ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

Môn học: Mạng Máy Tính

Học kỳ 2 (2019 - 2020)

Lớp 17CK2

Đề tài: Socket

Sinh viên thực hiện đồ án:

Hồ Thiên Phúc MSSV: 1760147

Lưu Mạnh Quan MSSV: 1760151

Thành phố Hồ Chí Minh, tháng 7 năm 2020

I. THÔNG TIN NHÓM

a. Các thành viên trong nhóm

Thành viên 1: Hồ Thiên Phúc – MSSV: 1760147
Thành viên 2: Lưu Mạnh Quan – MSSV: 1760151

b. Bảng phân chia công việc

	1760147	**1760151**
Tạo kết nối	Có (dùng đa tiểu trình**)	Có (dùng tuần tự***)
Đọc dữ liệu từ file lên server	Không	Có
Gửi dữ liệu sang client	Có *	Có
Xử lý các truy vấn từ client	Có *	Có
Gửi kết quả truy vấn về client	Có *	Có
Client nhận dữ liệu từ server	Có *	Có
Client hiển thị dữ liệu cho User	Có *	Có
Client nhận yêu cầu từ User	Có *	Có
Client gửi truy vấn cho server	Có *	Có
Client nhận kết quả và trả về User	Có *	Có
Chống trường hợp race condition	Có	Không
Đóng kết nối	Có (dùng đa tiểu trình**)	Có (dùng tuần tự***)

** 1760151 đã làm xong phần này và 1760147 tối ưu code để cải thiện hiệu năng.*
*** Kết nối 1-n theo mô hình multithread.*
**** Kết nối 1-n theo mô hình tuần tự.*

II. NGÔN NGỮ VÀ MÔI TRƯỜNG SỬ DỤNG

a. Ngôn ngữ

Đồ án sử dụng ngôn ngữ C/C++ để xây dựng giao thức đơn giản bằng Socket theo kiến trúc Server – Client.

b. Môi trường

Đồ án được xây dựng và biên dịch trong môi trường Visual Studio 2017

III. GIAO THỨC TRAO ĐỔI GIỮA SERVER VÀ CLIENT

a. Tổng quát

Bắt đầu một phiên truyền nhận dữ liệu, server sẽ khởi động, đọc dữ liệu từ file và lưu vào một struct "Vé xe", sau đó mở kết nối và lắng nghe kết nối từ các client. Khi client kết nối đến server, server gửi toàn bộ dữ liệu về "Vé xe" đến cho client. Client nhận các "vé xe" và lưu vào một mảng, sau đó hiển thị ra màn hình cho người dùng xem. Client nhận các yêu cầu truy vấn từ người dùng và gửi đến server. Server nhận truy vấn đó đồng thời "lock" các dữ liệu trong struct "Vé xe" ngăn các client khác cùng truy cập và thay đổi dữ liệu. Sau đó tiến hành kiểm tra và trả về kết quả là một "mã lỗi" – dùng để báo tình trạng "vé có tồn tại hay không?", "vé có còn đủ cho người dùng mua hay không?",… Server gửi mã lỗi này cho client, client nhận "mã lỗi" và hiển thị cho người dùng các thông tin cần thiết như: tổng giá vé, số vé còn lại. Client sẽ hỏi người dùng có muốn tiếp tục mua vé hay không, nếu có thì gửi đến server thông báo tiếp tục truy vấn và lặp lại việc truy vấn, nhận kết quả. Nếu không thì sẽ

kết thúc việc mua vé, gửi thông báo kết thúc truy vấn đến server và ngắt kết. Server nhận khi được thông báo kết thúc truy vấn, server tiến hành "unlock" dữ liệu, để các client ở tiểu trình khác có thể truy cập và thay đổi dữ liệu.

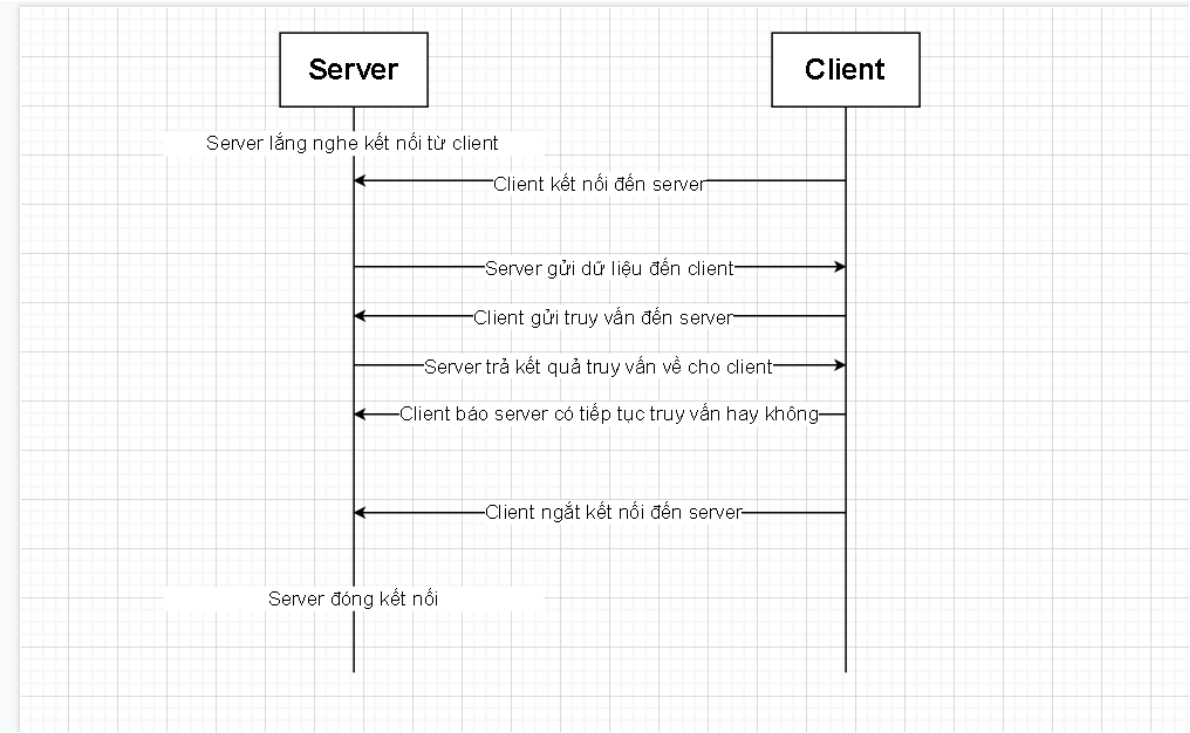

Figure 1: Sơ đồ tổng quát việc giao tiếp giữa Server và Client

b. Sơ đồ hoạt động của Server

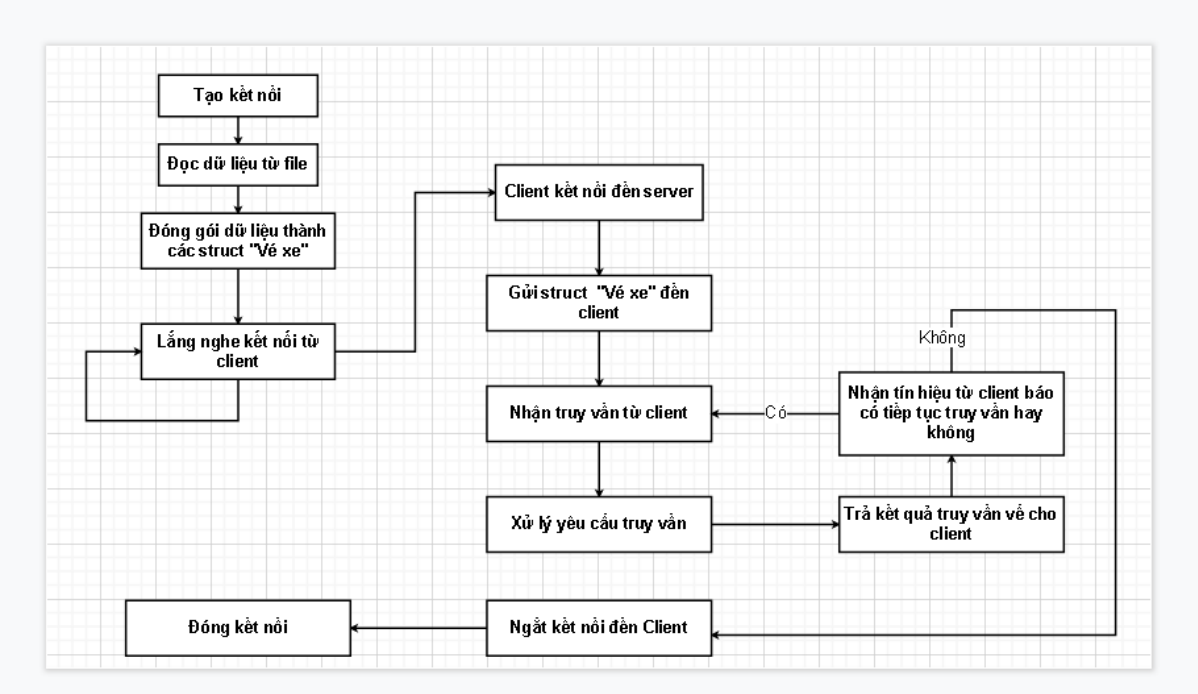

Figure 2: Sơ đồ hoạt động của server

Server khi khởi động sẽ tiến hành đọc dữ liệu từ file và lưu vào một mảng struct "VeXe". Mảng struct "VeXe" sẽ lưu toàn cục trong server để tiện cho việc xử lý và truy vấn dữ liệu

trong mảng này. Ở đây một biến báo trạng thái “dữ liệu có được phép truy cập vào hay không” tên là “mutexLock” sẽ được khởi tạo.

```
VEXE *HangVeXe;
int so_luong = 0;
bool mutexLock = false;
```

Đồng thời dựng các struct để lưu trữ các vé xe, các truy vấn đến từ client và phản hồi của server

```
//Definition struct Ticket
struct VEXE {
    char ID[255];
    char Ten[255];
    char LoaiVe[255];
    int SoLuong;
    int GiaTien;
};
//Definition struct Request Ticket
struct RequestTicket {
    char ID[255];
    char Ten[255];
    char LoaiVe[255];
    int SoLuong;
};
struct ResponseRequest {
    int errCode;
    int slVeConLai;
    int giaTien;
};
```

```
//Nap du lieu vao struct
int count = 0;
std::fstream fileInput("VeXe.txt", std::ios::in);
// kIỂM TRA XEM FILE CÓ BỊ LỖI
if (fileInput.fail()) {
    std::cout << "Failed to open this file!" << std::endl;
}
// lấy số lượng vé để khởi tao struct Hang ve xe
char temp1[255];
fileInput.getline(temp1, 255);
String_convert_int(temp1, so_luong);
HangVeXe = new VEXE[so_luong];
//Khởi tạo mảng
//Đọc dữ liệu vô struct mặt hàng
while (!fileInput.eof()) {
    char temp[255];
    fileInput.getline(temp, 255);
    strcpy(HangVeXe[count].ID, temp);
    fileInput.getline(temp, 255);
    strcpy(HangVeXe[count].Ten, temp);
    fileInput.getline(temp, 255);
    strcpy(HangVeXe[count].LoaiVe, temp);
    fileInput.getline(temp, 255);
    int xx;
    String_convert_int(temp, xx);
    HangVeXe[count].SoLuong = xx;
    fileInput.getline(temp, 255);
    String_convert_int(temp, xx);
    HangVeXe[count].GiaTien = xx;
    //tu tang so luong ve ve xe trong hang doi
    count++;
}
std::cout << std::endl;
fileInput.close(); // ĐÓNG FILE NẠP
```

Sau đó, server tiến hành tạo port và lắng nghe các client kết nối đến server

```
server.Create(4567);
do {
    printf("Server lang nghe ket noi tu client\n");
    server.Listen();
```

Khi có một client kết nối đến, server chấp nhận kết nối và tạo con trỏ Socket. Do CSocket không sử dụng được trong thread, nên chúng ta phải chuyển về SOCKET chuẩn. Bằng cách dùng hàm Detach chuyển thành SOCKET.

```
server.Accept(s);
//Khoi tao con tro Socket
SOCKET* hConnected = new SOCKET();
//Chuyển đổi CSocket thanh Socket
*hConnected = s.Detach();
//Khoi tao thread tuong ung voi moi client Connect vao server.
//Nhu vay moi client se doc lap nhau, khong phai cho doi tung client xu ly rieng
threadStatus = CreateThread(NULL, 0, function_cal, hConnected, 0, &threadID);
```

Lúc này server sẽ tạo ra mỗi tiểu trình riêng biệt để xử lý cho từng client.

Trong mỗi tiểu trình, đầu tiên ta chuyển từ SOCKET chuẩn thành Csocket.

```
SOCKET* hConnected = (SOCKET*)arg;
CSocket mysock;
//Chuyen ve lai CSocket
mysock.Attach(*hConnected);
```

Tiến hành gửi số lượng vé cho client

```
//gửi số lượng vé
mysock.Send(&so_luong, sizeof(so_luong), 0);
```

Tiếp theo gửi mảng “VeXe” cho client

```
//gửi vé
for (int j = 0; j < so_luong; j++) {
    mysock.Send(&HangVeXe[j], sizeof HangVeXe[j], 0);
}
```

Mỗi tiểu trình sẽ kiểm tra xem dữ liệu có được phép truy cập hay không bằng việc kiểm tra biến “mutexLock”, nếu giá trị là “False” ta tiến hành bước tiếp theo, nếu “True” tiểu trình sẽ đợi đến khi “mutexLock” trở về “False” mới tiếp tục

```
while (mutexLock)
{
}
mutexLock = true;
```

Ta khởi tạo các giá trị cần thiết, nhận yêu cầu truy vấn gửi về từ client, xử lý các truy vấn đó và gửi lại client một struct mang thông tin phản hồi về: kết quả truy vấn có thành công hay không, số lượng vé còn lại (khi hết vé) và giá tiền cần thanh toán (nếu còn vé). Sau đó ta nhận tín hiệu từ client báo có tiếp tục quá trình truy vấn hay không

```
do {
    ResponseRequest ReRes;
    RequestTicket ReciveReq;
    //Khởi tạo giá trị
    ReRes.errCode = 0;
    ReRes.slVeConLai = 0;
    ReRes.giaTien = 0;
    //Nhận yêu cầu từ khách hàng
    mysock.Receive((RequestTicket*)&ReciveReq, sizeof RequestTicket, 0);

    for (int j = 0; j < so_luong; j++) {
        if ((strcmp(HangVeXe[j].ID, ReciveReq.ID) == 0) && (strcmp(HangVeXe[j].Ten, ReciveReq.Ten) == 0)) {
            ReRes.errCode += 1;
            if (strcmp(HangVeXe[j].LoaiVe, ReciveReq.LoaiVe) == 0) {
                ReRes.errCode += 2;
                ReRes.slVeConLai = HangVeXe[j].SoLuong;
                if (HangVeXe[j].SoLuong >= ReciveReq.SoLuong) {
                    ReRes.errCode += 4;
                    ReRes.giaTien = ReciveReq.SoLuong * HangVeXe[j].GiaTien;
                    HangVeXe[j].SoLuong -= ReciveReq.SoLuong;
                }
            }
        }
    }
    //Gửi về Client mã lỗi
    cout << "Error code: " << ReRes.errCode << "\n";
    mysock.Send(&ReRes, sizeof ReRes, 0);
    mysock.Receive((int*)&t, sizeof t, 0);
} while (t == 1);
```

Sau khi kết thúc truy vấn, ta chuyển biến “mutexLock” về false để các tiểu trình khác có thể truy cập dữ liệu và tiến hành truy vấn

```
mutexLock = false;
```

Cuối cùng ta ngắt kết nối một tiểu trình, kết thúc quá trình.

```
// ĐÓNG KẾT NỐI
delete hConnected;
```

c. Sơ đồ hoạt động của Client

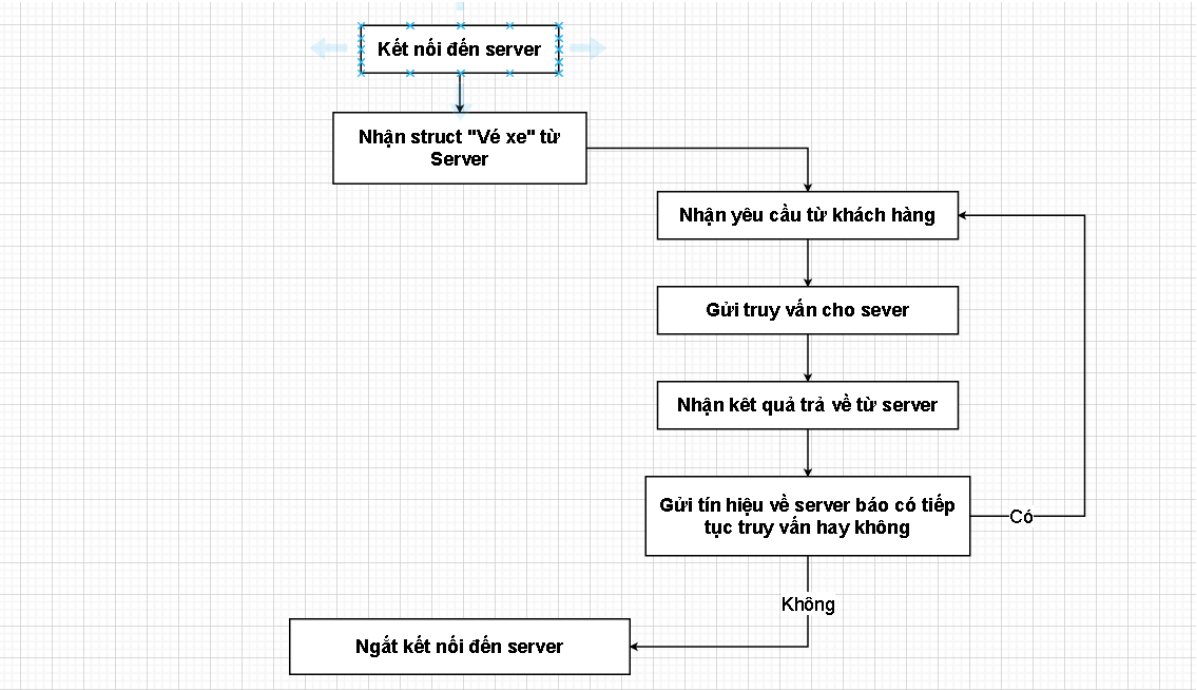

Figure 3: Sơ đồ hoạt động của Client

Trước khi bắt đầu, ta định nghĩa các struct cần thiết để client có thể đọc hiểu được những dữ liệu server gửi qua.

```
//Definition struct Ticket
struct VEXE {
    char ID[255];
    char Ten[255];
    char LoaiVe[255];
    int SoLuong;
    int GiaTien;
};
//Definition struct Request Ticket
struct RequestTicket {
    char ID[255];
    char Ten[255];
    char LoaiVe[255];
    int SoLuong;
};
struct ResponseRequest {
    int errCode;
    int slVeConLai;
    int giaTien;
};
```

Tạo socket Client và tiến hành kết nối đến server

```
AfxSocketInit(NULL);
CSocket client;

client.Create();
client.Connect(_T("127.0.0.1"), 4567);
```

Khai báo các biến và nhận mảng struct “VeXe”

```
//Khai báo các biến
int sovexe = 0;
VEXE *HangVeXe;
//Nhận số lượng vé xe
client.Receive(&sovexe, sizeof(int), 0);
//Khởi tạo mảng vé xe
HangVeXe = new VEXE[sovexe];
for (int j = 0; j < sovexe; j++) {
    client.Receive(&HangVeXe[j], sizeof HangVeXe[j], 0);
}
```

In ra toàn bộ các "VeXe" vừa nhận được

```
//In danh sách vé xe
for (int i = 0; i < sovexe; i++)
{
    cout << "< ID: " << HangVeXe[i].ID << " >\t";
    cout << "< Ten Ve Xe: " << HangVeXe[i].Ten << " >\t";
    cout << "< Loai Ve Xe: " << HangVeXe[i].LoaiVe << " >\t";
    cout << "< So luong ve: " << HangVeXe[i].SoLuong << " >\t";
    cout << "< Gia ve: " << HangVeXe[i].GiaTien << " >\n";
}
```

Tiến hành mời người dùng nhập thông tin tìm kiếm về vé xe. Ở đây việc nhập thông tin và nhận kết quả trả về được đặt trong một vòng lặp do while để người dùng có thể thực hiện thao tác truy vấn nhiều lần

```
cout << "Vui long nhap thong tin ve can tim kiem \n";
char select = 'y';
do {
    RequestTicket Req;
    ResponseRequest ResFromServer;
    string MessageError;
    //Nhập vé cần tìm kiếm
    cout << "Nhap id ve ";
    gets_s(Req.ID);
    String_to_upper(Req.ID);
    cout << "Nhap ten ve ";
    gets_s(Req.Ten);
    String_to_upper(Req.Ten);
    cout << "Nhap loai ve ";
    gets_s(Req.LoaiVe);
    String_to_upper(Req.LoaiVe);
    cout << "Nhap so luong ve ";
    cin >> Req.SoLuong;
```

Gửi truy vấn cho server

```
//Gửi yêu cầu truy vấn cho server
client.Send(&Req, sizeof Req, 0);
```

Nhận kết quả trả về từ Server

```
//Nhận phản hồi từ server
client.Receive((ResponseRequest*)&ResFromServer, sizeof ResFromServer, 0);
```

In ra thông báo cho khách hàng

```
//In thông báo dựa trên errCode gửi về từ server
switch (ResFromServer.errCode)
{
case 0:
{
    MessageError = "Khong co ve can tim\n";
    break;
}
case 1:
{
    MessageError = "Khong co loai ve nay\n";
    break;
}
case 3:
{
    MessageError = "Vuot qua so ve con lai\n";
    break;
}
case 7:
{
    MessageError = "Loai ve nay dang co sang. ";
    break;
}
}
cout << MessageError;
//In ra giá tiền hoặc số lượng vé còn lại
if (ResFromServer.errCode == 7)
{
    cout << "So tien can phai tra la: " << ResFromServer.giaTien;
}
else if (ResFromServer.errCode == 3)
{
    cout << "So ve con lai la: " << ResFromServer.slVeConLai;
}
```

Hỏi khách hàng có tiếp tục tìm kiếm thông tin hay không

```
//Hỏi có tiếp tục mua vé không
cout << "\nQuy khach co muon tiep tuc mua ve khong? (Y/N)";
getchar();
select = getchar();
```

Gửi tín hiệu cho server

```
//Gửi cho server thông báo có tiếp tục hay không
int t = 0;
if (select == 89 || select == 121)
{
    t = 1;
}
client.Send(&t, sizeof t, 0);
```

Sau khi kết thúc truy vấn, client ngắt kết nối với server.

d. Format Packet

Send IP	Send Port	Recive IP	Recive Port
Data			

IV. MỨC ĐỘ HOÀN THÀNH

- Mức độ hoàn thành đồ án: 100%
- Các chức năng làm được:

    - Đọc file
    - Truyền nhận dữ liệu
    - Hiển thị dữ liệu cho người dùng
    - Truy vấn và xử lý các truy vấn
    - Trả kết quả cần thiết cho người dùng
    - Cho phép truy vấn nhiều lần
    - Kết nối 1-n dùng đa tiểu trình
    - Chống "race condition"
- Các chức năng còn chưa làm được
    - Chống "race condition" dùng kỹ thuật đơn giản, khiến các client trả kết quả cho người dùng theo cách tuần tự, client trước đợi client sau, tốn nhiều thời gian
    - Ý tưởng giải quyết: dùng mảng lưu trạng thái "có khóa dữ liệu hay không" cho từng "Vé xe"

TÀI LIỆU THAM KHẢO

1. Mai Văn Cường, *Giáo trình Mạng máy tính*, Nhà xuất bản Khoa học và Kỹ thuật