

**ĐẠI HỌC BÁCH KHOA HÀ NỘI**  
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

\*\*\*\*\*



# **BÁO CÁO**

**Bài tập thực hành :**

**Chương 5: I/O Multiplexing**

**Học phần: Thực hành Lập Trình Mạng**

**Giảng viên hướng dẫn** : Trần Hải Anh

**Mã lớp** : 151907

**Sinh viên thực hiện** : Phạm Vân Anh

**Mã số sinh viên** : 20214988

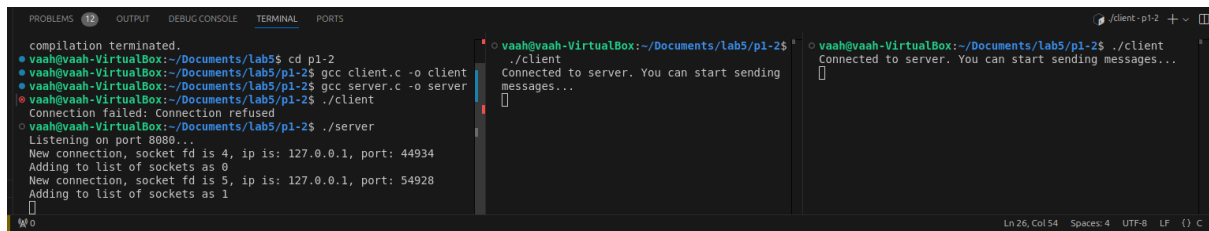
**Hà Nội, tháng 10 năm 2024**

## **Mục lục**

<b>Phần 1: Lập Trình Socket TCP - Thiết Lập Cơ Bản.....</b>	<b>3</b>
1.1    Triển Khai Server TCP.....	3
1.2    Server phát tin nhắn từ 1 client và phát tới các client khác .....	3
1.3    Kết quả: .....	3
<b>Phần 2: Logic Server và Phát Tin Nhắn.....</b>	<b>4</b>
2.1: Phát Tin Nhắn Đến Tất Cả Các Client .....	4
2.2: Xử Lý Client Ngắt Kết Nối .....	5
<b>Phần 3: I/O Multiplexing để Xử Lý Nhiều Client.....</b>	<b>7</b>
3.1: Sử Dụng select() để Xử Lý Nhiều Client .....	7
3.2: Sử Dụng pselect() để Thực Hiện I/O Multiplexing An Toàn Với Tín Hiệu 8	
3.3: Sử Dụng poll() để Xử Lý Nhiều Client .....	10
3.3: Tổng kết .....	12
<b>Phần 4: Tính Năng Phòng Chat và Trải Nghiệm Người Dùng .....</b>	<b>16</b>
4.3: Cách mà chức năng phòng chat (phát tin nhắn, tên người dùng, v.v.) được triển khai .....	18
4.4 Những khó khăn gặp phải trong quá trình triển khai và cách khắc phục:..	19

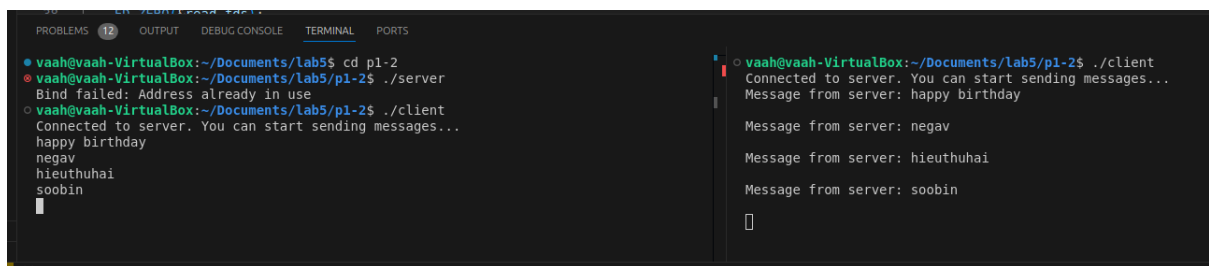
# Phần 1: Lập Trình Socket TCP - Thiết Lập Cơ Bản

## 1.1 Triển Khai Server TCP



```
compilation terminated.
vaah@vaah-VirtualBox:~/Documents/lab5$ cd p1-2
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ gcc client.c -o client
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ gcc server.c -o server
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connection failed: Connection refused
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./server
Listening on port 8080...
New connection, socket fd is 4, ip is: 127.0.0.1, port: 44934
Adding to list of sockets as 0
New connection, socket fd is 5, ip is: 127.0.0.1, port: 54928
Adding to list of sockets as 1
[]
```

## 1.2 Server phát tin nhắn từ 1 client và phát tới các client khác



```
vaah@vaah-VirtualBox:~/Documents/lab5$ cd p1-2
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./server
Bind failed: Address already in use
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
happy birthday
negav
hieuthuhai
soobin
[]

vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
Message from server: happy birthday

Message from server: negav

Message from server: hieuthuhai

Message from server: soobin
[]
```

## 1.3 Kết quả:

- Server có thể xử lý nhiều client, phát tin nhắn từ một client tới client khác trong phòng chat.
- Mỗi client có thể gửi tin nhắn và nhìn thấy tin nhắn của các client khác trong thời gian thực.

## Phần 2: Logic Server và Phát Tin Nhắn

### 2.1: Phát Tin Nhắn Đến Tất Cả Các Client

- Khi gửi tin nhắn từ 1 client, các client khác sẽ nhận được tin nhắn.

```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Bind failed: Address already in use
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
Message from server: helo guys

Message from server: saranghae

Message from server: anhong

Message from server: negav

babi
█
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
Message from server: helo guys

Message from server: saranghae

Message from server: anhong

negav
Message from server: babi

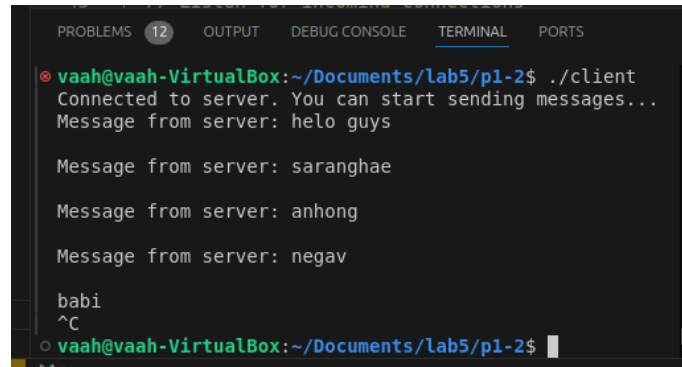
█
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
helo guys
saranghae
anhong
Message from server: negav

Message from server: babi
█
```

## 2.2: Xử Lý Client Ngắt Kết Nối

- Client ngắt kết nối



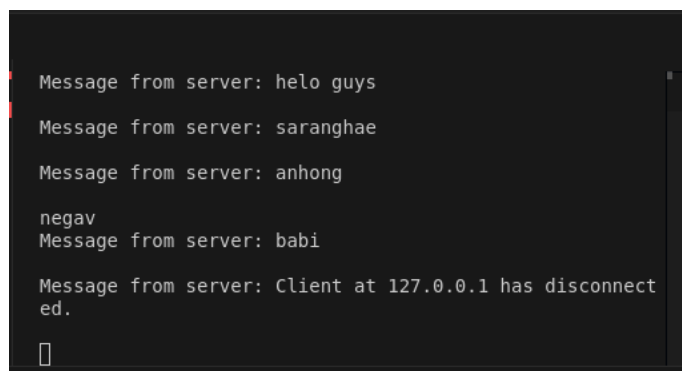
```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
Message from server: helo guys

Message from server: saranghae

Message from server: anhong

Message from server: negav

babi
^C
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$
```



```
Message from server: helo guys

Message from server: saranghae

Message from server: anhong

negav
Message from server: babi

Message from server: Client at 127.0.0.1 has disconnected.


```

- Server hủy kết nối

```
if (FD_ISSET(sd, &readfds)) {
    // Check if it was for closing, and read the incoming message
    if ((valread = read(sd, buffer, BUFFER_SIZE)) == 0) {
        // Someone disconnected, get details
        getpeername(sd, (struct sockaddr*)&address, (socklen_t*)&addrlen);
        printf("Client disconnected, ip %s, port %d\n", inet_ntoa(address.sin_addr),
            ntohs(address.sin_port));

        // Close the socket and mark as 0
        close(sd);
        client_sockets[i] = 0;

        // Inform other clients that a user has disconnected
        char disconnect_msg[BUFFER_SIZE];
        snprintf(disconnect_msg, sizeof(disconnect_msg), "Client at %s has disconnected.\n",
            inet_ntoa(address.sin_addr));
        for (int j = 0; j < max_clients; j++) {
            if (client_sockets[j] > 0) {
                send(client_sockets[j], disconnect_msg, strlen(disconnect_msg), 0);
            }
        }
    }
}
```

Kết quả:

```
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
babi sarang hae
Message from server: chincha

Message from server: Client at 127.0.0.1 has disconnected.

□
```

```
⊗ vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./client
Connected to server. You can start sending messages...
Message from server: babi sarang hae

chincha
^C
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ █
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p1-2$ ./server
Listening on port 8080...
New connection, socket fd is 4, ip is: 127.0.0.1, port: 59236
Adding to list of sockets as 0
New connection, socket fd is 5, ip is: 127.0.0.1, port: 59252
Adding to list of sockets as 1
Client disconnected, ip 127.0.0.1, port 59252
□
```

## Phần 3: I/O Multiplexing để Xử Lý Nhiều Client

### 3.1: Sử Dụng select() để Xử Lý Nhiều Client

- Server

```
// Clear the socket set
FD_ZERO(&readfds);

// Add server socket to set
FD_SET(server_fd, &readfds);
int max_sd = server_fd;

// Add client sockets to set
for (i = 0; i < max_clients; i++) {
    sd = client_sockets[i];
    if (sd > 0) FD_SET(sd, &readfds);
    if (sd > max_sd) max_sd = sd;
}

// Wait for activity on one of the sockets
activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);

// Incoming connection
if (FD_ISSET(server_fd, &readfds)) {
    if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }
    printf("New connection, socket fd is %d, ip is: %s, port: %d\n", new_socket,
inet_ntoa(address.sin_addr), ntohs(address.sin_port));

    // Add new socket to array
    for (i = 0; i < max_clients; i++) {
        if (client_sockets[i] == 0) {
            client_sockets[i] = new_socket;
            printf("Adding to list of sockets as %d\n", i);
            break;
        }
    }
}
```

## - Client

```
FD_ZERO(&read_fds);
FD_SET(STDIN_FILENO, &read_fds); // Monitor user input
FD_SET(sockfd, &read_fds);      // Monitor server messages

// Wait for activity
int activity = select(sockfd + 1, &read_fds, NULL, NULL, NULL);

// Check for input from the server
if (FD_ISSET(sockfd, &read_fds)) {
    int valread = read(sockfd, buffer, BUFFER_SIZE);
    if (valread == 0) {
        printf("Server disconnected.\n");
        close(sockfd);
        break;
    }
    buffer[valread] = '\0';
    printf("Message from server: %s\n", buffer);
}

// Check for user input
if (FD_ISSET(STDIN_FILENO, &read_fds)) {
    fgets(buffer, BUFFER_SIZE, stdin);
    send(sockfd, buffer, strlen(buffer), 0);
}
}
```

## 3.2: Sử Dụng pselect() để Thực Hiện I/O Multiplexing An Toàn Với Tín Hiệu

### - Server

```
// Signal mask setup
sigset_t block_mask, orig_mask;
sigemptyset(&block_mask);
sigaddset(&block_mask, SIGINT);
sigprocmask(SIG_BLOCK, &block_mask, &orig_mask); // Block SIGINT temporarily

while (!stop_server) {
    FD_ZERO(&readfds);
    FD_SET(server_fd, &readfds);
    int max_sd = server_fd;

    for (i = 0; i < max_clients; i++) {
        sd = client_sockets[i];
        if (sd > 0) FD_SET(sd, &readfds);
        if (sd > max_sd) max_sd = sd;
    }
}
```



```

}

// Use pselect for signal-safe I/O multiplexing
activity = pselect(max_sd + 1, &readfds, NULL, NULL, NULL, &orig_mask);

// Handle signal stop_server
if (activity < 0 && stop_server) {
    break;
}

```

## - Client

```

// Setup signal mask
sigset_t block_mask, orig_mask;
sigemptyset(&block_mask);
sigaddset(&block_mask, SIGINT);
sigprocmask(SIG_BLOCK, &block_mask, &orig_mask); // Block SIGINT temporarily

while (!stop_client) {
    FD_ZERO(&read_fds);
    FD_SET(STDIN_FILENO, &read_fds); // Monitor user input
    FD_SET(sockfd, &read_fds);      // Monitor server messages

    // Use pselect to handle signals safely
    int activity = pselect(sockfd + 1, &read_fds, NULL, NULL, NULL, &orig_mask);

    // If interrupted by signal, stop client
    if (activity < 0 && stop_client) {
        break;
    }

    // Check for input from the server
    if (FD_ISSET(sockfd, &read_fds)) {
        int valread = read(sockfd, buffer, BUFFER_SIZE);
        if (valread == 0) {
            printf("Server disconnected.\n");
            close(sockfd);
            break;
        }
        buffer[valread] = '\0';
        printf("Message from server: %s\n", buffer);
    }

    // Check for user input
    if (FD_ISSET(STDIN_FILENO, &read_fds)) {
        fgets(buffer, BUFFER_SIZE, stdin);
        send(sockfd, buffer, strlen(buffer), 0);
    }
}

```

```

    }
}

printf("Shutting down client...\n");
close(sockfd);

```

### 3.3: Sử Dụng poll() để Xử Lý Nhiều Client

- Server
  - Xử lý phía server:

```

// Function to broadcast a message to all clients except the sender
void broadcast_message(int sender_fd, struct pollfd *fds, int num_clients, char *message) {
for (int i = 1; i < num_clients; i++) { // Start at 1 to skip server socket
    int client_fd = fds[i].fd;
    if (client_fd != sender_fd && client_fd > 0) {
        send(client_fd, message, strlen(message), 0);
    }
}
}

```

- Xử lý kết nối mới:

```

// Check for new connection on server socket
if (fds[0].revents & POLLIN) {
    new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen);
    if (new_socket < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    printf("New connection, socket fd is %d, ip is: %s, port: %d\n", new_socket,
inet_ntoa(address.sin_addr), ntohs(address.sin_port));

    // Add new client socket to the pollfd array
    for (i = 1; i < MAX_CLIENTS; i++) {
        if (fds[i].fd == -1) {
            fds[i].fd = new_socket;
            fds[i].events = POLLIN; // Monitor this socket for read events
            break;
        }
    }

    if (i == MAX_CLIENTS) {
        printf("Too many clients connected. Connection rejected.\n");
        close(new_socket); // Too many clients connected, reject the new one
    }
}

```

- Client
  - Xử lý data từ client

```

/ Check for data from clients
for (i = 1; i < MAX_CLIENTS; i++) {
    if (fds[i].revents & POLLIN) {
        // Read incoming message
        valread = read(fds[i].fd, buffer, BUFFER_SIZE);
        if (valread == 0) {
            // Client disconnected
            getpeername(fds[i].fd, (struct sockaddr*)&address, (socklen_t*)&addrlen);
            printf("Client disconnected, ip %s, port %d\n", inet_ntoa(address.sin_addr),
ntohs(address.sin_port));
            close(fds[i].fd);
            fds[i].fd = -1; // Mark the slot as available
        } else {
            // Broadcast message to all other clients
            buffer[valread] = '\0';
            broadcast_message(fds[i].fd, fds, MAX_CLIENTS, buffer);
        }
    }
}
}

```

- Xử lý phái client:

```

while (1) {
    int poll_count = poll(fds, MAX_CLIENTS, -1); // Wait indefinitely until an event occurs

    if (poll_count < 0) {
        perror("Poll failed");
        exit(EXIT_FAILURE);
    }

    // Check for new connection on server socket
    if (fds[0].revents & POLLIN) {
        new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen);
        if (new_socket < 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }
        printf("New connection, socket fd is %d, ip is: %s, port: %d\n", new_socket,
inet_ntoa(address.sin_addr), ntohs(address.sin_port));

        // Add new client socket to the pollfd array
        for (i = 1; i < MAX_CLIENTS; i++) {
            if (fds[i].fd == -1) {

```

```

    fds[i].fd = new_socket;
    fds[i].events = POLLIN; // Monitor this socket for read events
    break;
}
}

if (i == MAX_CLIENTS) {
    printf("Too many clients connected. Connection rejected.\n");
    close(new_socket); // Too many clients connected, reject the new one
}
}

```

### 3.3: Tổng kết

Cả `select()`, `pselect()`, và `poll()` đều có thể mang lại hiệu quả khi xử lý I/O multiplexing, nhưng mỗi phương pháp lại có cơ chế hoạt động và cách sử dụng khác nhau. Dưới đây là sự phân biệt giữa từng phương pháp:

- **select()**
  - **Cách thức hoạt động:** theo dõi một danh sách các file descriptors (bao gồm các socket, pipe, stdin, stdout,...) để kiểm tra xem có sự kiện nào xảy ra (như dữ liệu sẵn sàng để đọc, sẵn sàng để ghi, hoặc lỗi) trên các file descriptors đó hay không.
  - **Giới hạn số lượng file descriptors:** có giới hạn số lượng file descriptors mà nó có thể xử lý, thường là **1024** trên nhiều hệ điều hành. Điều này có thể gây hạn chế nếu cần giám sát rất nhiều kết nối đồng thời.
  - **Cách hoạt động:**
    - Cần tính giá trị `max_fd` (file descriptor lớn nhất) và thêm các file descriptor vào `fd_set` (cấu trúc dữ liệu lưu trữ danh sách file descriptors cần giám sát).
    - Sau khi `select()` hoàn tất, cần kiểm tra thủ công từng file descriptor để xác định xem nó có sẵn sàng cho I/O hay không.
  - **Hạn chế:** Khi số lượng file descriptors lớn, `select()` có thể không hoạt động hiệu quả do việc kiểm tra từng file descriptor thủ công và sự giới hạn số lượng file descriptors mà nó hỗ trợ.

- **pselect()**

- **Cách thức hoạt động:** là một phiên bản mở rộng của select() với khả năng xử lý tín hiệu (signal). Nó cho phép tạm thời thay đổi và khôi phục mặt nạ tín hiệu (sigmask) khi đang thực hiện đợi I/O, giúp xử lý tín hiệu một cách an toàn hơn.
- **Tính năng bổ sung:**
  - Không chỉ giám sát I/O mà còn tích hợp khả năng quản lý tín hiệu một cách hiệu quả, pselect() còn có thể tạm thời khóa một số tín hiệu trong quá trình đợi I/O để tránh xung đột, và khôi phục lại mặt nạ tín hiệu sau khi pselect() kết thúc.
- **Cách hoạt động:**
  - Tương tự select(), pselect() theo dõi danh sách file descriptors để kiểm tra sự kiện I/O, nhưng với tính năng mở rộng là cung cấp mặt nạ tín hiệu (signal mask).
- **Hữu ích:** đặc biệt hữu ích trong các chương trình xử lý tín hiệu như SIGINT hoặc SIGTERM để dừng chương trình một cách an toàn.

- **poll()**

- **Cách thức hoạt động:** hoạt động tương tự select() nhưng không bị giới hạn về số lượng file descriptors. poll() sử dụng một mảng cấu trúc pollfd[] để lưu trữ danh sách các file descriptors và các sự kiện muốn theo dõi (chẳng hạn như POLLIN, POLLOUT, POLLERR,...).
- **Không cần tính max\_fd:** poll() không yêu cầu tính toán max\_fd như select(), thay vào đó chỉ cần khai báo một mảng pollfd[], trong đó mỗi phần tử của mảng chứa một file descriptor và sự kiện tương ứng cần theo dõi.
- **Tính linh hoạt cao:**
  - Có thể dễ dàng thêm hoặc xóa file descriptors khỏi mảng pollfd[] mà không bị giới hạn số lượng như select().
  - Khi số lượng file descriptors lớn, poll() hoạt động hiệu quả hơn vì không cần phải kiểm tra từng file descriptor theo cách thủ công.

- **Cách hoạt động:**
  - poll() kiểm tra tất cả các file descriptors trong mảng pollfd[], sau đó xác định file descriptor nào đã sẵn sàng cho sự kiện I/O.
- **Ưu điểm:** Hiệu quả hơn select() khi xử lý với số lượng lớn file descriptors do không có giới hạn số lượng và dễ dàng quản lý việc thêm hoặc xóa file descriptors.

Tiêu chí	select()	pselect()	poll()
<b>Cơ chế theo dõi</b>	Giám sát file descriptors bằng fd_set	Giám sát file descriptors và tín hiệu	Giám sát file descriptors bằng pollfd[]
<b>Giới hạn file descriptors</b>	Thường bị giới hạn (1024 trên nhiều hệ thống)	Tương tự select() nhưng thêm quản lý tín hiệu	Không giới hạn số lượng
<b>Xử lý tín hiệu</b>	Không tích hợp xử lý tín hiệu	Tích hợp khả năng quản lý tín hiệu an toàn	Không tích hợp tín hiệu
<b>Hiệu quả</b>	Không hiệu quả khi số lượng file descriptors lớn	Tương tự select(), nhưng tốt hơn khi cần quản lý tín hiệu	Hiệu quả hơn khi xử lý số lượng lớn file descriptors
<b>Thêm/Xóa file descriptors</b>	Phải tính lại max_fd khi thêm hoặc xóa	Tương tự select()	Dễ dàng thêm/xóa với mảng pollfd[]
<b>Độ phân giải thời gian (Time Resolution)</b>	Sử dụng <b>cấu trúc struct timeval</b> để chỉ định thời gian chờ (timeout), có độ phân giải là <b>micro giây</b>	Sử dụng <b>cấu trúc struct timespec</b> , có độ phân giải cao hơn, lên đến <b>nano giây (nanosecond)</b>	Cho phép đặt thời gian chờ bằng <b>milli giây (milliseconds)</b> .

### **Khi nào nên sử dụng từng phương pháp:**

- **select()**: Thích hợp cho các chương trình nhỏ, số lượng file descriptors giới hạn và không cần xử lý tín hiệu phức tạp.
- **pselect()**: Sử dụng khi cần xử lý đồng thời I/O và tín hiệu một cách an toàn, chẳng hạn như khi muốn dừng chương trình hoặc server khi nhận tín hiệu SIGINT.
- **poll()**: Phù hợp với các chương trình lớn, có số lượng lớn file descriptors, nơi cần linh hoạt trong việc quản lý thêm/xóa file descriptors mà không bị giới hạn về số lượng.

## Phần 4: Tính Năng Phòng Chat và Trải Nghiệm Người Dùng

Server hoạt động, có 4 client tham gia

Server thông báo có người tham gia mới vào kênh

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
● vaah@vaah-VirtualBox:~/Documents/lab5/p4$ gcc chat_server.c -o chat_server
● vaah@vaah-VirtualBox:~/Documents/lab5/p4$ gcc chat_client.c -o chat_client
○ vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_server
Listening on port 8080...
New client joined: Helo (IP: 127.0.0.1, Port: 37062)
New client joined: im here (IP: 127.0.0.1, Port: 38824)
New client joined: babi (IP: 127.0.0.1, Port: 38830)
New client joined: negav (IP: 127.0.0.1, Port: 38838)
□
```

Server nhận được tin nhắn từ 1 client, sau đó gửi đi các client khác

```
DEBUG CONSOLE TERMINAL PORTS
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: Helo
im here has joined the chat.
babi has joined the chat.
negav has joined the chat.
helo guys
im here: hi
babi: bar
negav: you
□
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: babi
negav has joined the chat.
Helo: helo guys
im here: hi
bar
negav: you
□
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: im here
babi has joined the chat.
negav has joined the chat.
Helo: helo guys
hi
babi: bar
negav: you
□
```

```
* History restored
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: negav
Helo: helo guys
im here: hi
babi: bar
you
□
```



Khi 1 client disconnect, thông báo tới các client khác

```
• vaah@vaah-VirtualBox:~/Documents/lab5/p4$ gcc chat_server.c -o chat_server
• vaah@vaah-VirtualBox:~/Documents/lab5/p4$ gcc chat_client.c -o chat_client
|o vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_server
Listening on port 8080...
New client joined: Helo (IP: 127.0.0.1, Port: 37062)
New client joined: im here (IP: 127.0.0.1, Port: 38824)
New client joined: babi (IP: 127.0.0.1, Port: 38830)
New client joined: negav (IP: 127.0.0.1, Port: 38838)
Client Helo disconnected
□
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: babi
negav has joined the chat.
Helo: helo guys
im here: hi
bar
negav: you
Helo has left the chat.
□
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: im here
babi has joined the chat.
negav has joined the chat.
Helo: helo guys
hi
babi: bar
negav: you
Helo has left the chat.
□
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: negav
Helo: helo guys
im here: hi
babi: bar
you
Helo has left the chat.
□
```

Khi các client disconnect, server tắt.

```
• vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_server
Listening on port 8080...
New client joined: Helo (IP: 127.0.0.1, Port: 37062)
New client joined: im here (IP: 127.0.0.1, Port: 38824)
New client joined: babi (IP: 127.0.0.1, Port: 38830)
New client joined: negav (IP: 127.0.0.1, Port: 38838)
Client Helo disconnected
Client im here disconnected
Client negav disconnected
Client babi disconnected
^CCaught signal 2
Shutting down server...
vaah@vaah-VirtualBox:~/Documents/lab5/p4$
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: im here
babi has joined the chat.
negav has joined the chat.
Helo: helo guys
hi
babi: bar
negav: you
Helo has left the chat.
^C
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: babi
negav has joined the chat.
Helo: helo guys
im here: hi
bar
negav: you
Helo has left the chat.
im here has left the chat.
□
```

```
vaah@vaah-VirtualBox:~/Documents/lab5/p4$ ./chat_client
Enter your username: babi
negav has joined the chat.
Helo: helo guys
im here: hi
bar
negav: you
Helo has left the chat.
im here has left the chat.
negav has left the chat.
□
```

### 4.3: Cách mà chức năng phòng chat (phát tin nhắn, tên người dùng, v.v.) được triển khai

#### - Kết nối ban đầu:

- **Client:**

- Client khởi tạo một kết nối TCP tới server bằng cách tạo socket và sử dụng hàm connect().
- Sau khi kết nối thành công, client yêu cầu người dùng nhập **username**. Sau đó, tên người dùng được gửi tới server qua socket.

- **Server:**

- Server lắng nghe các kết nối từ client bằng cách sử dụng hàm socket(), bind(), và listen().
- Khi có kết nối mới, server sử dụng hàm accept() để chấp nhận kết nối và thêm client vào danh sách các kết nối đang hoạt động.
- Server nhận **tên người dùng** từ client và lưu trữ thông tin này.
- Server gửi thông báo tới tất cả các client khác về việc có một client mới tham gia phòng chat.

#### - Phát tin nhắn giữa các client:

- **Client:**

- Client lắng nghe tin nhắn từ server bằng cách sử dụng cơ chế **poll()**. Khi có sự kiện tin nhắn từ server, client nhận tin nhắn và in ra màn hình.
- Người dùng có thể nhập tin nhắn, và client sẽ gửi tin nhắn đó tới server thông qua socket bằng cách sử dụng hàm send().

- **Server:**

- Server lắng nghe tin nhắn từ tất cả các client bằng cách sử dụng hàm select(). Khi có tin nhắn từ bất kỳ client nào, server đọc tin nhắn này.
- Sau đó, server phát (broadcast) tin nhắn đến tất cả các client khác (ngoại trừ client đã gửi tin nhắn).

- **Ngắt kết nối:**
  - **Client:**
    - Khi một client đóng kết nối (ngắt kết nối hoặc thoát), server sẽ phát hiện ngắt kết nối này qua hàm `select()` hoặc khi nhận được tín hiệu ngắt kết nối từ client.
  - **Server:**
    - Khi phát hiện một client đã ngắt kết nối, server sẽ xóa thông tin của client đó khỏi danh sách các kết nối hoạt động và đóng socket của client.
    - Server cũng gửi thông báo đến tất cả các client khác về việc client đã rời khỏi phòng chat.
- **Dừng server (Ctrl + C):**
  - **Server:**
    - Khi server nhận tín hiệu SIGINT (Ctrl + C), nó sẽ thực hiện việc dừng an toàn: thông báo cho tất cả các client rằng server đang dừng và ngắt kết nối các client.
    - Sau đó, server sẽ đóng tất cả các kết nối và kết thúc chương trình.

#### **4.4 Những khó khăn gặp phải trong quá trình triển khai và cách khắc phục:**

##### **1. Quản lý nhiều client cùng một lúc:**

- **Khó khăn:** Theo dõi đồng thời nhiều kết nối từ client và nhận/gửi tin nhắn từ tất cả client.
- **Cách khắc phục:** Sử dụng `poll()` hoặc `select()` để xử lý nhiều socket cùng lúc. Điều này cho phép server lắng nghe trên nhiều kết nối mà không cần tạo luồng mới cho mỗi kết nối.

##### **2. Đảm bảo không phát tin nhắn lại cho client đã gửi:**

- **Khó khăn:** Tránh phát tin nhắn của client gửi đi quay lại chính client đó.
- **Cách khắc phục:** Kiểm tra client đang gửi tin với các client khác trong hàm `broadcast_message()` bằng cách so sánh `sender_fd` với các client khác.

### **3. Quản lý tên người dùng:**

- Khó khăn: Khi nhận tên người dùng từ client, cần xử lý chính xác tên mà không bị lỗi mất ký tự cuối hoặc trùng lặp.
- Cách khắc phục: Dùng `fgets()` để nhận tên người dùng từ client, sau đó loại bỏ ký tự `\n` thừa bằng hàm `strcspn()`. Đảm bảo chuỗi luôn được kết thúc bởi ký tự null `\0`.

### **4. Xử lý ngắt kết nối (SIGINT):**

- Khó khăn: Khi server nhận tín hiệu ngắt (`Ctrl + C`), cần đảm bảo mọi client được ngắt kết nối an toàn.
- Cách khắc phục: Thiết lập signal handler để server có thể bắt tín hiệu và thực hiện các thao tác dọn dẹp (đóng các kết nối, in thông báo) trước khi dừng.

### **5. Hiển thị giao diện đẹp:**

- Khó khăn: Hiển thị thông báo với màu sắc khác nhau và phân biệt tin nhắn giữa các client.
- Cách khắc phục: Sử dụng các mã escape ANSI để in màu, giúp người dùng dễ dàng phân biệt giữa các loại thông báo (tin nhắn, người mới tham gia, người rời phòng).