

**TRƯỜNG ĐẠI HỌC TIỀN GIANG**  
**KHOA KỸ THUẬT CÔNG NGHỆ**



**BÁO CÁO BÀI TẬP THỰC HÀNH**  
**NGUYÊN LÝ HỆ ĐIỀU HÀNH**

**TRIỂN KHAI CHƯƠNG TRÌNH CLIENT – SERVER**  
**ĐA TIỂU TRÌNH DÙNG TCP SOCKET**

**Giảng viên hướng dẫn: ThS. NGUYỄN VĂN NÓI**

**Sinh viên thực hiện: TRẦN THANH TÂN**

**Mã số sinh viên: 021101021**

**Lớp: ĐH CNTT 21A**

*Tiền Giang, tháng 5 năm 2024*

## LỜI CẢM ƠN

Em xin bày tỏ lòng biết ơn sâu sắc đến Thầy Nguyễn Văn Nổi đã tận tình hướng dẫn và đồng hành cùng em trong suốt quá trình học tập môn Nguyên lý hệ điều hành. Thời gian qua, nhờ sự giảng dạy của Thầy, em đã có thể nắm bắt sâu sắc hơn các kiến thức về lập trình đa tiêu trình, giao tiếp giữa các tiến trình và đồng bộ các tiến trình, điều này đã giúp em từng bước thực hiện tốt bài tập thực hành này và tích lũy được nhiều kinh nghiệm quý báu.

Dù đã nỗ lực hết mình, chương trình triển khai của em vẫn còn những thiếu sót. Em rất mong nhận được sự đóng góp ý kiến và hướng dẫn thêm từ Thầy để chương trình có thể được hoàn thiện hơn.

Em xin chân thành cảm ơn Thầy!

*Tiền Giang*, ngày 2 tháng 5 năm 2024

Sinh viên thực hiện

Trần Thanh Tân

# MỤC LỤC

<b>LỜI CẢM ƠN.....</b>	<b>2</b>
<b>DANH MỤC HÌNH ẢNH .....</b>	<b>4</b>
<b>CHƯƠNG 1: GIỚI THIỆU.....</b>	<b>5</b>
<b>1. 1. Giới thiệu chương trình .....</b>	<b>5</b>
<b>1.2. Mục tiêu của chương trình .....</b>	<b>5</b>
<b>CHƯƠNG 2: PHÂN TÍCH VÀ XÂY DỰNG CHƯƠNG TRÌNH.....</b>	<b>6</b>
<b>2.1. Phân tích.....</b>	<b>6</b>
2.1.1. Cơ sở lý thuyết.....	6
2.1.2. Công nghệ sử dụng.....	7
<b>2.2. Xây dựng chương trình.....</b>	<b>10</b>
2.2.1. Xây dựng project: .....	10
2.2.2. Phân tích từng thành phần .....	11
2.2.3. Phân tích quy trình hoạt động của chương trình .....	22
<b>CHƯƠNG 3: KẾT QUẢ THỰC HIỆN .....</b>	<b>24</b>
<b>3.1. Kết quả thực hiện triển khai chương trình .....</b>	<b>24</b>
<b>3.2. Ghi kết quả vào tập tin.....</b>	<b>25</b>
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>26</b>

## DANH MỤC HÌNH ẢNH

Hình 1. Cấu trúc chương trình.....	10
Hình 2. Kết quả triển khai chương trình.....	24
Hình 3. Kết quả lưu vào tập tin .....	25

# CHƯƠNG 1: GIỚI THIỆU

## 1.1. Giới thiệu chương trình

Chương trình được thiết kế nhằm mục đích triển khai một hệ thống Client-Server sử dụng TCP socket để xử lý và giải quyết một bài toán toán xác định chiều cao  $x$  để cắt hình chữ nhật thành một chiếc hộp không nắp có thể tích lớn nhất  $V_{max}$  bằng việc sử dụng ngôn ngữ lập trình C++ và phát triển trong môi trường Visual Studio 2015, ứng dụng của lý thuyết lập trình đa tiến trình, giao tiếp giữa các tiến trình, và đồng bộ hóa tiến trình một cách hiệu quả.

## 1.2. Mục tiêu của chương trình

Phát triển hai phần chính là Client và Server:

### - Ở Client:

- + Đọc dữ liệu: Client bắt đầu bằng việc đọc danh sách các hình chữ nhật từ tập tin DanhHCN\_31012024.txt, mỗi hình chữ nhật được xác định bởi chiều dài (Length) và chiều rộng (Width).

- + Gửi yêu cầu: Sau đó, Client sẽ lần lượt gửi thông tin về chiều dài và chiều rộng của từng hình chữ nhật tới Server qua TCP socket.

- + Nhận kết quả: Client nhận kết quả tính toán chiều cao  $x$  và thể tích  $V_{max}$  từ Server và cập nhật những thông tin này vào danh sách ban đầu.

- + Lưu trữ kết quả: Cuối cùng, client sẽ lưu kết quả được cập nhật vào tập tin ketqua.csv.

### - Ở Server:

- + Nhận dữ liệu: Server lắng nghe và nhận dữ liệu từ các Client qua TCP socket.

- + Xử lý tính toán: Dựa trên chiều dài và chiều rộng nhận được, Server sẽ tính toán chiều cao  $x$  sao cho hình hộp tạo thành có thể tích lớn nhất.

- + Gửi phản hồi: Server sau đó gửi lại kết quả  $x$  và  $V_{max}$  cho Client.

## CHƯƠNG 2: PHÂN TÍCH VÀ XÂY DỰNG CHƯƠNG TRÌNH

### 2.1. Phân tích

#### 2.1.1. Cơ sở lý thuyết

##### 2.1.1.1. Lập trình đa tiến trình

Lập trình đa tiến trình là một kỹ thuật quan trọng trong việc tối ưu hóa hiệu suất và tận dụng tài nguyên của hệ thống. Bằng cách sử dụng đa tiến trình, chương trình có thể thực hiện đồng thời nhiều tác vụ khác nhau mà không ảnh hưởng đến hiệu suất hoạt động. Trong chương trình này, việc sử dụng đa tiến trình giúp cho việc xử lý các yêu cầu từ Client và tính toán trên Server được thực hiện một cách song song, tăng cường hiệu suất và đồng thời giảm thiểu thời gian chờ đợi.

##### 2.1.1.2. Giao tiếp giữa các tiến trình

Để giao tiếp giữa Client và Server, chương trình sử dụng giao thức TCP Socket. TCP (Transmission Control Protocol) là một giao thức truyền dữ liệu tin cậy và đảm bảo, được sử dụng rộng rãi trong việc truyền dữ liệu qua mạng. Việc sử dụng TCP Socket giúp đảm bảo tính tin cậy và hiệu quả trong việc truyền dữ liệu giữa Client và Server.

##### 2.1.1.3. Đồng bộ tiến trình bằng Semaphore

Semaphore được sử dụng để đồng bộ hóa việc truy cập vào các biến dùng chung, giúp đảm bảo tính nhất quán và an toàn trong quá trình truy cập và thay đổi dữ liệu. Sử dụng Semaphore giúp tránh được các vấn đề về đọc/ghi đồng thời vào dữ liệu, từ đó tăng cường tính ổn định và đáng tin cậy của chương trình.

##### 2.1.1.4. Giải thuật

Thuật toán tìm giá trị cực đại dựa trên phương pháp đạo hàm số và điều chỉnh giá trị  $x$  dựa trên đạo hàm tại  $x$  để tiến gần tới điểm cực đại của hàm số. Điều kiện dừng của vòng lặp là số lần lặp đạt tối đa  $N_{MAX}$ . ALPHA trong thuật toán này được dùng như một hằng số hội tụ để điều chỉnh tốc độ tiến của giải thuật. Được mô tả như sau:

- Tìm Max của  $f(x) = x * (W - 2x) * (L - 2x)$  với  $0 \leq x \leq W/2$

- Ý nghĩa của đạo hàm tại điểm  $x_0$ : nếu  $f'(x_0) > 0 \rightarrow$  đồng biến  $\rightarrow$  tăng  $x_0 \rightarrow$  tăng  $y_0$ . Ngược lại: nếu  $f'(x_0) < 0 \rightarrow$  nghịch biến  $\rightarrow$  giảm  $x_0 \rightarrow$  tăng  $y_0$

- Tính  $f'(x_0)$

$$f(x_0 + \Delta t) = f(x_0) + f'(x_0) \cdot \Delta t$$

$$f(x_0 - \Delta t) = f(x_0) - f'(x_0) \Delta t$$

$$\Rightarrow f'(x_0) = [f(x_0 + \Delta t) - f(x_0 - \Delta t)] / (2\Delta t)$$

- Giải thuật:

*$n = 0$ ; // số lần lặp*

*$x = 0$ ; // giá trị khởi tạo  $x_0$*

*while ( $n \leq NMAX$ ) // số lần lặp tối đa  $NMAX$*

*{*

*$df = f'(x)$ ;*

*if ( $df > 0$ )*

*$x = x + ALPHA * df$ ; //  $ALPHA$  hằng số hội tụ*

*else*

*$x = x - ALPHA * df$ ;*

*$y = f(x)$ ;*

*$n++$ ;*

*}*

*Xuất ( $x, y$ )*

## **2.1.2. Công nghệ sử dụng**

### **2.1.2.1. Phần mềm**

*Visual Studio 2015*: là một môi trường phát triển tích hợp (IDE) cho việc lập trình C++. Nó cung cấp các công cụ hỗ trợ mạnh mẽ cho việc phát triển ứng dụng, từ việc viết mã nguồn, biên dịch đến debug và triển khai.

### 2.1.2.2. Thư viện sử dụng

#### - *stdio.h* (Standard Input Output):

+ Chức năng: *stdio.h* cung cấp các hàm và định nghĩa liên quan đến nhập và xuất dữ liệu từ và ra các luồng chuẩn (standard streams) như *stdin* (đầu vào tiêu chuẩn), *stdout* (đầu ra tiêu chuẩn), và *stderr* (lỗi tiêu chuẩn).

+ Trong chương trình này, *stdio.h* được sử dụng để thực hiện đọc và ghi dữ liệu từ và vào các tập tin. Cụ thể, các hàm như *fopen\_s*, *fgets*, và *fprintf* được sử dụng để mở, đọc và ghi dữ liệu vào các tập tin. Hàm *printf()* được dùng để in kết quả ra màn hình.

#### - *winsock2.h*:

+ Chức năng: *winsock2.h* là một phần của Windows Sockets API, cung cấp các hàm và cấu trúc dữ liệu để thực hiện giao tiếp mạng trong hệ điều hành Windows, sử dụng giao thức TCP/IP.

+ Trong chương trình này, *winsock2.h* được sử dụng để tạo và quản lý các socket mạng.

- Các hàm như *connect()*, *bind()*, *listen()*, *accept()*, *send()*, và *recv()* được sử dụng để tạo, kết nối và truyền dữ liệu qua các kết nối socket giữa Client và Server.

- *SOCKET* được sử dụng để khai báo và quản lý các kết nối mạng.

- *WSADATA* được sử dụng khi khởi tạo Winsock bằng hàm *WSAStartup()*.

- *closesocket()* được sử dụng để giải phóng tài nguyên socket sau khi kết thúc giao tiếp.

- *WSACleanup()* được sử dụng để giải phóng các tài nguyên mạng và kết thúc sử dụng Winsock sau khi ứng dụng hoàn thành công việc của mình.

#### - *windows.h*:

+ Chức năng: *windows.h* cung cấp các hàm và cấu trúc dữ liệu liên quan đến hệ điều hành Windows, bao gồm quản lý tiến trình, tiểu trình, tập tin, bộ nhớ, và giao diện người dùng.

+ Trong chương trình này, *windows.h* được sử dụng để gọi các hàm hệ thống và tạo tiểu trình (thread) thông qua hàm *CreateThread()*.



- *HANDLE* được sử dụng để khai báo và quản lý các tài nguyên hệ thống. Cụ thể, các hàm như *CreateThread()*, *CreateSemaphore()* trả về một *HANDLE* để đại diện cho tài nguyên đã tạo.

- *LPVOID* được sử dụng khi truyền con trỏ tới dữ liệu của kiểu không xác định. Nó được sử dụng trong hàm *CreateThread()* để truyền tham số cho tiểu trình.

- *DWORD* được sử dụng để đại diện cho các thông số hoặc kích thước dữ liệu.

- *WINAPI* được sử dụng để định nghĩa kiểu của các hàm API. nó được sử dụng trong khai báo hàm cụ thể trong chương trình là *CreateThread()*, *FindMaxVmax()* và *FindVmax()*.

- *ReleaseSemaphore()* được sử dụng để giải phóng một Semaphore sau khi nó đã được sử dụng để đồng bộ hóa các tiến trình.

- *WaitForSingleObject()* được sử dụng để đợi cho đến khi một Semaphore được giải phóng hoặc một tiểu trình kết thúc.

- *CloseHandle()* được sử dụng để giải phóng tài nguyên hệ thống sau khi chúng không còn cần thiết nữa.

- *math.h*:

- + Chức năng: *math.h* cung cấp các hàm toán học và hằng số được sử dụng trong tính toán.

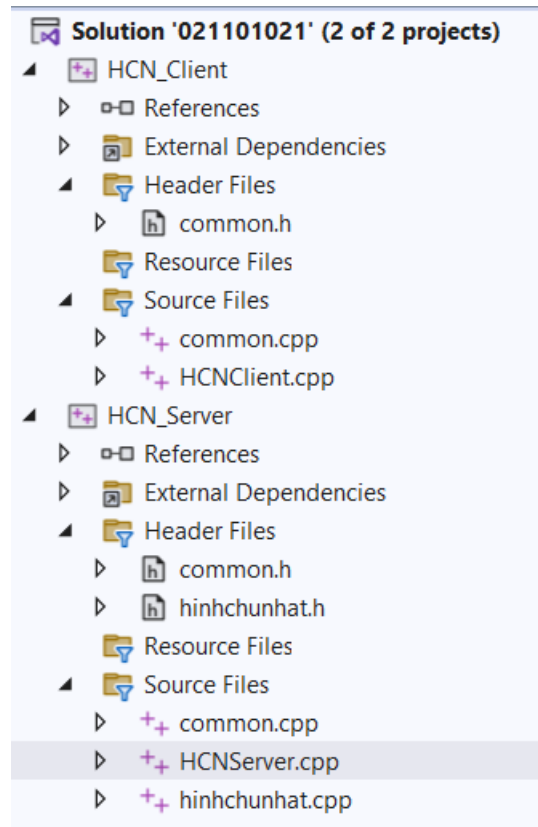
- + Trong chương trình này, *math.h* được sử dụng để thực hiện các phép toán toán học trong tính toán. Cụ thể, hàm *fabs()* được sử dụng để tính giá trị tuyệt đối.

## 2.2. Xây dựng chương trình

### 2.2.1. Xây dựng project:

#### 2.2.1.1. Tạo project:

- Chương trình được tạo trong môi trường lập trình Visual Studio.
- Bao gồm hai project: HCN\_Client và HCN\_Server.



Hình 1. Cấu trúc chương trình

#### 2.2.1.2. Cấu trúc project:

- Mỗi project bao gồm các file mã nguồn và các tệp header liên quan.
- Trong project HCN\_Client:
  - + Các file mã nguồn: HCNClient.cpp, common.cpp.

- + Các tệp header: common.h.
- Trong project HCN\_Server:
  - + Các file mã nguồn: HCNServer.cpp, hinhchunhat.cpp, common.cpp.
  - + Các tệp header: hinhchunhat.h, common.h.

## 2.2.2. Phân tích từng thành phần

### 2.2.2.1. Project HCN\_Client:

- *common.h*:

+ Tệp này chứa các định nghĩa cấu trúc và khai báo hàm chung được sử dụng trong project HCN\_Client:

+ Trong đó:

- Cấu trúc *request\_t* đại diện cho yêu cầu gửi thông tin về chiều dài và chiều rộng của hình chữ nhật từ Client tới Server.

```
typedef struct _request_t
{
    double width;
    double length;

} request_t, * prequest_t;
```

- Cấu trúc *reply\_t* chứa thông tin về độ cao cắt tối đa x và thể tích tương ứng Vmax của hình chữ nhật, được Server gửi lại cho Client.

```
typedef struct _reply_t
{
    double x;
    double Vmax;

} reply_t, * preply_t;
```

- Cấu trúc *HCN\_Max* chứa thông tin về hình chữ nhật có Vmax lớn nhất.

```
typedef struct _HCN_Max
{
    request_t req;
    reply_t rep;

} HCN_Max, * PHCN_Max;
```

- Hai hàm *InitializeWinsock()* và *CreateConnect()* dùng để khởi tạo và tạo kết nối tới Server thông qua giao thức TCP.

```
int InitializeWinsock();
```

```
SOCKET CreateConnect(char* SVRIPAddress, u_short port);
```

- *common.cpp*:

+ Tập này cài đặt các hàm đã được khai báo trong *common.h* ở project HCN\_Client.

+ Trong đó:

- Hàm *InitializeWinsock()* dùng để khởi tạo Winsock để sử dụng thư viện socket trên Windows.

```
int InitializeWinsock()
{
    WSADATA wsa;
    printf("\nInitialising Winsock...\n");
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
        printf("Failed. Error Code: %d", WSAGetLastError());
        return 0;
    }
    printf("Initialised.\n");
    return 1;
}
```

- Hàm *CreateConnect()* tạo kết nối tới server dựa trên địa chỉ IP và cổng được cung cấp.

```
SOCKET CreateConnect(char* SVRIPAddress, u_short port)
{
    SOCKET s;
    struct sockaddr_in server;
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
        printf("Could not create socket : %d", WSAGetLastError());
        exit(-3);
    }
    printf("Socket created.\n");
    server.sin_addr.s_addr = inet_addr(SVRIPAddress);
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    if (connect(s, (struct sockaddr*)&server, sizeof(server)) < 0)
    {
        printf("Connect %s error !", SVRIPAddress);
        exit(-4);
    }
    printf("Connected to Server %s", SVRIPAddress);
}
```

```

        return s;
    }

```

- HCNClient.cpp:

+ Tập này thực hiện các chức năng chính của Client.

+ Trong đó:

- Các biến toàn cục *max\_x*, *max\_Vmax*, *max\_width*, *max\_length* được khai báo để lưu trữ thông tin về hình chữ nhật có Vmax lớn nhất.

```

volatile double max_x = 0;

volatile double max_Vmax = 0;

volatile double max_width = 0;

volatile double max_length = 0;

```

- Hàm *FindMaxVmax(LPVOID lpParam)*: là hàm đa tiêu trình dùng để tìm và cập nhật thông tin về hình chữ nhật có Vmax lớn nhất. Nó nhận thông tin từ mỗi hình chữ nhật đã xử lý và so sánh với thông tin của hình chữ nhật có Vmax lớn nhất hiện tại.

```

DWORD WINAPI FindMaxVmax(LPVOID lpParam)
{
    HCN_Max* data = (HCN_Max*)lpParam;
    double x = data->rep.x;
    double Vmax = data->rep.Vmax;
    double width = data->req.width;
    double length = data->req.length;

    WaitForSingleObject(hSem[0], INFINITE);

    if (Vmax > max_Vmax)
    {
        max_x = x;
        max_Vmax = Vmax;
        max_width = width;
        max_length = length;
    }

    ReleaseSemaphore(hSem[0], 1, NULL);

    delete data;
    return 0;
}

```

- Hàm *ClientThread(LPVOID lpParam)*: thực hiện công việc chính của Client, bao gồm đọc danh sách hình chữ nhật từ tệp *DanhHCN\_31012024.txt*, gửi yêu cầu tới Server và xử lý kết quả. Đồng thời, nó cũng tạo và quản lý các luồng con để xử lý đồng thời các yêu cầu và kết quả từ Server. Sau cùng lưu kết quả vào tệp *ketqua.csv* và hiển thị thông tin về hình chữ nhật có Vmax lớn nhất.

```
DWORD WINAPI ClientThread(LPVOID lpParam)
{
    SOCKET s = *((SOCKET*)lpParam);
    FILE* inputFile = NULL;
    FILE* outputFile = NULL;
    char line[100];
    double WIDTH, LENGTH, x, Vmax;
    errno_t err;

    err = fopen_s(&inputFile, "DanhHCN_31012024.txt", "r");
    if (err != 0 || inputFile == NULL)
    {
        printf("Cannot open file DanhHCN_31012024.txt!\n");
        return 1;
    }

    fgets(line, sizeof(line), inputFile);

    err = fopen_s(&outputFile, "ketqua.csv", "w");
    if (err != 0 || outputFile == NULL)
    {
        printf("Unable to create file ketqua.csv\n");
        fclose(inputFile);
        return 1;
    }

    fprintf(outputFile, "width,length,x,Vmax\n");

    while (fgets(line, sizeof(line), inputFile))
    {
        sscanf_s(line, "%lf\t%lf", &WIDTH, &LENGTH);
        request_t req = { WIDTH, LENGTH };
        reply_t rep;

        printf("Width: %3.0lf, Length: %3.0lf\n", req.width,
            req.length);

        WaitForSingleObject(hSem[0], INFINITE);

        if (send(s, (char*)&req, sizeof(request_t), 0) !=
            sizeof(request_t))
        {
            printf("Cannot send \n");
        }
        else
            printf("Send ... \n");
    }
}
```

```

        if (recv(s, (char*)&rep, sizeof(reply_t), 0) !=
sizeof(reply_t))
            printf("Cannot receive \n");
        else
            printf("Receive ... \n");

        printf("Result: x = %3.5lf, Vmax = %3.5lf\n", rep.x,
rep.Vmax);

printf(".....
.\n");
        fprintf(outputFile, "%lf,%lf,%lf,%lf\n", req.width,
req.length, rep.x, rep.Vmax);

        ReleaseSemaphore(hSem[0], 1, NULL);

        HCN_Max* Data = new HCN_Max;
        Data->req = req;
        Data->rep = rep;

        HANDLE hThread = CreateThread(NULL, 0, FindMaxVmax, Data,
0, NULL);
        if (hThread)
        {
            WaitForSingleObject(hThread, INFINITE);
            CloseHandle(hThread);
        }

        fclose(inputFile);
        fclose(outputFile);

        printf("Recorded results to file ketqua.csv
successfully!\n\n");

        printf("Information on the rectangle with the largest maximum
volume:\n");
        printf("Width: %3.0lf Length: %3.0lf, x: %3.5fl, Vmax:
%3.5lf\n", max_width, max_length, max_x, max_Vmax);

printf(".....
.\n\n");
        return 0;
    }
}

```

- Hàm *main()*: Chương trình bắt đầu bằng việc khởi tạo môi trường mạng và tạo kết nối tới server thông qua hàm *InitializeWinsock()* và *CreateConnect(IPServer, port)*. Sau đó Sau đó, một luồng chính được tạo để thực hiện công việc của client bằng cách gọi hàm *CreateThread(NULL, 0, ClientThread, &s, 0, NULL)* cho phép chương trình gửi yêu cầu đến Server và xử lý kết quả mà không làm gián đoạn chương trình chính. Khi công việc đã

hoàn thành, kết nối tới Server được đóng và các tài nguyên được dọn dẹp. Chương trình dừng lại để người dùng có thể xem kết quả trước khi kết thúc.

```
int main(int argc, char* argv[])
{
    char* IPServer = (char*)"127.0.0.1";
    u_short port = 54321;
    printf("%s IPServer Port\n", argv[0]);
    if (argc > 2)
    {
        port = atoi(argv[2]);
        IPServer = argv[1];
    }
    printf("IPServer: %s. Port %ld\n", IPServer, (int)port);

    hSem[0] = CreateSemaphore(NULL, 1, 1, NULL);

    InitializeWinsock();
    SOCKET s = CreateConnect(IPServer, port);

    HANDLE hClientThread = CreateThread(NULL, 0, ClientThread, &s,
0, NULL);
    if (hClientThread != NULL)
    {
        WaitForSingleObject(hClientThread, INFINITE);
    }

    CloseHandle(hSem[0]);
    closesocket(s);
    WSACleanup();
    system("pause");
    return 0;
}
```

#### 2.2.2.2. Project HCN Server:

- *common.h*:

+ Tập này chứa các định nghĩa cấu trúc và khai báo hàm chung được sử dụng trong HCN\_Server:

+ Trong đó:

- Cấu trúc *request\_t* đại diện cho yêu cầu gửi thông tin về chiều dài và chiều rộng của hình chữ nhật từ Client tới Server.

```
typedef struct _request_t
{
    double width;
    double length;
```



```
    } request_t, * prequest_t;
```

- Cấu trúc *reply\_t* chứa thông tin về độ cao cắt tối đa  $x$  và thể tích tương ứng  $V_{max}$  của hình chữ nhật, được Server gửi lại cho Client.

```
typedef struct _reply_t
{
    double x;
    double Vmax;

} reply_t, * preply_t;
```

- *common.cpp*:

+ Tập này cài đặt các hàm đã được khai báo trong *common.h* ở project HCN\_Server.

+ Trong đó:

- Hàm *InitializeWinsock()* dùng để khởi tạo Winsock để sử dụng thư viện socket trên Windows.

```
int InitializeWinsock()
{
    WSADATA wsa;
    printf("\nInitialising Winsock...\n");
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
        printf("Failed. Error Code: %d", WSAGetLastError());
        return 0;
    }
    printf("Initialised.\n");
    return 1;
}
```

- Hàm *CreateBindListen()* tạo và lắng nghe kết nối tới Server trên một cổng cụ thể.

```
SOCKET CreateBindListen(u_short port) {
    SOCKET s;
    struct sockaddr_in server;
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
        printf("Could not create socket: %d", WSAGetLastError());
        exit(-1);
    }
    printf("Socket created.\n");
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(port);
    if (bind(s, (struct sockaddr*)&server, sizeof(server)) ==
        SOCKET_ERROR) {
```

```

        printf("Bind failed with error code : %d",
WSAGetLastError());
        exit(-2);
    }
    puts("Bind done");
    listen(s, 30);
    puts("Waiting for incoming connections...");
    return s;
}

```

- *hinhchunhat.h*:

+ Tập chứa các định nghĩa cấu trúc dữ liệu và khai báo hàm liên quan đến tính toán độ cao cắt tối đa  $x$  và thể tích tương ứng  $V_{\max}$  của hình chữ nhật.

+ Bao gồm khai báo các hàm  $f()$ ,  $df()$ ,  $timx()$

```

#pragma once

double f(double x, double WIDTH, double LENGTH);

double df(double x, double WIDTH, double LENGTH);

double timx(double WIDTH, double LENGTH);

```

- *hinhchunhat.cpp*:

+ Tập chứa cài đặt các hàm tính toán độ cao cắt tối đa  $x$  và thể tích tương ứng  $V_{\max}$  của hình chữ nhật, dựa trên yêu cầu cụ thể của chương trình.

+ Bao gồm các hàm  $f()$ ,  $df()$ ,  $timx$  đã được khai báo trong *hinhchunhat.h*.

```

const double EPSILON = 1e-6;
double deltat = 0.0001;

double f(double x, double width, double length) {
    return x * (width - 2 * x) * (length - 2 * x);
}

double df(double x, double width, double length) {
    return (f(x + deltat, width, length) - f(x - deltat, width,
length)) / (2 * deltat);
}

double timx(double width, double length) {
    double x = 0, y = f(x, width, length), dy, y0;
    double ALPHA = 0.00001;
    int NMAX = 100000;
    int n = 0;

    while (n < NMAX) {

```

```

    y0 = y;
    dy = df(x, width, length);

    if (dy >= 0)
        x = x + ALPHA * dy;
    else
        x = x - ALPHA * dy;

    if (x >= width / 2 || x >= length / 2) {
        return x;
        break;
    }

    y = f(x, width, length);
    n++;
    if (fabs(y - y0) < EPSILON)
        break;
}
return x;
}

```

- *HCNServer.cpp*:

+ Tập này thực hiện các chức năng chính của Server.

+ Hàm *FindVmax(LPVOID lpParam)*:

- Hàm này được sử dụng để xử lý mỗi yêu cầu từ một client. Nó được thiết kế để chạy trong một luồng riêng biệt cho mỗi kết nối client. Các tham số đầu vào của nó là một con trỏ tới dữ liệu của yêu cầu được truyền qua tham số *lpParam*.

```

DWORD WINAPI FindVmax(LPVOID lpParam)
{
    ...
}

```

- Đầu tiên, dữ liệu yêu cầu từ client được giải nén từ con trỏ *lpParam*. Sau đó, các biến cần thiết được khởi tạo, bao gồm *x* và *Vmax*.

- Tiếp theo, hàm sẽ tính toán độ cao cắt tối đa *x* và thể tích tương ứng *Vmax* của hình chữ nhật dựa trên thông tin yêu cầu nhận được từ client. Các thuật toán và công thức tính toán có thể được triển khai ở đây.

- Sau khi tính toán hoàn thành, kết quả được đóng gói vào một cấu trúc dữ liệu đáp ứng và gửi lại cho client thông qua kết nối socket.

```

DWORD WINAPI FindVmax(LPVOID lpParam)
{
    SOCKET clientSocket = *(SOCKET*)lpParam;
    while (true)
    {
        request_t req;
        reply_t rep;
        int bytesReceived = recv(clientSocket, (char*)&req,
sizeof(request_t), 0);
        if (bytesReceived == sizeof(request_t))
        {
            printf("Receive: Width = %3.0lf, Length = %3.0lf\n",
req.width, req.length);
            double x = timx(req.width, req.length);
            double Vmax = f(x, req.width, req.length);
            rep.x = x;
            rep.Vmax = Vmax;

            printf("Send: x = %3.5lf, Vmax = %3.5lf\n", rep.x,
rep.Vmax);

            printf(".....
.\n");

            send(clientSocket, (char*)&rep, sizeof(reply_t), 0);
        }
        else if (bytesReceived == 0)
        {
            printf("Client disconnected!\n");
            break;
        }
        else
        {
            printf("Receive failed with error: %d\n",
WSAGetLastError());
            break;
        }
    }
    closesocket(clientSocket);
    return 0;
}

```

+ *Hàm AcceptClient(SOCKET s):*

- Đối với mỗi kết nối từ client, một tiểu trình mới sẽ được tạo ra để xử lý yêu cầu từ client đó, giúp cho việc xử lý nhiều yêu cầu cùng một lúc trở nên có hiệu suất hơn. Khi một kết nối được chấp nhận, nó trả về một socket mới đại diện cho kết nối đó.

- Sau khi kết nối được chấp nhận, hàm này sẽ tạo một luồng mới để xử lý yêu cầu từ client đó. Luồng sẽ chạy hàm TimVmax, với dữ liệu yêu cầu từ client được truyền vào.

```

void AcceptClient(SOCKET s)
{
    struct sockaddr_in client;
    int c = sizeof(struct sockaddr_in);

    while (true) {
        SOCKET clientSocket = accept(s, (struct sockaddr*)&client,
&c);
        if (clientSocket == INVALID_SOCKET)
        {
            printf("Accept failed with error: %d\n",
WSAGetLastError());
            continue;
        }

        char* client_ip = inet_ntoa(client.sin_addr);
        int client_port = ntohs(client.sin_port);
        printf("Connection accepted from IP: %s, Port: %d\n",
client_ip, client_port);

        HANDLE hThread = CreateThread(NULL, 0, FindVmax,
&clientSocket, 0, NULL);
        if (hThread != NULL)
            CloseHandle(hThread);
    }
}

```

+ Hàm *main()*:

- Trước tiên, khởi tạo thư viện Winsock bằng cách gọi *InitializeWinsock()*. Điều này cần thiết để sử dụng các hàm liên quan đến socket trên nền tảng Windows.

- Sau đó tạo một socket mới và liên kết nó với một cổng cụ thể trên máy chủ bằng cách gọi *CreateBindListen()*. Điều này cho phép máy chủ lắng nghe các kết nối từ client.

- Chương trình sẽ liên tục chấp nhận các kết nối từ client và tạo một luồng xử lý mới cho mỗi kết nối bằng cách gọi *AcceptClient()*.

```

int main(int argc, char* argv[])
{
    u_short port = 54321;
    if (argc > 1) port = atoi(argv[1]);

    hSemaphore = CreateSemaphore(NULL, 1, 1, NULL);

    InitializeWinsock();
    SOCKET svrsock = CreateBindListen(port);

    AcceptClient(svrsock);
}

```

```

        CloseHandle(hSemaphore);
        closesocket(svrsock);
        WSACleanup();

        return 0;
    }

```

### 2.2.3. Phân tích quy trình hoạt động của chương trình

- Khởi tạo và lắng nghe kết nối:

- + Server sẽ khởi tạo một socket và liên kết nó với một cổng cụ thể.
- + Server sẽ lắng nghe các kết nối đến từ Client.

- Chấp nhận kết nối:

- + Khi một Client kết nối đến, server sẽ chấp nhận kết nối đó.
- + Một luồng thực thi mới sẽ được tạo để xử lý yêu cầu từ client đó.

- Gửi yêu cầu và nhận kết quả:

- + Client sẽ đọc danh sách hình chữ nhật từ tập tin và gửi yêu cầu về Server.
- + Server sẽ nhận yêu cầu từ client, tính toán x và Vmax cho từng hình chữ nhật và gửi kết quả về cho client.

- Xử lý và tính toán:

- + Server sẽ sử dụng thuật toán để tính toán x và Vmax cho từng hình chữ nhật.
- + Thuật toán sử dụng để tính toán x và Vmax được triển khai trong các hàm hỗ trợ như f(), df(), và timx().

- Đồng bộ tiến trình:

- + Semaphore được triển khai trong chương trình để đồng bộ hóa truy cập vào một số tài nguyên chia sẻ giữa các tiểu trình.
- + Đảm bảo rằng chỉ một tiểu trình được phép thực hiện các thao tác trên biến toàn cục lưu trữ thông tin về hình chữ nhật có Vmax lớn nhất.

- Ghi kết quả vào tập tin:

- + Client sẽ nhận kết quả từ server và ghi vào tập tin ketqua.csv.

- + Client cũng cập nhật thông tin lớn nhất của hình chữ nhật và hiển thị thông tin này trên màn hình.

- *Đóng kết nối và dọn dẹp tài nguyên:*

- + Khi hoàn thành, cả Client và Server sẽ đóng kết nối và dọn dẹp tài nguyên đã được sử dụng.

- + Sử dụng hàm *CloseHandle(hSemaphore)* để đóng Semaphore, *closesocket()* để đóng kết nối và *WSACleanup()* để giải phóng tài nguyên mạng.

## CHƯƠNG 3: KẾT QUẢ THỰC HIỆN

### 3.1. Kết quả thực hiện triển khai chương trình

- Chương trình đã được triển khai và thực hiện thành công, đạt được các mục tiêu của chương trình đề ra ban đầu.
- Qua thực nghiệm, phía Client (HCN\_Client) đã đọc lần lượt chiều dài và chiều rộng trong DanhHCN\_31012024.txt gửi cho phía Server (HCN\_Server) thành công.
- Server đã nhận được chiều dài và chiều rộng mà Client gửi qua và tiến hành tính toán để tìm ra độ cao  $x$  để cắt hình chữ nhật thành chiếc hộp không nắp có thể tích lớn nhất  $V_{max}$ . Server cũng đã gửi cả  $x$  và  $V_{max}$  lại cho Client.
- Quá trình thực hiện chương trình cũng cho thấy sự ổn định và hiệu quả của cách triển khai TCP socket trong việc xử lý và truyền tải dữ liệu giữa Client - Server và đồng bộ giữa chúng. Server xử lý và phản hồi các yêu cầu từ Client một cách nhanh chóng và chính xác, đồng thời đảm bảo tính toàn vẹn của dữ liệu.
- Đồng thời, sau khi sử dụng hàm đa tiêu trình tìm hình chữ nhật có  $V_{max}$  lớn nhất. Sau khi thực hiện xong, phía Client cũng đã hiển thị thông tin về hình chữ nhật cần tìm kiếm này (chiều dài, chiều rộng,  $x$  và  $V_{max}$ ) trên màn hình Console.

```
C:\Users\Admin\Desktop\021101021 - Nop ket qua\vb4\Debug\HCN_Client.exe
Receive ...
Result: x = 33.22024, Vmax = 1677721.40675
.....
Width: 446, Length: 538
Send ...
Receive ...
Result: x = 80.93160, Vmax = 8649508.64127
.....
Width: 133, Length: 322
Send ...
Receive ...
Result: x = 29.12266, Vmax = 574208.47890
.....
Width: 669, Length: 801
Send ...
Receive ...
Result: x = 121.02712, Vmax = 28881854.10238
.....
Width: 922, Length: 373
Send ...
Receive ...
Result: x = 81.94823, Vmax = 12990608.28897
.....
Recorded results to file ketqua.csv successfully!
Information on the rectangle with the largest maximum volume:
Width: 999 Length: 998, x: 166.416531, Vmax: 73741198.88657
.....
Press any key to continue . . . .
23
24 if (Vmax > max_Vmax)
% No issues found
ds
rch (Ctrl+E) Search Depth:

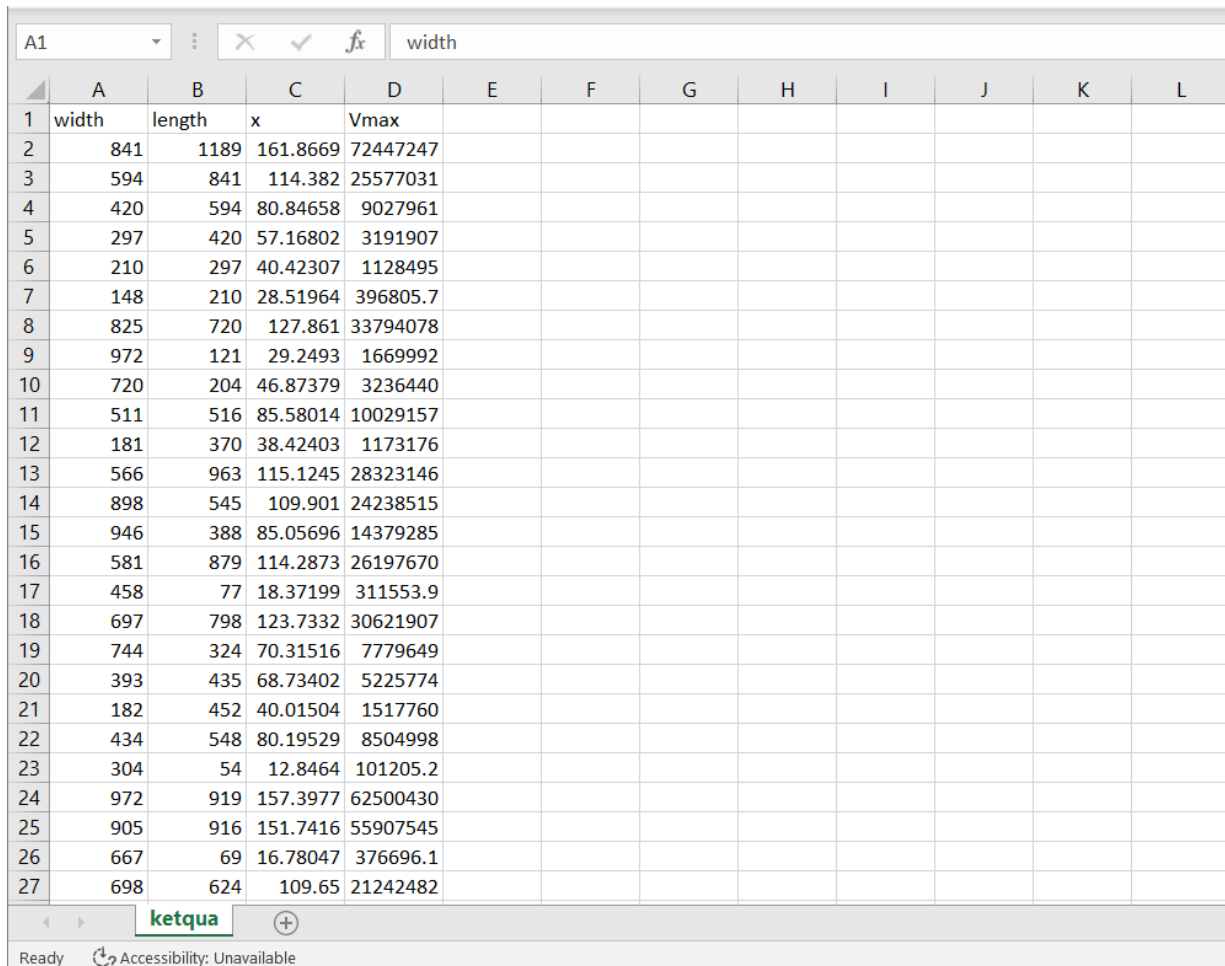
C:\Users\Admin\Desktop\021101021 - Nop ket qua\vb4\Debug\HCN_Server.exe
Receive: Width = 462, Length = 693
Send: x = 90.63845, Vmax = 13020439.48304
.....
Receive: Width = 444, Length = 373
Send: x = 67.31620, Vmax = 4964112.82983
.....
Receive: Width = 23, Length = 826
Send: x = 5.70932, Vmax = 53861.49639
.....
Receive: Width = 329, Length = 909
Send: x = 73.47356, Vmax = 10193275.10656
.....
Receive: Width = 753, Length = 140
Send: x = 33.22024, Vmax = 1677721.40675
.....
Receive: Width = 446, Length = 538
Send: x = 80.93160, Vmax = 8649508.64127
.....
Receive: Width = 133, Length = 322
Send: x = 29.12266, Vmax = 574208.47890
.....
Receive: Width = 669, Length = 801
Send: x = 121.02712, Vmax = 28881854.10238
.....
Receive: Width = 922, Length = 373
Send: x = 81.94823, Vmax = 12990608.28897
.....
Client disconnected!
```

Hình 2. Kết quả triển khai chương trình



### 3.2. Ghi kết quả vào tập tin

Kết quả không chỉ được ghi nhận rõ ràng trên giao diện Console của chương trình mà đồng thời được lưu trữ trong tập tin ketqua.csv, cho phép việc lưu trữ và phân tích dữ liệu một cách dễ dàng. Tập tin ketqua.csv chứa thông tin chi tiết của từng hình chữ nhật sau khi đã tính toán, bao gồm các thông tin về chiều rộng, chiều dài, chiều cao x và thể tích Vmax tương ứng.



	A	B	C	D	E	F	G	H	I	J	K	L
1	width	length	x	Vmax								
2	841	1189	161.8669	72447247								
3	594	841	114.382	25577031								
4	420	594	80.84658	9027961								
5	297	420	57.16802	3191907								
6	210	297	40.42307	1128495								
7	148	210	28.51964	396805.7								
8	825	720	127.861	33794078								
9	972	121	29.2493	1669992								
10	720	204	46.87379	3236440								
11	511	516	85.58014	10029157								
12	181	370	38.42403	1173176								
13	566	963	115.1245	28323146								
14	898	545	109.901	24238515								
15	946	388	85.05696	14379285								
16	581	879	114.2873	26197670								
17	458	77	18.37199	311553.9								
18	697	798	123.7332	30621907								
19	744	324	70.31516	7779649								
20	393	435	68.73402	5225774								
21	182	452	40.01504	1517760								
22	434	548	80.19529	8504998								
23	304	54	12.8464	101205.2								
24	972	919	157.3977	62500430								
25	905	916	151.7416	55907545								
26	667	69	16.78047	376696.1								
27	698	624	109.65	21242482								

Hình 3. Kết quả lưu vào tập tin

## TÀI LIỆU THAM KHẢO

1. Trần Trung Dũng, Phạm Tuấn Sơn (2019). *Hệ điều hành*. Nhà Xuất bản Khoa học và Kỹ thuật.