**Nội dung**

[**BÓC TÁCH HEADER 3**](#_Toc137928446)

[**KIỂM TRA CÚ PHÁP NHẬP TỪ BÀN PHÍM 4**](#_Toc137928447)

[**XỬ LÝ STRING 5**](#_Toc137928448)

[**DOMAIN TO IP 6**](#_Toc137928449)

[**SERVER (Nhận từ client, gửi tới client từ bàn phím) 7**](#_Toc137928450)

[**CLIENT (Gửi lời chào tới server, nhận dữ liệu từ server) 9**](#_Toc137928451)

[**CLIENT DOMAIN 11**](#_Toc137928452)

[**CLIENT NHẬN FILE TỪ SERVER VÀ LƯU DẠNG TÊN MỚI 14**](#_Toc137928453)

[**SERVER GỬI FILE TỚI CLIENT 15**](#_Toc137928454)

[**CLIENT NHẬN 1 MẢNG SỐ THỰC VÀ IN RA MÀN HÌNH 17**](#_Toc137928455)

[**SERVER GỬI MẢNG CHO CLIENT 19**](#_Toc137928456)

[**HTTP SERVER 21**](#_Toc137928457)

[**UDP RECEIVER 23**](#_Toc137928458)

[**UDP SENDER 24**](#_Toc137928459)

[**BLOCKING CLIENT 25**](#_Toc137928460)

[**BLOCKING SERVER 26**](#_Toc137928461)

[**NON-BLOCKING SERVER 28**](#_Toc137928462)

[**SELECT CLIENT 31**](#_Toc137928463)

[**SELECT SERVER 33**](#_Toc137928464)

[**SOICT CLIENT 35**](#_Toc137928465)

[**SOICT SERVER 36**](#_Toc137928466)

[**UDP FILE RECEIVER 38**](#_Toc137928467)

[**UDP FILE RECEIVER SINGLE 40**](#_Toc137928468)

[**UDP FILE SENDER 42**](#_Toc137928469)

[**CHAT SERVER 43**](#_Toc137928470)

[**POLL CLIENT 47**](#_Toc137928471)

[**POLL SERVER 49**](#_Toc137928472)

[**SELECT SERVER V2 51**](#_Toc137928473)

[**CHAT SERVER V2 53**](#_Toc137928474)

[**TELNET SERVER 57**](#_Toc137928475)

[**UDP CHAT 62**](#_Toc137928476)

[**MỘT VÀI HÀM LƯU Ý 64**](#_Toc137928477)

[**Cấu trúc địa chỉ IPv4 64**](#_Toc137928478)

[**Các hàm chuyển đổi địa chỉ 64**](#_Toc137928479)

[**Phân giải tên miền 65**](#_Toc137928480)

[**Hàm socket() 66**](#_Toc137928481)

[**Hàm bind() 66**](#_Toc137928482)

[**Hàm listen() 67**](#_Toc137928483)

[**Hàm accept() 67**](#_Toc137928484)

[**Hàm send() 67**](#_Toc137928485)

[**Hàm write() 68**](#_Toc137928486)

[**Hàm recv() 69**](#_Toc137928487)

[**Hàm read() 69**](#_Toc137928488)

[**Hàm close() 70**](#_Toc137928489)

[**Hàm shutdown() 70**](#_Toc137928490)

[**Hàm connect() 70**](#_Toc137928491)

[**Hàm sendto() (UDP) 70**](#_Toc137928492)

[**Hàm recvfrom() (UDP) 71**](#_Toc137928493)

[**Hàm ioctl() -> bất đồng bộ 71**](#_Toc137928494)

[**Hàm fcntl() -> bất đồng bộ 71**](#_Toc137928495)

[**Hàm select() 71**](#_Toc137928496)

[**Cấu trúc fd\_set 72**](#_Toc137928497)

[**Cấu trúc pollfd 72**](#_Toc137928498)

[**Hàm poll() 72**](#_Toc137928499)

[**Truyền dữ liệu sử dụng TCP - Ứng dụng server 72**](#_Toc137928500)

[**Ứng dụng client 73**](#_Toc137928501)

# BÓC TÁCH HEADER

#include <stdio.h>

int main()

{

    char packet\_bytes[] = {

        0x45, 0x00, 0x02, 0x04, 0x0a, 0x8c, 0x40, 0x00,

        0x80, 0x06, 0x00, 0x00, 0x0a, 0x5a, 0x4c, 0x96,

        0x34, 0x56, 0x44, 0x2e

    };

    int version, ihl, total\_length;

    // Phân tích phiên bản IP từ 4 bit đầu tiên của byte đầu tiên

    version = packet\_bytes[0] >> 4;

    printf("version = %d\n", version);

    // Phân tích độ dài tiêu đề IP từ 4 bit cuối cùng của byte đầu tiên

    // Nhân với 4 để tính độ dài thực tế trong byte

    ihl = (packet\_bytes[0] & 0xf) \* 4;

    printf("ihl = %d bytes\n", ihl);

    // Phân tích tổng độ dài gói tin IP từ byte thứ 3 và 4

    total\_length = packet\_bytes[2] \* 256 + packet\_bytes[3];

    printf("total\_length = %d bytes\n", total\_length);

    // In địa chỉ nguồn IP từ byte thứ 13 đến 16

    printf("IP src: %u.%u.%u.%u\n",

        (unsigned char)packet\_bytes[12],

        (unsigned char)packet\_bytes[13],

        (unsigned char)packet\_bytes[14],

        (unsigned char)packet\_bytes[15]);

    // In địa chỉ đích IP từ byte thứ 17 đến 20

    printf("IP dst: %u.%u.%u.%u\n",

        (unsigned char)packet\_bytes[16],

        (unsigned char)packet\_bytes[17],

        (unsigned char)packet\_bytes[18],

        (unsigned char)packet\_bytes[19]);

}

# KIỂM TRA CÚ PHÁP NHẬP TỪ BÀN PHÍM

#include <stdio.h>

#include <string.h>

int main()

{

    char str[256];

    printf("Nhap xau ky tu: ");

    fgets(str, sizeof(str), stdin);

    // Nhập một xâu ký tự từ người dùng và lưu vào mảng str

    char cmd[16], tmp[16];

    float x, y;

    int n = sscanf(str, "%s%f%f%s", cmd , &x, &y, tmp);

    printf("n = %d\n", n);

    // Phân tích xâu ký tự str thành các giá trị cmd, x, y và tmp

    // Và lưu số lượng các giá trị đã phân tích thành công vào biến n

    if (n < 3)

    {

        printf("ERROR thieu tham so\n");

        return 1;

        // Nếu số lượng tham số phân tích thành công ít hơn 3

        // Thì in ra thông báo lỗi và kết thúc chương trình

    }

    if (n > 3)

    {

        printf("ERROR thua tham so\n");

        return 1;

        // Nếu số lượng tham số phân tích thành công lớn hơn 3

        // Thì in ra thông báo lỗi và kết thúc chương trình

    }

    if (strcmp(cmd, "ADD") != 0 && strcmp(cmd, "SUB") != 0 &&

        strcmp(cmd, "MUL") != 0 && strcmp(cmd, "DIV") != 0)

    {

        printf("ERROR sai lenh\n");

        return 1;

        // Kiểm tra giá trị của cmd

        // Nếu cmd không phải là "ADD", "SUB", "MUL" hoặc "DIV"

        // Thì in ra thông báo lỗi và kết thúc chương trình

    }

    printf("OK dung cu phap\n");

    // Nếu không có lỗi, in ra thông báo thành công

    return 0;

}

# XỬ LÝ STRING

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main()

{

    char str[] = "227 Entering Passive Mode (213,229,112,130,216,4)";

    // Chuỗi str chứa thông tin kết quả nhập cảnh (FTP response)

    unsigned char i1, i2, i3, i4; // Địa chỉ IP

    unsigned char p1, p2; // Cổng kết nối

    char \*pos = strchr(str, '(');

    // Tìm vị trí của ký tự '(' trong chuỗi str

    char \*p = strtok(pos, "(),");

    // Sử dụng strtok để tách chuỗi từ vị trí '(' thành các phần tử, phân tách bởi dấu ',' và '('

    // p trỏ tới phần tử đầu tiên trong chuỗi tách được (213)

    i1 = atoi(p);

    // Chuyển đổi giá trị của p thành số nguyên và gán cho biến i1

    p = strtok(NULL, "(),");

    // Tiếp tục lấy các giá trị phần tử tiếp theo (229)

    i2 = atoi(p);

    // Chuyển đổi giá trị của p thành số nguyên và gán cho biến i2

    p = strtok(NULL, "(),");

    // Tiếp tục lấy các giá trị phần tử tiếp theo (112)

    i3 = atoi(p);

    // Chuyển đổi giá trị của p thành số nguyên và gán cho biến i3

    p = strtok(NULL, "(),");

    // Tiếp tục lấy các giá trị phần tử tiếp theo (130)

    i4 = atoi(p);

    // Chuyển đổi giá trị của p thành số nguyên và gán cho biến i4

    p = strtok(NULL, "(),");

    // Tiếp tục lấy các giá trị phần tử tiếp theo (216)

    p1 = atoi(p);

    // Chuyển đổi giá trị của p thành số nguyên và gán cho biến p1

    p = strtok(NULL, "(),");

    // Tiếp tục lấy các giá trị phần tử tiếp theo (4)

    p2 = atoi(p);

    // Chuyển đổi giá trị của p thành số nguyên và gán cho biến p2

    printf("%u.%u.%u.%u:%d\n", i1, i2, i3, i4, p1 \* 256 + p2);

    // In ra địa chỉ IP và cổng kết nối

    return 0;

}

# DOMAIN TO IP

#include <stdio.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

int main()

{

    struct addrinfo \*result, \*p, hints;

    hints.ai\_family = AF\_INET;

    int ret = getaddrinfo("hust.edu.vn", "http", &hints, &result);

    // Lấy thông tin về địa chỉ IP từ tên miền "hust.edu.vn" và giao thức "http"

    // Kết quả được lưu vào cấu trúc addrinfo và con trỏ result

    if (ret != 0)

    {

        printf("Failed to get IP: %s\n", gai\_strerror(ret));

        return 1;

        // Nếu không thành công trong việc lấy địa chỉ IP

        // In ra thông báo lỗi và kết thúc chương trình

    }

    printf("Success to get IP.\n");

    // Nếu thành công, in ra thông báo thành công

    p = result;

    // Khởi tạo con trỏ p trỏ tới cấu trúc addrinfo result

    while (p != NULL)

    {

        if (p->ai\_family == AF\_INET)

        {

            printf("IPv4: %s\n",

                inet\_ntoa(((struct sockaddr\_in \*)p->ai\_addr)->sin\_addr));

            // Nếu địa chỉ IP là IPv4

            // In ra địa chỉ IP dưới dạng chuỗi trong kết cấu sockaddr\_in

        }

        else if (p->ai\_family == AF\_INET6)

        {

            char buf[64];

            printf("IPv6: %s\n", inet\_ntop(AF\_INET6,

                &((struct sockaddr\_in6 \*)p->ai\_addr)->sin6\_addr,

                buf, INET6\_ADDRSTRLEN));

            // Nếu địa chỉ IP là IPv6

            // In ra địa chỉ IP dưới dạng chuỗi trong kết cấu sockaddr\_in6

        }

        p = p->ai\_next;

        // Di chuyển đến addrinfo kế tiếp trong danh sách

    }

    return 0;

}

# SERVER (Nhận từ client, gửi tới client từ bàn phím)

#include <stdio.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket dùng để lắng nghe kết nối đến

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

        // Nếu tạo socket không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Thiết lập thông tin địa chỉ và cổng của socket

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

        // Gắn kết socket với địa chỉ và cổng đã thiết lập

        // Nếu gắn kết không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

        // Lắng nghe kết nối từ client

        // Nếu lắng nghe không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    printf("Khoi tao thanh cong!!!\n");

    // In ra thông báo thành công khi khởi tạo thành công

    struct sockaddr\_in client\_addr;

    int client\_addr\_len = sizeof(client\_addr);

    int client = accept(listener,

        (struct sockaddr \*)&client\_addr,

        &client\_addr\_len);

    // Chấp nhận kết nối từ client

    // Tạo socket mới để trao đổi dữ liệu với client

    printf("Accepted socket %d from IP: %s:%d\n",

        client,

        inet\_ntoa(client\_addr.sin\_addr),

        ntohs(client\_addr.sin\_port));

    // In ra thông tin client đã kết nối thành công, bao gồm địa chỉ IP và cổng

    char buf[256];

    int ret = recv(client, buf, sizeof(buf), 0);

    // Nhận dữ liệu từ client và lưu vào buffer

    // Trả về số lượng byte nhận được

    if (ret < sizeof(buf))

        buf[ret] = 0;

    // Đảm bảo buffer là chuỗi ký tự hợp lệ bằng cách thêm ký tự kết thúc chuỗi (null terminator)

    printf("%d bytes from client: %s", ret, buf);

    // In ra số lượng byte nhận được và nội dung dữ liệu nhận được từ client

    while (1)

    {

        printf("Enter string: ");

        fgets(buf, sizeof(buf), stdin);

        // Nhập dữ liệu từ bàn phím và lưu vào buffer

        send(client, buf, strlen(buf), 0);

        // Gửi dữ liệu từ buffer đến client

        if (strncmp(buf, "exit", 4) == 0)

            break;

        // Nếu chuỗi nhập vào là "exit", thoát khỏi vòng lặp

    }

    close(client);

    close(listener);

    // Đóng socket kết nối với client và socket lắng nghe

}

# CLIENT (Gửi lời chào tới server, nhận dữ liệu từ server)

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

int main()

{

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để kết nối tới server

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    // Thiết lập thông tin địa chỉ và cổng của server

    int ret = connect(client, (struct sockaddr \*)&addr, sizeof(addr));

    // Kết nối tới server thông qua socket đã tạo

    if (ret == -1)

    {

        perror("connect() failed");

        return 1;

        // Nếu kết nối không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    char \*msg = "Hello server\n";

    send(client, msg, strlen(msg), 0);

    // Gửi dữ liệu đến server thông qua socket đã kết nối

    char buf[256];

    while (1)

    {

        ret = recv(client, buf, sizeof(buf), 0);

        // Nhận dữ liệu từ server và lưu vào buffer

        // Trả về số lượng byte nhận được

        if (ret <= 0)

            break;

        // Nếu không nhận được dữ liệu hoặc kết thúc kết nối, thoát khỏi vòng lặp

        if (ret < sizeof(buf))

            buf[ret] = 0;

        // Đảm bảo buffer là chuỗi ký tự hợp lệ bằng cách thêm ký tự kết thúc chuỗi (null terminator)

        printf("%d bytes received: %s\n", ret, buf);

        // In ra số lượng byte nhận được và nội dung dữ liệu nhận được từ server

    }

    close(client);

    // Đóng socket kết nối tới server

}

# CLIENT DOMAIN

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

    if (argc != 2)

    {

        printf("Sai tham so.\n");

        return 1;

        // Kiểm tra số lượng tham số đầu vào. Nếu không đúng, in ra thông báo lỗi và kết thúc chương trình

    }

    // Xu ly xau argv[1]

    char \*pos = strchr(argv[1] + 7, '/');

    // Tìm vị trí xuất hiện đầu tiên của ký tự '/' trong đường dẫn

    // (bỏ qua 7 ký tự đầu tiên: "http://")

    char \*domain = NULL;

    char \*path = NULL;

    if (pos != NULL)

    {

        // Co path (/)

        int domain\_len = (pos - argv[1]) - 7;

        // Độ dài của tên miền (ký tự trước '/')

        domain = malloc(domain\_len + 1);

        memcpy(domain, argv[1] + 7, domain\_len);

        domain[domain\_len] = 0;

        // Sao chép tên miền vào biến domain và thêm ký tự kết thúc chuỗi (null terminator)

        int path\_len = strlen(argv[1]) - domain\_len - 7;

        // Độ dài của đường dẫn (ký tự sau '/')

        path = malloc(path\_len + 1);

        memcpy(path, argv[1] + domain\_len + 7, path\_len);

        path[path\_len] = 0;

        // Sao chép đường dẫn vào biến path và thêm ký tự kết thúc chuỗi (null terminator)

    }

    else

    {

        // Khong co path (/)

        int domain\_len = strlen(argv[1]) - 7;

        // Độ dài của tên miền (không có '/')

        domain = malloc(domain\_len + 1);

        memcpy(domain, argv[1] + 7, domain\_len);

        domain[domain\_len] = 0;

        // Sao chép tên miền vào biến domain và thêm ký tự kết thúc chuỗi (null terminator)

        int path\_len = 1;

        // Độ dài của đường dẫn (chỉ là '/')

        path = malloc(path\_len + 1);

        path[0] = '/';

        path[1] = 0;

        // Sao chép đường dẫn vào biến path và thêm ký tự kết thúc chuỗi (null terminator)

    }

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để kết nối tới server

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    struct addrinfo \*result;

    int ret = getaddrinfo(domain, "http", NULL, &result);

    // Lấy thông tin địa chỉ IP của tên miền từ hàm getaddrinfo

    // Sử dụng giao thức "http"

    if (ret != 0 || result == NULL)

    {

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

        return 1;

        // Nếu không lấy được thông tin địa chỉ IP hoặc gặp lỗi, in ra thông báo lỗi và kết thúc chương trình

    }

    ret = connect(client, result->ai\_addr, result->ai\_addrlen);

    // Kết nối tới server thông qua socket đã tạo và thông tin địa chỉ IP lấy được

    if (ret == -1)

    {

        perror("connect() failed");

        return 1;

        // Nếu kết nối không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    char msg[256];

    sprintf(msg, "GET %s HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n", path, domain);

    // Tạo chuỗi dữ liệu gửi đến server theo định dạng yêu cầu HTTP GET

    send(client, msg, strlen(msg), 0);

    // Gửi dữ liệu đến server thông qua socket đã kết nối

    char buf[256];

    while (1)

    {

        ret = recv(client, buf, sizeof(buf), 0);

        // Nhận dữ liệu từ server và lưu vào buffer

        // Trả về số lượng byte nhận được

        if (ret <= 0)

            break;

        // Nếu không nhận được dữ liệu hoặc kết thúc kết nối, thoát khỏi vòng lặp

        if (ret < sizeof(buf))

            buf[ret] = 0;

        // Đảm bảo buffer là chuỗi ký tự hợp lệ bằng cách thêm ký tự kết thúc chuỗi (null terminator)

        printf("%s", buf);

        // In ra dữ liệu nhận được từ server

    }

    close(client);

    // Đóng socket kết nối tới server

}

# CLIENT NHẬN FILE TỪ SERVER VÀ LƯU DẠNG TÊN MỚI

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

int main()

{

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để kết nối tới server

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    // Khai báo địa chỉ và cổng của server mà chương trình sẽ kết nối tới

    int ret = connect(client, (struct sockaddr \*)&addr, sizeof(addr));

    // Kết nối tới server thông qua socket đã tạo và địa chỉ server

    if (ret == -1)

    {

        perror("connect() failed");

        return 1;

        // Nếu kết nối không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    char buf[2048];

    // Buffer để nhận dữ liệu từ server

    int len;

    recv(client, &len, sizeof(len), 0);

    // Nhận độ dài tên file từ server (số lượng byte sẽ nhận được)

    recv(client, buf, len, 0);

    buf[len] = 0;

    // Nhận tên file từ server và thêm ký tự kết thúc chuỗi (null terminator)

    char new\_name[256] = "new\_";

    strcat(new\_name, buf);

    // Tạo tên mới cho file bằng cách thêm tiền tố "new\_"

    FILE \*f = fopen(new\_name, "wb");

    // Mở file với tên mới để ghi dữ liệu nhận được từ server

    while (1)

    {

        ret = recv(client, buf, sizeof(buf), 0);

        if (ret <= 0)

            break;

        // Nhận dữ liệu từ server và lưu vào buffer

        fwrite(buf, 1, ret, f);

        // Ghi dữ liệu từ buffer vào file

    }

    fclose(f);

    // Đóng file

    close(client);

    // Đóng socket kết nối tới server

}

# SERVER GỬI FILE TỚI CLIENT

#include <stdio.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để lắng nghe kết nối từ client

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

        // Nếu tạo socket không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Khai báo địa chỉ và cổng của server

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

        // Gán địa chỉ và cổng cho socket, nếu không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

        // Lắng nghe kết nối từ client, với một hàng đợi tối đa có thể chứa 5 kết nối đồng thời

        // Nếu không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    struct sockaddr\_in client\_addr;

    int client\_addr\_len = sizeof(client\_addr);

    int client = accept(listener,

        (struct sockaddr \*)&client\_addr,

        &client\_addr\_len);

    // Chấp nhận kết nối từ client và tạo một socket mới để giao tiếp với client

    printf("Accepted socket %d from IP: %s:%d\n",

        client,

        inet\_ntoa(client\_addr.sin\_addr),

        ntohs(client\_addr.sin\_port));

    // In ra thông tin về socket kết nối từ client (địa chỉ IP và cổng)

    char \*filename = "ecard.mp4";

    // Tên file sẽ được truyền tới client

    FILE \*f = fopen(filename, "rb");

    // Mở file để đọc dữ liệu

    char buf[2048];

    int ret;

    int len = strlen(filename);

    send(client, &len, sizeof(len), 0);

    send(client, filename, strlen(filename), 0);

    // Gửi độ dài tên file và tên file cho client

    while (!feof(f))

    {

        ret = fread(buf, 1, sizeof(buf), f);

        if (ret <= 0)

            break;

        send(client, buf, ret, 0);

        // Đọc dữ liệu từ file và gửi tới client

    }

    fclose(f);

    // Đóng file

    close(client);

    close(listener);

    // Đóng socket kết nối và socket lắng nghe

}

# CLIENT NHẬN 1 MẢNG SỐ THỰC VÀ IN RA MÀN HÌNH

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

int main()

{

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để kết nối tới server

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    // Khai báo địa chỉ và cổng của server mà chương trình muốn kết nối tới

    int ret = connect(client, (struct sockaddr \*)&addr, sizeof(addr));

    // Kết nối tới server với địa chỉ và cổng đã khai báo

    if (ret == -1)

    {

        perror("connect() failed");

        return 1;

        // Nếu kết nối không thành công, in ra thông báo lỗi và kết thúc chương trình

    }

    float arr[100];

    // Mảng để nhận dữ liệu từ server

    while (1)

    {

        ret = recv(client, arr, sizeof(arr), 0);

        // Nhận dữ liệu từ server và lưu vào mảng arr

        if (ret <= 0)

            break;

        // Nếu không nhận được dữ liệu nữa, thoát khỏi vòng lặp

        for (int i = 0; i < ret / sizeof(float); i++)

            printf("%f\n", arr[i]);

        // In giá trị của mảng arr

    }

    close(client);

    // Đóng socket kết nối tới server

}

# SERVER GỬI MẢNG CHO CLIENT

#include <stdio.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để lắng nghe kết nối từ client

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

        // Nếu không tạo được socket, in ra thông báo lỗi và kết thúc chương trình

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Địa chỉ và cổng để client kết nối tới

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

        // Nếu không bind được địa chỉ và cổng cho socket, in ra thông báo lỗi và kết thúc chương trình

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

        // Nếu không lắng nghe được kết nối từ client, in ra thông báo lỗi và kết thúc chương trình

    }

    struct sockaddr\_in client\_addr;

    int client\_addr\_len = sizeof(client\_addr);

    int client = accept(listener,

        (struct sockaddr \*)&client\_addr,

        &client\_addr\_len);

    // Chấp nhận kết nối từ client và tạo socket để giao tiếp với client

    printf("Accepted socket %d from IP: %s:%d\n",

        client,

        inet\_ntoa(client\_addr.sin\_addr),

        ntohs(client\_addr.sin\_port));

    // In ra thông tin kết nối của client

    float arr[200];

    int n;

    while (1)

    {

        printf("Nhap so phan tu mang: ");

        scanf("%d", &n);

        // Nhập số phần tử của mảng từ người dùng

        for (int i = 0; i < n; i++)

            arr[i] = i;

        // Gán giá trị các phần tử của mảng là các số tăng dần từ 0 đến n-1

        send(client, arr, n \* sizeof(float), 0);

        // Gửi mảng về client

        if (n == 0)

            break;

        // Nếu số phần tử là 0, thoát khỏi vòng lặp

    }

    close(client);

    close(listener);

    // Đóng socket kết nối và socket lắng nghe

}

# HTTP SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để lắng nghe kết nối từ client

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

        // Nếu không tạo được socket, in ra thông báo lỗi và kết thúc chương trình

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Địa chỉ và cổng để client kết nối tới

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

        // Nếu không bind được địa chỉ và cổng cho socket, in ra thông báo lỗi và kết thúc chương trình

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

        // Nếu không lắng nghe được kết nối từ client, in ra thông báo lỗi và kết thúc chương trình

    }

    printf("Khoi tao thanh cong!!!\n");

    // In ra thông báo thành công khi server đã sẵn sàng để lắng nghe kết nối từ client

    struct sockaddr\_in client\_addr;

    int client\_addr\_len = sizeof(client\_addr);

    int client = accept(listener,

        (struct sockaddr \*)&client\_addr,

        &client\_addr\_len);

    // Chấp nhận kết nối từ client và tạo socket để giao tiếp với client

    printf("Accepted socket %d from IP: %s:%d\n",

        client,

        inet\_ntoa(client\_addr.sin\_addr),

        ntohs(client\_addr.sin\_port));

    // In ra thông tin kết nối của client

    char buf[256];

    int ret;

    char \*request = NULL;

    int size = 0;

    while (1)

    {

        ret = recv(client, buf, sizeof(buf), 0);

        if (ret <= 0)

            break;

        request = realloc(request, size + ret);

        memcpy(request + size, buf, ret);

        size += ret;

        if (strstr(request, "\r\n\r\n") != NULL)

            break;

    }

    // Nhận yêu cầu HTTP từ client, với giới hạn kích thước mảng đệm buf là 256 bytes

    printf("%d bytes from client: %s", size, request);

    free(request);

    // In ra thông tin yêu cầu từ client và giải phóng bộ nhớ đã cấp phát cho yêu cầu

    strcpy(buf, "HTTP/1.1 200 OK\r\nContentType: text/html\r\n\r\n<html><body><h1>Hello World</h1></body></html>");

    send(client, buf, strlen(buf), 0);

    // Tạo phản hồi HTTP thành công ("Hello World") và gửi phản hồi về client

    close(client);

    close(listener);

    // Đóng socket kết nối và socket lắng nghe

}

# UDP RECEIVER

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

int main()

{

    int receiver = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    // Tạo socket để nhận gói tin UDP

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_DGRAM (UDP), và giao thức UDP (IPPROTO\_UDP)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Địa chỉ và cổng để receiver lắng nghe gói tin

    bind(receiver, (struct sockaddr \*)&addr, sizeof(addr));

    // Liên kết địa chỉ và cổng với socket receiver để lắng nghe các gói tin đến

    char buf[16];

    while (1)

    {

        int ret = recvfrom(receiver, buf, sizeof(buf), 0,

            NULL, NULL);

        // Nhận gói tin UDP và lưu nội dung vào mảng đệm buf

        // Hàm recvfrom cũng trả về số lượng byte đã nhận được và thông tin về địa chỉ nguồn gửi gói tin (được bỏ qua trong ví dụ này)

        if (ret <= 0)

        {

            break;

        }

        // Nếu không nhận được gói tin hoặc có lỗi, thoát khỏi vòng lặp

        buf[ret] = 0;

        // Đảm bảo kết thúc chuỗi nội dung bằng ký tự null

        printf("%d bytes received: %s\n", ret, buf);

        // In ra số lượng byte nhận được và nội dung của gói tin

    }

}

# UDP SENDER

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

int main()

{

    int sender = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    // Tạo socket để gửi gói tin UDP

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_DGRAM (UDP), và giao thức UDP (IPPROTO\_UDP)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    // Địa chỉ và cổng của receiver để gửi gói tin UDP đến

    char buf[256];

    while (1)

    {

        printf("Nhap xau: ");

        fgets(buf, sizeof(buf), stdin);

        // Nhập xâu từ bàn phím và lưu vào mảng đệm buf

        int ret = sendto(sender, buf, strlen(buf), 0,

            (struct sockaddr \*)&addr, sizeof(addr));

        // Gửi gói tin UDP chứa nội dung xâu từ buf tới địa chỉ và cổng của receiver

        printf("%d bytes sent\n", ret);

        // In ra số lượng byte đã gửi đi

        if (strncmp(buf, "exit", 4) == 0)

            break;

        // Nếu người dùng nhập "exit", thoát khỏi vòng lặp

    }

}

# BLOCKING CLIENT

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <string.h>

#include <arpa/inet.h>

int main() {

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để thiết lập kết nối TCP

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    // Địa chỉ và cổng của server để thiết lập kết nối TCP

    if (connect(client, (struct sockaddr \*)&addr, sizeof(addr))) {

        perror("connect() failed");

        return 1;

    }

    // Thiết lập kết nối TCP với server

    char buf[256];

    while (1)

    {

        // Nhận dữ liệu từ bàn phím và gửi đến server

        printf("Nhap xau: ");

        fgets(buf, sizeof(buf), stdin);

        send(client, buf, strlen(buf), 0);

        // Nhận dữ liệu từ server và in ra màn hình

        int ret = recv(client, buf, sizeof(buf), 0);

        if (ret <= 0)

            break;

        buf[ret] = 0;

        printf("Received: %s\n", buf);

    }

    // Kết thúc, đóng socket

    close(client);

    return 0;

}

# BLOCKING SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để lắng nghe kết nối TCP

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Thiết lập địa chỉ và cổng lắng nghe

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    // Ràng buộc địa chỉ và cổng lắng nghe với socket

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    // Bắt đầu lắng nghe kết nối đến từ client

    int clients[64];

    int num\_clients = 0;

    // Mảng lưu trữ các client đã kết nối và số lượng client hiện tại

    char buf[256];

    while (1)

    {

        int client = accept(listener, NULL, NULL);

        // Chấp nhận kết nối mới từ client

        // Lưu trữ socket của client được chấp nhận vào mảng clients

        if (client == -1)

        {

            perror("accept() failed");

            break;

        }

        printf("New client connected: %d\n", client);

        clients[num\_clients++] = client;

        // In ra màn hình thông báo về client mới kết nối và lưu trữ socket của client vào mảng clients

        for (int i = 0; i < num\_clients; i++)

        {

            int ret = recv(clients[i], buf, sizeof(buf), 0);

            // Nhận dữ liệu từ client

            // Lưu trữ dữ liệu vào buffer buf

            if (ret <= 0)

                continue;

            // Nếu không nhận được dữ liệu hoặc có lỗi xảy ra, tiếp tục vòng lặp để nhận từ các client khác

            buf[ret] = 0;

            printf("Received from %d: %s\n", clients[i], buf);

            // In ra màn hình dữ liệu nhận được từ client

        }

    }

    close(listener);

    // Đóng socket lắng nghe

    return 0;

}

# NON-BLOCKING SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

#include <sys/ioctl.h>

#include <errno.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket để lắng nghe kết nối TCP

    // Sử dụng giao thức IPv4 (AF\_INET), kiểu socket SOCK\_STREAM (TCP), và giao thức TCP (IPPROTO\_TCP)

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    unsigned long ul = 1;

    ioctl(listener, FIONBIO, &ul);

    // Đặt socket lắng nghe vào chế độ không chặn (non-blocking mode)

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    // Thiết lập địa chỉ và cổng lắng nghe

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    // Ràng buộc địa chỉ và cổng lắng nghe với socket

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    // Bắt đầu lắng nghe kết nối đến từ client

    int clients[64];

    int num\_clients = 0;

    // Mảng lưu trữ các client đã kết nối và số lượng client hiện tại

    char buf[256];

    while (1)

    {

        int client = accept(listener, NULL, NULL);

        // Chấp nhận kết nối mới từ client

        // Lưu trữ socket của client được chấp nhận vào mảng clients

        if (client == -1)

        {

            if (errno == EWOULDBLOCK || errno == EAGAIN)

            {

                // Nếu không có kết nối mới đến hoặc còn chờ dữ liệu từ client

                // Bỏ qua và tiếp tục vòng lặp

            }

            else

            {

                perror("accept() failed");

                break;

            }

        }

        else

        {

            printf("New client connected: %d\n", client);

            clients[num\_clients++] = client;

            unsigned long ul = 1;

            ioctl(client, FIONBIO, &ul);

            // Đặt socket client vào chế độ không chặn (non-blocking mode)

        }

        for (int i = 0; i < num\_clients; i++)

        {

            int ret = recv(clients[i], buf, sizeof(buf), 0);

            // Nhận dữ liệu từ client

            if (ret == -1)

            {

                if (errno == EWOULDBLOCK || errno == EAGAIN)

                {

                    // Nếu không có dữ liệu mới từ client hoặc còn chờ dữ liệu

                    // Bỏ qua và chuyển sang client tiếp theo

                }

                else

                {

                    perror("recv() failed");

                    continue;

                }

            }

            else if (ret == 0)

            {

                // Kết nối bị đóng

                // TODO: Xóa socket client ra khỏi mảng clients

                continue;

            }

            else

            {

                buf[ret] = 0;

                printf("Received from %d: %s\n", clients[i], buf);

            }

        }

    }

    close(listener);

    return 0;

}

# SELECT CLIENT

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <string.h>

#include <arpa/inet.h>

int main() {

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket kết nối TCP

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    fd\_set fdread, fdtest;

    // Khai báo hai tập hợp (set) fdread và fdtest để theo dõi sự kiện đầu vào

    FD\_ZERO(&fdread);

    FD\_SET(STDIN\_FILENO, &fdread);

    // Xóa tập hợp fdread và thêm đầu vào từ bàn phím (STDIN\_FILENO) vào fdread

    client = 65;

    FD\_SET(client, &fdread);

    // Thêm socket client vào tập hợp fdread

    char buf[256];

    while (1)

    {

        fdtest = fdread;

        // Sao chép tập hợp fdread vào fdtest để theo dõi

        int ret = select(client + 1, &fdtest, NULL, NULL, NULL);

        // Sử dụng hàm select để theo dõi sự kiện đầu vào từ fdtest

        if (FD\_ISSET(STDIN\_FILENO, &fdtest))

        {

            fgets(buf, sizeof(buf), stdin);

            send(client, buf, strlen(buf), 0);

            // Đọc dữ liệu từ bàn phím và gửi đi qua socket client

        }

        if (FD\_ISSET(client, &fdtest))

        {

            ret = recv(client, buf, sizeof(buf), 0);

            if (ret <= 0)

                break;

            buf[ret] = 0;

            printf("Received: %s\n", buf);

            // Nhận dữ liệu từ socket client và in ra màn hình

        }

    }

    // Kết thúc, đóng socket

    close(client);

    return 0;

}

# SELECT SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket kết nối TCP

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9090);

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    fd\_set fdread;

    // Tập hợp (set) fdread để theo dõi sự kiện đầu vào

    int clients[64];

    int num\_clients = 0;

    // Mảng lưu trữ các client socket và số lượng client hiện tại

    char buf[256];

    struct timeval tv;

    while (1)

    {

        FD\_ZERO(&fdread);

        // Xóa tập hợp fdread

        FD\_SET(listener, &fdread);

        // Thêm listener socket vào tập hợp fdread

        for (int i = 0; i < num\_clients; i++)

            FD\_SET(clients[i], &fdread);

        // Thêm các client socket vào tập hợp fdread

        tv.tv\_usec = 0;

        tv.tv\_sec = 5;

        // Thiết lập timeout cho hàm select

        int ret = select(FD\_SETSIZE, &fdread, NULL, NULL, &tv);

        // Sử dụng hàm select để theo dõi sự kiện đầu vào từ fdread

        if (ret < 0)

        {

            perror("select() failed");

            break;

        }

        if (ret == 0)

        {

            printf("Timed out.\n");

            continue;

        }

        if (FD\_ISSET(listener, &fdread))

        {

            int client = accept(listener, NULL, NULL);

            // TODO: Kiem tra gioi han

            clients[num\_clients++] = client;

            printf("New client connected %d\n", client);

            // Chấp nhận kết nối từ client mới và thêm client vào mảng clients

        }

        for (int i = 0; i < num\_clients; i++)

            if (FD\_ISSET(clients[i], &fdread))

            {

                int ret = recv(clients[i], buf, sizeof(buf), 0);

                if (ret <= 0)

                {

                    // TODO: Xoa client ra khoi mang

                    continue;

                }

                buf[ret] = 0;

                printf("Received from %d: %s\n", clients[i], buf);

                // Nhận dữ liệu từ client và in ra màn hình

            }

    }

    close(listener);

    return 0;

}

# SOICT CLIENT

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <unistd.h>

int main()

{

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    // Tạo socket kết nối TCP

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    if (connect(client, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("connect() failed");

        return 1;

    }

    // Kết nối đến server

    FILE \*f = fopen("test.txt", "rb");

    // Mở file "test.txt" để đọc

    char buf[256];

    while (!feof(f))

    {

        int ret = fread(buf, 1, sizeof(buf), f);

        // Đọc dữ liệu từ file

        if (ret <= 0)

            break;

        send(client, buf, ret, 0);

        // Gửi dữ liệu đến server qua kết nối TCP

    }

    fclose(f);

    // Đóng file

    close(client);

    // Đóng kết nối socket

    return 0;

}

# SOICT SERVER

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netdb.h>

#include <unistd.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    int client = accept(listener, NULL, NULL);

    // Chấp nhận kết nối từ client

    // Khai báo chuỗi cần tìm

    char \*find = "0123456789";

    int len = strlen(find) - 1; // Độ dài chuỗi đệm

    char buf[265]; // 256 + 9

    // Khởi tạo chuỗi đệm chứa dấu cách

    memset(buf, 32, len);

    int count = 0;

    while (1)

    {

        // Nhận dữ liệu mới vào vị trí sau chuỗi đệm

        int ret = recv(client, buf + len, sizeof(buf) - len, 0);

        if (ret <= 0)

            break;

        // Đếm số lần xuất hiện của chuỗi con trong buffer chứa dữ liệu nhận được

        char \*p = strstr(buf, find);

        while (p != NULL && (p - buf) < ret)

        {

            count++;

            p = strstr(p + 1, find);

        }

        // Cập nhật chuỗi đệm chứa các ký tự cuối của buffer hiện tại

        memcpy(buf, buf + ret, len);

    }

    printf("Number of sub string: %d\n", count);

    close(client);

    close(listener);

    return 0;

}

# UDP FILE RECEIVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

struct client\_info

{

    char client\_addr\_str[24];   // Luu tru dia chi client IP:Port

    char filename[256];     // Luu tru ten file can ghi

    long filesize;          // Kich thuoc file

    FILE \*file;

    int total\_count;

    int current\_count;

};

struct client\_info clients[32];

int num\_clients = 0;

int main(int argc, char \*argv[])

{

    if (argc != 2)

    {

        printf("Sai so luong tham so.\n");

        return 1;

    }

    // Tao UDP socket

    int receiver = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    // Khai bao dia chi ben nhan

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(atoi(argv[1]));

    bind(receiver, (struct sockaddr \*)&addr, sizeof(addr));

    char buf[2048];

    struct sockaddr\_in client\_addr;

    int client\_addr\_len = sizeof(client\_addr);

    char client\_addr\_str[24];

    while (1)

    {

        int ret = recvfrom(receiver, buf, sizeof(buf), 0, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

        // Tao chuoi dia chi duy nhat cua client

        sprintf(client\_addr\_str, "%s:%d", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

        // Kiem tra client da duoc luu trong danh sach chua

        // Neu co => Ghi du lieu vao file

        // Neu chua => Tao doi tuong moi

        int i = 0;

        for (; i < num\_clients; i++)

            if (strcmp(clients[i].client\_addr\_str, client\_addr\_str) == 0)

                break;

        if (i == num\_clients)

        {

            // Chua co trong danh sach

            struct client\_info new\_client;

            strcpy(new\_client.client\_addr\_str, client\_addr\_str);

            memcpy(&new\_client.filesize, buf, sizeof(new\_client.filesize));

            strcpy(new\_client.filename, "new\_");

            strcat(new\_client.filename, buf + sizeof(new\_client.filesize));

            new\_client.file = fopen(new\_client.filename, "wb");

            clients[num\_clients] = new\_client;

            num\_clients++;

            printf("Start receiving %s from %s\n", new\_client.filename, new\_client.client\_addr\_str);

        }

        else

        {

            // Co roi, ghi du lieu vao file

            fwrite(buf, 1, ret, clients[i].file);

            if (ftell(clients[i].file) == clients[i].filesize)

            {

                fclose(clients[i].file);

                printf("Finish receiving %s from %s\n", clients[i].filename, clients[i].client\_addr\_str);

                // Da ghi xong, xoa khoi mang

                if (i < num\_clients - 1)

                    clients[i] = clients[num\_clients - 1];

                num\_clients--;

            }

        }

    }

    return 0;

}

# UDP FILE RECEIVER SINGLE

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <math.h>

int main(int argc, char \*argv[])

{

    if (argc != 2)

    {

        printf("Sai so luong tham so.\n");

        return 1;

    }

    // Tao UDP socket

    int receiver = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    // Khai bao dia chi ben nhan

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    if (bind(receiver, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed.\n");

        return 1;

    }

    char buf[2048];

    printf("Waiting for filesize and name...\n");

    // Doc kich thuoc va ten file

    int ret = recvfrom(receiver, buf, sizeof(buf), 0, NULL, NULL);

    long filesize;

    char filename[256] = "new\_";

    memcpy(&filesize, buf, sizeof(filesize));

    strcat(filename, buf + sizeof(filesize));

    FILE \*file = fopen(filename, "wb");

    printf("Start receiving %s\n", filename);

    long total\_count = ceil(filesize / sizeof(buf));

    long count = 0;

    while (1)

    {

        ret = recvfrom(receiver, buf, sizeof(buf), 0, NULL, NULL);

        fwrite(buf, 1, ret, file);

        count += 1;

        if (count == total\_count)

        {

            fclose(file);

            printf("Finish receiving %s\n", filename);

            break;

        }

    }

    return 0;

}

# UDP FILE SENDER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

int main(int argc, char \*argv[])

{

    // argv[1]: dia chi IP

    // argv[2]: cong

    // argv[3]: file can chuyen

    if (argc != 4)

    {

        printf("Sai so luong tham so.\n");

        return 1;

    }

    // Tao UDP socket

    int sender = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    // Khai bao dia chi ben nhan

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

    addr.sin\_port = htons(atoi(argv[2]));

    // Doc noi dung file va gui di

    // Gửi tin nhắn

    char \*filename = argv[3];

    FILE \*f = fopen(filename, "rb");

    if (f == NULL)

    {

        printf("Khong the mo file.\n");

        return 1;

    }

    // Lay kich thuoc file

    fseek(f, 0, SEEK\_END);

    long fsize = ftell(f);

    fseek(f, 0, SEEK\_SET);

    char buf[2048];

    // Gui msg chua kich thuoc file va ten file

    memcpy(buf, &fsize, sizeof(long));

    strcpy(buf + sizeof(long), filename);

    int nbytes = sizeof(long) + strlen(filename) + 1;

    sendto(sender, buf, nbytes, 0, (struct sockaddr \*)&addr, sizeof(addr));

    // Doc noi dung file va gui sang server

    while (!feof(f))

    {

        int len = fread(buf, 1, sizeof(buf), f);

        if (len <= 0)

            break;

        sendto(sender, buf, len, 0, (struct sockaddr \*)&addr, sizeof(addr));

    }

    close(sender);

    fclose(f);

    return 0;

}

# CHAT SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9090);

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    fd\_set fdread, fdtest;

    FD\_ZERO(&fdread);

    FD\_SET(listener, &fdread);

    char buf[256];

    int users[64];      // Mang socket client da dang nhap

    char \*user\_ids[64]; // Mang luu tru id cua client da dang nhap

    int num\_users = 0;  // So client da dang nhap

    while (1)

    {

        fdtest = fdread;

        int ret = select(FD\_SETSIZE, &fdtest, NULL, NULL, NULL);

        if (ret < 0)

        {

            perror("select() failed");

            break;

        }

        for (int i = 0; i < FD\_SETSIZE; i++)

            if (FD\_ISSET(i, &fdtest))

            {

                if (i == listener)

                {

                    // Chap nhan ket noi

                    int client = accept(listener, NULL, NULL);

                    if (client < FD\_SETSIZE)

                    {

                        printf("New client connected: %d\n", client);

                        FD\_SET(client, &fdread);

                    }

                    else

                    {

                        printf("Too many connections.\n");

                        close(client);

                    }

                }

                else

                {

                    // Nhan du lieu

                    int ret = recv(i, buf, sizeof(buf), 0);

                    if (ret <= 0)

                    {

                        printf("Client %d disconnected.\n", i);

                        close(i);

                        FD\_CLR(i, &fdread);

                    }

                    else

                    {

                        buf[ret] = 0;

                        printf("Received from %d: %s\n", i, buf);

                        // Kiem tra trang thai dang nhap cua client

                        int client = i;

                        int j = 0;

                        for (; j < num\_users; j++)

                            if (users[j] == client) break;

                        if (j == num\_users)

                        {

                            // Chua dang nhap

                            // Xu ly cu phap yeu cau dang nhap

                            char cmd[32], id[32], tmp[32];

                            ret = sscanf(buf, "%s%s%s", cmd, id, tmp);

                            if (ret == 2)

                            {

                                if (strcmp(cmd, "client\_id:") == 0)

                                {

                                    char \*msg = "Dung cu phap. Gui tin nhan.\n";

                                    send(client, msg, strlen(msg), 0);

                                    int k = 0;

                                    for (; k < num\_users; k++)

                                        if (strcmp(user\_ids[k], id) == 0) break;

                                    if (k < num\_users)

                                    {

                                        char \*msg = "ID da ton tai. Yeu cau nhap lai.\n";

                                        send(client, msg, strlen(msg), 0);

                                    }

                                    else

                                    {

                                        users[num\_users] = client;

                                        user\_ids[num\_users] = malloc(strlen(id) + 1);

                                        strcpy(user\_ids[num\_users], id);

                                        num\_users++;

                                    }

                                }

                                else

                                {

                                    char \*msg = "Nhap sai. Yeu cau nhap lai.\n";

                                    send(client, msg, strlen(msg), 0);

                                }

                            }

                            else

                            {

                                char \*msg = "Nhap sai. Yeu cau nhap lai.\n";

                                send(client, msg, strlen(msg), 0);

                            }

                        }

                        else

                        {

                            // Da dang nhap

                            char sendbuf[512];

                            sprintf(sendbuf, "%s: %s", user\_ids[j], buf);

                            for (int k = 0; k < num\_users; k++)

                                if (users[k] != client)

                                    send(users[k], sendbuf, strlen(sendbuf), 0);

                        }

                    }

                }

            }

    }

    close(listener);

    return 0;

}

# POLL CLIENT

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <string.h>

#include <arpa/inet.h>

#include <poll.h>

int main() {

    int client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    addr.sin\_port = htons(9000);

    if (connect(client, (struct sockaddr \*)&addr, sizeof(addr))) {

        perror("connect() failed");

        return 1;

    }

    struct pollfd fds[2];

    fds[0].fd = STDIN\_FILENO;

    fds[0].events = POLLIN;

    fds[1].fd = client;

    fds[1].events = POLLIN;

    char buf[256];

    while (1)

    {

        int ret = poll(fds, 2, 0);

        if (ret < 0)

        {

            perror("poll() failed");

            break;

        }

        if (ret == 0)

        {

            printf("Timed out.\n");

            continue;

        }

        printf("ret = %d\n", ret);

        if (fds[0].revents & POLLIN)

        {

            // Su kien co du lieu tu ban phim

            fgets(buf, sizeof(buf), stdin);

            send(client, buf, strlen(buf), 0);

        }

        if (fds[1].revents & POLLIN)

        {

            ret = recv(client, buf, sizeof(buf), 0);

            if (ret <= 0)

                break;

            buf[ret] = 0;

            printf("Received: %s\n", buf);

        }

    }

    // Ket thuc, dong socket

    close(client);

    return 0;

}

# POLL SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

#include <poll.h>

#define MAX\_CLIENTS 64

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9000);

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    struct pollfd fds[MAX\_CLIENTS];

    int nfds = 1;

    fds[0].fd = listener;

    fds[0].events = POLLIN;

    char buf[256];

    while (1)

    {

        int ret = poll(fds, nfds, -1);

        if (ret < 0)

        {

            perror("poll() failed");

            break;

        }

        printf("ret = %d\n", ret);

        if (fds[0].revents & POLLIN)

        {

            int client = accept(listener, NULL, NULL);

            if (nfds < MAX\_CLIENTS)

            {

                printf("New client connected: %d\n", client);

                fds[nfds].fd = client;

                fds[nfds].events = POLLIN;

                nfds++;

            }

            else

            {

                printf("Too many connections\n");

                close(client);

            }

        }

        for (int i = 1; i < nfds; i++)

            if (fds[i].revents & POLLIN)

            {

                ret = recv(fds[i].fd, buf, sizeof(buf), 0);

                if (ret <= 0)

                {

                    printf("Client %d disconnected.\n", fds[i].fd);

                    close(fds[i].fd);

                    // Xoa phan tu i khoi mang

                    if (i < nfds - 1)

                        fds[i] = fds[nfds - 1];

                    nfds--;

                    i--;

                }

                else

                {

                    buf[ret] = 0;

                    printf("Received from %d: %s\n", fds[i].fd, buf);

                }

            }

    }

    close(listener);

    return 0;

}

# SELECT SERVER V2

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9090);

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    fd\_set fdread, fdtest;

    FD\_ZERO(&fdread);

    FD\_SET(listener, &fdread);

    char buf[256];

    struct timeval tv;

    while (1)

    {

        fdtest = fdread;

        tv.tv\_usec = 0;

        tv.tv\_sec = 5;

        int ret = select(FD\_SETSIZE, &fdtest, NULL, NULL, &tv);

        if (ret < 0)

        {

            perror("select() failed");

            break;

        }

        if (ret == 0)

        {

            printf("Timed out.\n");

            continue;

        }

        for (int i = 0; i < FD\_SETSIZE; i++)

            if (FD\_ISSET(i, &fdtest))

            {

                if (i == listener)

                {

                    // Chap nhan ket noi

                    int client = accept(listener, NULL, NULL);

                    if (client < FD\_SETSIZE)

                    {

                        printf("New client connected: %d\n", client);

                        FD\_SET(client, &fdread);

                    }

                    else

                    {

                        printf("Too many connections.\n");

                        close(client);

                    }

                }

                else

                {

                    // Nhan du lieu

                    int ret = recv(i, buf, sizeof(buf), 0);

                    if (ret <= 0)

                    {

                        printf("Client %d disconnected.\n", i);

                        close(i);

                        FD\_CLR(i, &fdread);

                    }

                    else

                    {

                        buf[ret] = 0;

                        printf("Received from %d: %s\n", i, buf);

                    }

                }

            }

    }

    close(listener);

    return 0;

}

# CHAT SERVER V2

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int main()

{

    // Tạo socket

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    // Thiết lập địa chỉ và cổng của server

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9090);

    // Gắn địa chỉ và cổng vào socket

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)))

    {

        perror("bind() failed");

        return 1;

    }

    // Lắng nghe kết nối từ client

    if (listen(listener, 5))

    {

        perror("listen() failed");

        return 1;

    }

    // Khởi tạo tập hợp các socket đang đọc và tập hợp để kiểm tra sự kiện

    fd\_set fdread, fdtest;

    FD\_ZERO(&fdread);

    FD\_SET(listener, &fdread);

    // Mảng lưu trữ các socket của client đã đăng nhập

    int users[64];

    // Mảng lưu trữ id của client đã đăng nhập

    char \*user\_ids[64];

    // Số lượng client đã đăng nhập

    int num\_users = 0;

    while (1)

    {

        // Sao chép tập hợp đang đọc vào tập hợp để kiểm tra sự kiện

        fdtest = fdread;

        // Gọi select để kiểm tra sự kiện trên các socket

        int ret = select(FD\_SETSIZE, &fdtest, NULL, NULL, NULL);

        if (ret < 0)

        {

            perror("select() failed");

            break;

        }

        for (int i = 0; i < FD\_SETSIZE; i++)

            if (FD\_ISSET(i, &fdtest))

            {

                if (i == listener)

                {

                    // Chap nhan ket noi

                    int client = accept(listener, NULL, NULL);

                    if (client < FD\_SETSIZE)

                    {

                        printf("New client connected: %d\n", client);

                        FD\_SET(client, &fdread);

                    }

                    else

                    {

                        printf("Too many connections.\n");

                        close(client);

                    }

                }

                else

                {

                    // Nhan du lieu

                    int ret = recv(i, buf, sizeof(buf), 0);

                    if (ret <= 0)

                    {

                        printf("Client %d disconnected.\n", i);

                        close(i);

                        FD\_CLR(i, &fdread);

                    }

                    else

                    {

                        buf[ret] = 0;

                        printf("Received from %d: %s\n", i, buf);

                        // Kiem tra trang thai dang nhap cua client

                        int client = i;

                        int j = 0;

                        for (; j < num\_users; j++)

                            if (users[j] == client) break;

                        if (j == num\_users)

                        {

                            // Chua dang nhap

                            // Xu ly cu phap yeu cau dang nhap

                            char cmd[32], id[32], tmp[32];

                            ret = sscanf(buf, "%s%s%s", cmd, id, tmp);

                            if (ret == 2)

                            {

                                if (strcmp(cmd, "client\_id:") == 0)

                                {

                                    char \*msg = "Dung cu phap. Gui tin nhan.\n";

                                    send(client, msg, strlen(msg), 0);

                                    int k = 0;

                                    for (; k < num\_users; k++)

                                        if (strcmp(user\_ids[k], id) == 0) break;

                                    if (k < num\_users)

                                    {

                                        char \*msg = "ID da ton tai. Yeu cau nhap lai.\n";

                                        send(client, msg, strlen(msg), 0);

                                    }

                                    else

                                    {

                                        users[num\_users] = client;

                                        user\_ids[num\_users] = malloc(strlen(id) + 1);

                                        strcpy(user\_ids[num\_users], id);

                                        num\_users++;

                                    }

                                }

                                else

                                {

                                    char \*msg = "Nhap sai. Yeu cau nhap lai.\n";

                                    send(client, msg, strlen(msg), 0);

                                }

                            }

                            else

                            {

                                char \*msg = "Nhap sai. Yeu cau nhap lai.\n";

                                send(client, msg, strlen(msg), 0);

                            }

                        }

                        else

                        {

                            // Da dang nhap

                            char target\_id[32];

                            ret = sscanf(buf, "%s", target\_id);

                            char sendbuf[512];

                            sprintf(sendbuf, "%s: %s", user\_ids[j], buf + strlen(target\_id) + 1);

                            if (strcmp(target\_id, "all") == 0)

                            {

                                for (int k = 0; k < num\_users; k++)

                                    if (users[k] != client)

                                        send(users[k], sendbuf, strlen(sendbuf), 0);

                            }

                            else

                            {

                                for (int k = 0; k < num\_users; k++)

                                    if (strcmp(user\_ids[k], target\_id) == 0)

                                        send(users[k], sendbuf, strlen(sendbuf), 0);

                            }

                        }

                    }

                }

            }

    }

    // Đóng socket lắng nghe

    close(listener);

    return 0;

}

# TELNET SERVER

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int users[64]; // Mảng lưu trữ các socket của client đã đăng nhập

int num\_users = 0; // Số lượng client đã đăng nhập

void process\_request(int client, char \*buf); // Hàm xử lý yêu cầu từ client

void remove\_user(int client); // Hàm xóa client đã ngắt kết nối

int main()

{

    int listener = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); // Tạo socket

    if (listener == -1)

    {

        perror("socket() failed");

        return 1;

    }

    struct sockaddr\_in addr;

    addr.sin\_family = AF\_INET;

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    addr.sin\_port = htons(9090);

    if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr))) // Gắn địa chỉ và cổng vào socket

    {

        perror("bind() failed");

        return 1;

    }

    if (listen(listener, 5)) // Lắng nghe kết nối từ client

    {

        perror("listen() failed");

        return 1;

    }

    fd\_set fdread, fdtest;

    FD\_ZERO(&fdread);

    FD\_SET(listener, &fdread);

    char buf[256]; // Buffer để nhận dữ liệu từ client

    while (1)

    {

        fdtest = fdread;

        int ret = select(FD\_SETSIZE, &fdtest, NULL, NULL, NULL); // Gọi select để kiểm tra sự kiện trên các socket

        if (ret < 0)

        {

            perror("select() failed");

            break;

        }

        for (int i = 0; i < FD\_SETSIZE; i++)

            if (FD\_ISSET(i, &fdtest))

            {

                if (i == listener)

                {

                    // Chấp nhận kết nối từ client mới

                    int client = accept(listener, NULL, NULL);

                    if (client < FD\_SETSIZE)

                    {

                        printf("New client connected: %d\n", client);

                        FD\_SET(client, &fdread);

                    }

                    else

                    {

                        printf("Too many connections.\n");

                        close(client);

                    }

                }

                else

                {

                    // Nhận dữ liệu từ client

                    int ret = recv(i, buf, sizeof(buf), 0);

                    if (ret <= 0)

                    {

                        printf("Client %d disconnected.\n", i);

                        close(i);

                        remove\_user(i);

                        FD\_CLR(i, &fdread);

                    }

                    else

                    {

                        buf[ret] = 0;

                        printf("Received from %d: %s\n", i, buf);

                        // Xử lý yêu cầu từ client

                        process\_request(i, buf);

                    }

                }

            }

    }

    close(listener);

    return 0;

}

void process\_request(int client, char \*buf)

{

    // Kiểm tra trạng thái đăng nhập của client

    int i = 0;

    for (; i < num\_users; i++)

        if (users[i] == client)

            break;

    if (i == num\_users)

    {

        // Chưa đăng nhập

        char user[32], pass[32], tmp[66], line[66];

        int ret = sscanf(buf, "%s%s%s", user, pass, tmp);

        if (ret == 2)

        {

            sprintf(tmp, "%s %s\n", user, pass);

            FILE \*f = fopen("users.txt", "r"); // Mở file chứa thông tin tài khoản

            int found = 0;

            while (fgets(line, sizeof(line), f) != NULL)

            {

                if (strcmp(line, tmp) == 0) // Kiểm tra thông tin đăng nhập

                {

                    found = 1;

                    break;

                }

            }

            fclose(f);

            if (found)

            {

                char \*msg = "Dang nhap thanh cong. Hay nhap lenh de thuc hien.\n";

                send(client, msg, strlen(msg), 0); // Gửi thông báo đăng nhập thành công cho client

                users[num\_users] = client; // Lưu socket của client đã đăng nhập

                num\_users++;

            }

            else

            {

                char \*msg = "Nhap sai tai khoan. Hay nhap lai.\n";

                send(client, msg, strlen(msg), 0); // Gửi thông báo đăng nhập thất bại cho client

            }

        }

        else

        {

            char \*msg = "Nhap sai cu phap. Hay nhap lai.\n";

            send(client, msg, strlen(msg), 0); // Gửi thông báo lỗi cú pháp cho client

        }

    }

    else

    {

        // Đã đăng nhập

        char tmp[256];

        // Xóa ký tự xuống dòng cuối nếu có

        if (buf[strlen(buf) - 1] == '\n')

            buf[strlen(buf) - 1] = '\0';

        sprintf(tmp, "%s > out.txt", buf);

        int ret = system(tmp); // Thực hiện lệnh và ghi kết quả vào file "out.txt"

        if (ret == 0)

        {

            FILE \*f = fopen("out.txt", "rb");

            while (!feof(f))

            {

                ret = fread(tmp, 1, sizeof(tmp), f);

                if (ret <= 0)

                    break;

                send(client, tmp, ret, 0); // Gửi kết quả về cho client

            }

            fclose(f);

        }

        else

        {

            char \*msg = "Lenh khong thuc hien duoc.\n";

            send(client, msg, strlen(msg), 0); // Gửi thông báo lỗi thực hiện lệnh cho client

        }

    }

}

void remove\_user(int client)

{

    int i = 0;

    for (; i < num\_users; i++)

        if (users[i] == client)

            break;

    if (i < num\_users)

    {

        if (i < num\_users - 1)

            users[i] = users[num\_users - 1];

        num\_users--;

    }

}

# UDP CHAT

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <string.h>

int main(int argc, char \*argv[])

{

    int sender = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    struct sockaddr\_in saddr;

    saddr.sin\_family = AF\_INET;

    saddr.sin\_addr.s\_addr = inet\_addr(argv[1]);

    saddr.sin\_port = htons(atoi(argv[2]));

    int receiver = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    struct sockaddr\_in raddr;

    raddr.sin\_family = AF\_INET;

    raddr.sin\_addr.s\_addr = INADDR\_ANY;

    raddr.sin\_port = htons(atoi(argv[3]));

    bind(receiver, (struct sockaddr \*)&raddr, sizeof(raddr));

    fd\_set fdread, fdtest;

    FD\_ZERO(&fdread);

    FD\_SET(STDIN\_FILENO, &fdread);

    FD\_SET(receiver, &fdread);

    char buf[256];

    while (1)

    {

        fdtest = fdread;

        int ret = select(receiver + 1, &fdtest, NULL, NULL, NULL);

        if (ret < 0)

        {

            perror("select() failed");

            break;

        }

        if (FD\_ISSET(STDIN\_FILENO, &fdtest))

        {

            fgets(buf, sizeof(buf), stdin);

            sendto(sender, buf, strlen(buf), 0,

                (struct sockaddr \*)&saddr, sizeof(saddr));

        }

        if (FD\_ISSET(receiver, &fdtest))

        {

            ret = recvfrom(receiver, buf, sizeof(buf), 0, NULL, NULL);

            buf[ret] = 0;

            printf("Received: %s\n", buf);

        }

    }

    close(sender);

    close(receiver);

}

# MỘT VÀI HÀM LƯU Ý

# Cấu trúc địa chỉ IPv4

#include <sys/types.h>

#include <sys/socket.h>

struct in\_addr {

in\_addr\_t s\_addr; /\* địa chỉ IPv4 32 bit \*/

/\* network byte ordered – big-endian \*/

};

struct sockaddr\_in {

uint8\_t sin\_len; /\* độ dài cấu trúc địa chỉ (16 bytes) \*/

sa\_family\_t sin\_family; /\* họ địa chỉ IPv4 - AF\_INET \*/

in\_port\_t sin\_port; /\* giá trị cổng \*/

/\* network byte ordered \*/

struct in\_addr sin\_addr; /\* 32 bit địa chỉ \*/

/\* network byte ordered \*/

char sin\_zero[8]; /\* không sử dụng \*/

};

• Khai báo địa chỉ trong ứng dụng server

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_addr.s\_addr = INADDR\_ANY;

addr.sin\_port = htons(9090);

• Khai báo địa chỉ trong ứng dụng client

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons(9090);

# Các hàm chuyển đổi địa chỉ

• Cần khai báo tệp <arpa/inet.h>

• Chuyển đổi địa chỉ IP dạng xâu sang số nguyên 32 bit (IPv4)

in\_addr\_t inet\_addr (

const char \*cp // xâu ký tự chứa địa chỉ IPv4

) => Hàm trả về địa chỉ dạng số nguyên, -1 nếu gặp lỗi

• Chuyển đổi địa chỉ IP dạng xâu sang cấu trúc in\_addr

int inet\_aton (

const char \*cp, // Xâu ký tự chứa địa chỉ IP

struct in\_addr \*inp // Cấu trúc địa chỉ IP

) => Hàm trả về 1 nếu thành công, 0 nếu gặp lỗi

• Chuyển đổi địa chỉ từ dạng in\_addr sang dạng xâu (IPv4)

char \*inet\_ntoa (

struct in\_addr in // Cấu trúc địa chỉ IPv4

) => Hàm trả về chuỗi ký tự chứa địa chỉ

• Chuyển đổi từ dạng số sang dạng xâu (cho IPv4 và IPv6)

const char \*inet\_ntop (

int af, // AF\_INET hoặc AF\_INET6

const void \*cp, // con trỏ in\_addr hoặc in6\_addr

char \*buf, // xâu ký tự chứa địa chỉ

socklen\_t len // INET\_ADDRSTRLEN hoặc INET6\_ADDRSTRLEN

) => hàm trả về xâu ký tự chứa địa chỉ, trả về NULL nếu gặp lỗi

• Chuyển đổi từ dạng xâu sang dạng số (cho IPv4 và IPv6)

int inet\_pton (

int af, // AF\_INET hoặc AF\_INET6

const char \*cp, // xâu địa chỉ

void \*buf // con trỏ in\_addr hoặc in6\_addr

) => Hàm trả về 1 nếu thành công, 0 nếu xâu ký tự không hợp lệ, -1

nếu gặp lỗi khác

• Chuyển đổi little-endian => big-endian (network order)

// Chuyển đổi 4 byte từ little-endian=>big-endian

uint32\_t htonl (uint32\_t hostlong)

// Chuyển đổi 2 byte từ little-endian=>big-endian

uint16\_t htons (uint16\_t hostshort)

• Chuyển đổi big-endian => little-endian (host order)

// Chuyển 4 byte từ big-endian=>little-endian

uint32\_t ntohl (uint32\_t netlong)

// Chuyển 2 byte từ big-endian=>little-endian

uint16\_t ntohs (uint16\_t netshort)

# Phân giải tên miền

struct addrinfo {

int ai\_flags; // Thường là AI\_CANONNAME

int ai\_family; // Thường là AF\_INET

int ai\_socktype; // Loại socket

int ai\_protocol; // Giao thứ giao vận

socklen\_t ai\_addrlen; // Chiều dài của ai\_addr

char\* ai\_canonname; // Tên miền

struct sockaddr\* ai\_addr; // Địa chỉ socket đã phân giải

struct addrinfo\* ai\_next; // Con trỏ tới cấu trúc sau

};

struct addrinfo \*res, \*p;

int ret = getaddrinfo(argv[1], "http", NULL, &res);

if (ret != 0)

{

printf("Failed to get IP\n");

return 1;

}

p = res;

while (p != NULL) {

if (p->ai\_family == AF\_INET) {

printf("IPv4\n");

struct sockaddr\_in addr;

memcpy(&addr, p->ai\_addr, p->ai\_addrlen);

printf("IP: %s\n", inet\_ntoa(addr.sin\_addr));

} else if (p->ai\_family == AF\_INET6) {

printf("IPv6\n");

char buf[64];

struct sockaddr\_in6 addr6;

memcpy(&addr6, p->ai\_addr, p->ai\_addrlen);

printf("IP: %s\n", inet\_ntop(p->ai\_family, &addr6.sin6\_addr,

buf, sizeof(addr6)));

}

p = p->ai\_next;

}

freeaddrinfo(res);

# Hàm socket()

#include <sys/socket.h>

int socket (

int domain, // Giao thức AF\_INET hoặc AF\_INET6

int type, // Kiểu socket SOCK\_STREAM hoặc SOCK\_DGRAM

int protocol // Giao thức IPPROTO\_TCP hoặc IPPROTO\_UDP

)

// Tạo socket TCP

int s1 = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (s1 == -1) {

printf("Không tạo được socket\n");

return 1;

}

// Tạo socket UDP

int s2 = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (s2 == -1) {

printf("Không tạo được socket\n");

return 1;

}

# Hàm bind()

#include <sys/socket.h>

int bind (

int sockfd, // mô tả của socket

const struct sockaddr \*addr, // con trỏ cấu trúc địa chỉ

socklen\_t addrlen // độ dài cấu trúc địa chỉ

) => Hàm trả về 0 nếu thành công, trả về -1 nếu gặp lỗi

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

addr.sin\_port = htons(9000);

// Gắn địa chỉ với socket

bind(listener, (struct sockaddr \*)&addr, sizeof(addr));

# Hàm listen()

#include <sys/socket.h>

int listen (

int fd, // mô tả của socket

int n // chiều dài hàng đợi chờ kết nối

)

=> Hàm trả về 0 nếu thành công, trả về -1 nếu gặp lỗi

# Hàm accept()

#include <sys/socket.h>

int accept (

int sockfd, // socket chờ kết nối đã được khởi tạo

struct sockaddr \*addr, // con trỏ địa chỉ client

socklen\_t \*addrlen // con trỏ độ dài địa chỉ client

) => Hàm trả về mô tả của socket nếu thành công (giá trị kiểu int),

trả về -1 nếu gặp lỗi

// s là socket đã được khởi tạo để chờ các kết nối

int s1 = accept(s, NULL, NULL);

// s1 là socket đại diện cho kết nối giữa server và client1

// trong trường hợp này không cần quan tâm đến địa chỉ của client1

struct sockaddr\_in clientAddr;

int clientAddrLen = sizeof(clientAddr);

int s2 = accept(s, (struct sockaddr \*)&clientAddr, &clientAddrLen);

// s2 là socket đại diện cho kết nối giữa server và client2

// clientAddr chứa dữ liệu địa chỉ của client2 (địa chỉ IP và cổng)

# Hàm send()

#include <sys/socket.h>

ssize\_t send (

int sockfd, // socket ở trạng thái đã kết nối

const void \*buf, // buffer chứa dữ liệu cần gửi

size\_t len, // số byte cần gửi

int flags // cờ quy định cách truyền, mặc định là 0

)

=> Hàm trả về số byte đã gửi nếu thành công, trả về -1 nếu

gặp lỗi

// client là socket đã được chấp nhận bởi server

// gửi đi 1 chuỗi ký tự

char\* str = "Hello Network Programming";

int ret = send(client, str, strlen(str), 0);

if (ret != -1)

printf(" %d bytes are sent", ret);

// gửi đi 1 mảng dữ liệu

char buf[256];

for (int i = 0; i < 10; i++)

buf[i] = i;

ret = send(client, buf, 10, 0);

// gửi đi biến dữ liệu bất kỳ

double d = 1.234;

ret = send(client, &d, sizeof(d), 0);

# Hàm write()

#include <unistd.h>

ssize\_t write (

int fd, // socket ở trạng thái đã kết nối

const void \*buf, // buffer chứa dữ liệu cần gửi

size\_t n // số byte cần gửi

)

=> Hàm trả về số byte đã gửi nếu thành công, trả về -1 nếu

gặp lỗi

// client là socket đã được chấp nhận bởi server

// gửi đi 1 chuỗi ký tự

char\* str = "Hello Network Programming";

int ret = write(client, str, strlen(str));

if (ret != -1)

printf(" %d bytes are sent", res);

// gửi đi 1 mảng dữ liệu

char buf[256];

for (int i = 0; i < 10; i++)

buf[i] = i;

ret = write(client, buf, 10);

// gửi đi biến dữ liệu bất kỳ

double d = 1.234;

ret = write(client, &d, sizeof(d));

# Hàm recv()

ssize\_t recv (

int sockfd, // socket ở trạng thái đã kết nối

void \*buf, // buffer chứa dữ liệu sẽ nhận được

size\_t n, // số byte muốn nhận (độ dài của buffer)

int flags // cờ quy định cách nhận, mặc định là 0

)

=> Hàm trả về số byte đã nhận nếu thành công, trả về 0 nếu

kết nối bị đóng, trả về -1 nếu gặp lỗi

// client là socket đã được chấp nhận bởi server

char buf[256];

// nhận 1 buffer dữ liệu

int ret = recv(client, buf, sizeof(buf), 0);

// nhận biến dữ liệu bất kỳ

double d;

ret = recv(client, &d, sizeof(d), 0);

// nhận dữ liệu đến khi ngắt kết nối

while (true) {

ret = recv(client, buf, sizeof(buf), 0);

// kiểm tra điều kiện kết nối

if (ret <= 0)

break;

// xử lý dữ liệu nhận được

}

# Hàm read()

#include <unistd.h>

ssize\_t read (

int fd, // socket ở trạng thái đã kết nối

void \*buf, // buffer chứa dữ liệu sẽ nhận được

size\_t nbytes // số byte muốn nhận (độ dài của buffer)

)

=> Hàm trả về số byte đã nhận nếu thành công, trả về 0 nếu

kết nối bị đóng, trả về -1 nếu gặp lỗi

// client là socket đã được chấp nhận bởi server

char buf[256];

// nhận 1 buffer dữ liệu

int ret = read(client, buf, sizeof(buf));

// nhận biến dữ liệu bất kỳ

double d;

ret = read(client, &d, sizeof(d));

// nhận dữ liệu đến khi ngắt kết nối

while (true) {

ret = read(client, buf, sizeof(buf));

// kiểm tra điều kiện kết nối

if (ret <= 0)

break;

// xử lý dữ liệu nhận được

}

# Hàm close()

#include <unistd.h>

int close (

int sockfd // socket cần đóng kết nối

)

=> Hàm trả về 0 nếu thành công, trả về -1 nếu gặp lỗi

# Hàm shutdown()

#include <sys/socket.h>

int shutdown (

int sockfd, // socket cần đóng kết nối

int how, // cách thức đóng kết nối

)

Lệnh shutdown(fd, SHUT\_RDWR) tương đương lệnh

close(fd)

# Hàm connect()

int connect (

int sockfd, // socket đã được tạo

const struct sockaddr \*addr, // con trỏ địa chỉ server

socklen\_t addrlen // độ dài cấu trúc địa chỉ

)

=> Hàm trả về 0 nếu thành công, trả về -1 nếu gặp lỗi

# Hàm sendto() (UDP)

ssize\_t sendto (

int sockfd, // socket đã được khởi tạo

const void \*buf, // buffer chứa dữ liệu cần gửi

size\_t len, // số byte cần gửi

int flags, // cờ quy định cách gửi, mặc định là 0

const struct sockaddr \*addr, // con trỏ địa chỉ bên nhận

socklen\_t addr\_len // độ dài cấu trúc địa chỉ

)

=> Hàm trả về số byte đã gửi nếu thành công, trả

về -1 nếu gặp lỗi

# Hàm recvfrom() (UDP)

ssize\_t recvfrom (

int sockfd, // socket đã khởi tạo

void \*buf, // buf chứa dữ liệu nhận được

size\_t len, // số byte muốn nhận (kích thước buffer)

int flags, // cờ quy định cách nhận, mặc định là 0

struct sockaddr \*src\_addr, // con trỏ địa chỉ bên gửi

socklen\_t \*addr\_len // con trỏ độ dài địa chỉ

)

=> Hàm trả về số byte đã nhận nếu thành công, trả

về -1 nếu gặp lỗi

# Hàm ioctl() -> bất đồng bộ

unsigned long ul = 1;

ioctl(socket\_fd, FIONBIO, &ul);

# Hàm fcntl() -> bất đồng bộ

fcntl(socket\_fd, F\_SETFL, O\_NONBLOCK);

# Hàm select()

Cú pháp hàm select()

#include <sys/select.h>

int select (int nfds, fd\_set \*readfds,

fd\_set \*writefds, fd\_set \*exceptfds,

struct timeval \*timeout)

• nfds: giá trị socket lớn nhất cộng 1 được gán vào 3 tập hợp

(không vượt quá FD\_SETSIZE)

• readfds: tập các socket chờ sự kiện đọc

• writefds: tập các socket chờ sự kiện ghi

• exceptfds: tập các socket chờ sự kiện ngoại lệ hoặc lỗi

• timeout: thời gian chờ các sự kiện

• NULL – chờ với thời gian vô hạn

• 0 – không chờ sự kiện nào

• > 0 – chờ với thời gian xác định

# Cấu trúc fd\_set

void FD\_CLR(int fd, fd\_set \*set); => Xóa fd ra khỏi tập set

int FD\_ISSET(int fd, fd\_set \*set); => Kiểm tra sự kiện của

fd xảy ra với tập set

void FD\_SET(int fd, fd\_set \*set); => Gắn fd vào tập set

void FD\_ZERO(fd\_set \*set); => Xóa tất cả các socket khỏi tập

set

# Cấu trúc pollfd

struct pollfd {

int fd; // Mô tả (socket) cần thăm dò

short int events; // Mặt nạ sự kiện cần kiểm tra

short int revents; // Mặt nạ sự kiện đã xảy ra

}

• Chương trình cần thiết lập mặt nạ sự kiện trong

trường events trước khi thăm dò và kiểm tra mặt nạ

sự kiện trong trường revents sau khi thăm dò.

• Một số mặt nạ sự kiện hay dùng:

• POLLIN/POLLRDNORM – Có kết nối / có dữ liệu để đọc

• POLLOUT – Sẵn sàng ghi dữ liệu

• POLLERR – Lỗi đọc / ghi dữ liệu

# Hàm poll()

int poll (

struct pollfd \*fds, // Tập hợp các mô tả cần đợi sự kiện

nfds\_t nfds, // Số lượng các mô tả, không vượt quá

RLIMIT\_NOFILE

int timeout // Thời gian chờ theo ms. Nếu bằng -1 thì hàm

chỉ trả về kết quả khi có sự kiện xảy ra.

) => Hàm trả về số lượng cấu trúc có sự kiện xảy ra nếu

thành công, trả về -1 nếu bị lỗi. Trả về 0 nếu hết giờ.

# Truyền dữ liệu sử dụng TCP - Ứng dụng server

• Tạo socket qua hàm socket()

• Gắn socket vào một giao diện mạng thông qua hàm

bind()

• Chuyển socket sang trạng thái đợi kết nối qua hàm

listen()

• Chấp nhận kết nối từ client thông qua hàm accept()

• Gửi dữ liệu tới client thông qua hàm send()/write()

• Nhận dữ liệu từ client thông qua hàm recv()/read()

• Đóng socket khi việc truyền nhận kết thúc bằng hàm

close()

# Ứng dụng client

• Tạo socket qua hàm socket()

• Điền thông tin về server vào cấu trúc sockaddr\_in

• Kết nối tới server qua hàm connect()

• Gửi dữ liệu tới server thông qua hàm send()

• Nhận dữ liệu từ server thông qua hàm recv()

• Đóng socket khi việc truyền nhận kết thúc bằng hàm

close()