看連線狀態，把已經被登入的情況排除，當確定帳號存在且尚未登入，我們就可以在修改該帳號的連線狀態後回傳登入成功提示給 client

```
int accountIndex = -1;
bool isLogined = false;
for (int i = 0; i < accounts.size(); i++)
{
    if (accounts[i].name == name)
    {
        accountIndex = i;

        if (accounts[i].sd != -1)
        {
            isLogined = true;
        }
        else
        {
            accounts[i].sd = sd;
        }
    }
}
```

由於登入和 List 都需要列出上線人數和相關資料，因此，我們可以透過每個帳號的 sd 屬性，若是 -1 以外的值，則表示其正在連線，透過這個方式我們就可以找出所有上線的用戶。

```
for (int i = 0; i < accounts.size(); i++)
{
    if (accounts[i].sd != -1) // loggined
    {
        if (accounts[i].sd == sd)      You, 3 hours ago • List
        {
            balance = accounts[i].balance;
        }

        numLoggined++;
        accountsLoggined += accounts[i].name +
         "#" + server_addr + "#" + to_string(server_port) + "\n";
    }
}
```