**Bài tập thực hành**

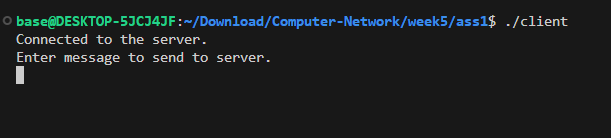
**Môn: TH Lập Trình Mạng**

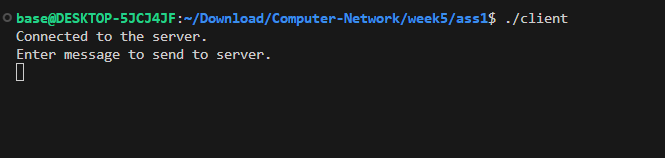
**Chương 5: I/O Multiplexing**

**Sinh viên:** Nguyễn Văn Thái - 20215135

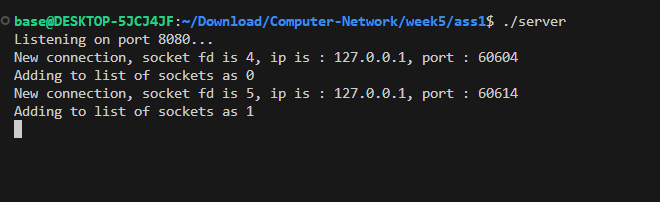
# **Phần 1: Lập Trình Socket TCP - Thiết Lập Cơ Bản**

Client:





Server hiển thị thông báo khi có client kết nối vào hệ thống:



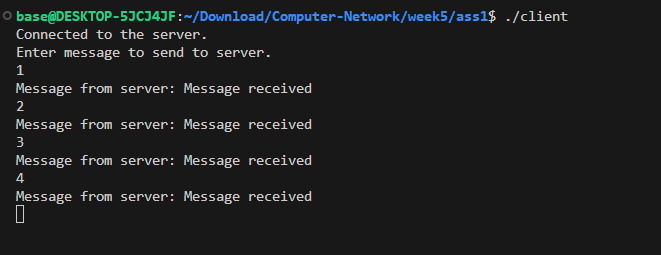
**Kết quả:**

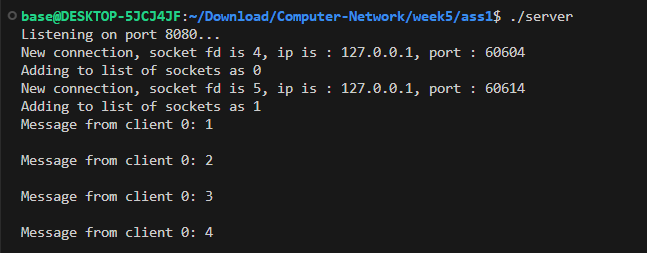
* Server có thể xử lý nhiều client, phát tin nhắn từ một client tới tất cả các

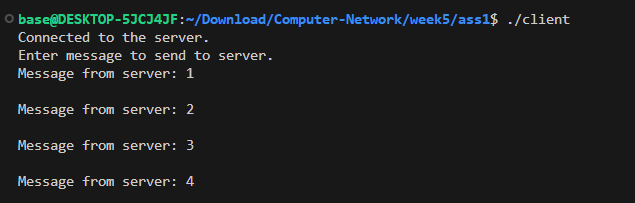
client khác trong phòng chat.

* Mỗi client có thể gửi tin nhắn và nhìn thấy tin nhắn của các client khác trong

thời gian thực.



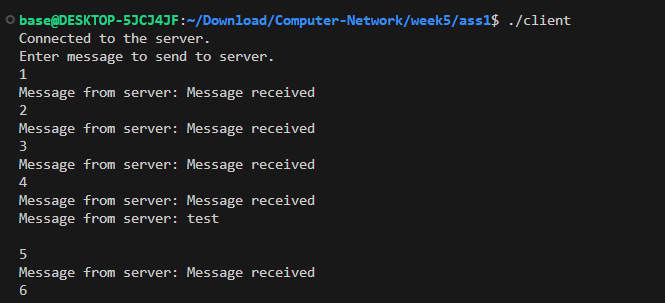


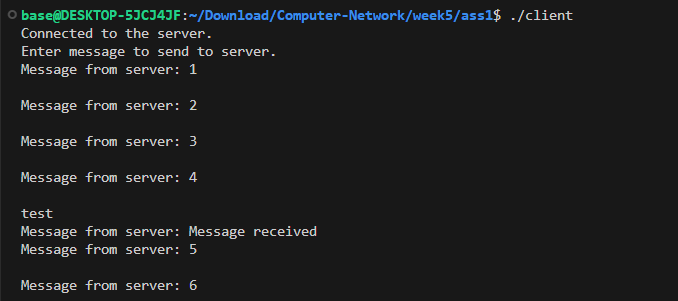


# **Phần 2: Logic Server và Phát Tin Nhắn**

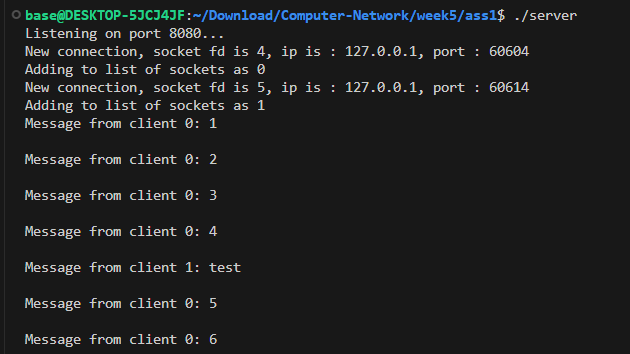
## 2.1: Phát Tin Nhắn Đến Tất Cả Các Client

**Client:**



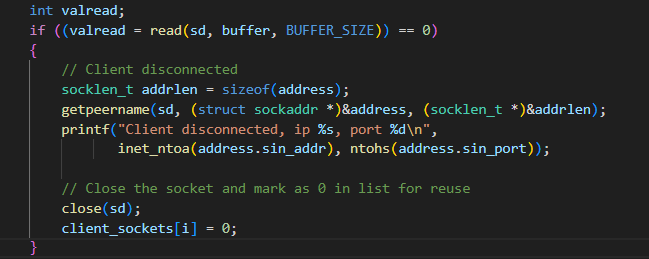


**Server:**

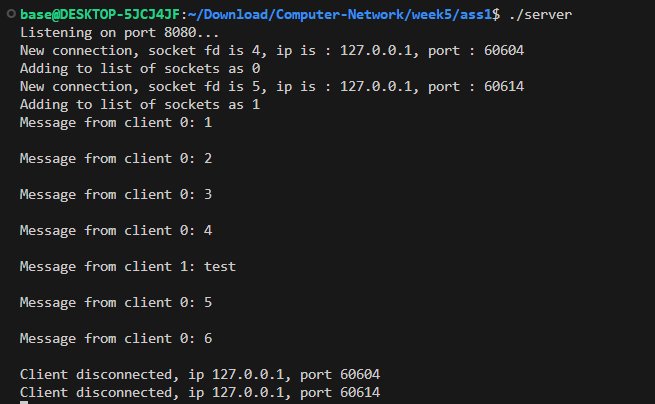


## 2.2: Xử Lý Client Ngắt Kết Nối

**Xử lý khi có ngắt kết nối từ client:**



**Kết quả:**

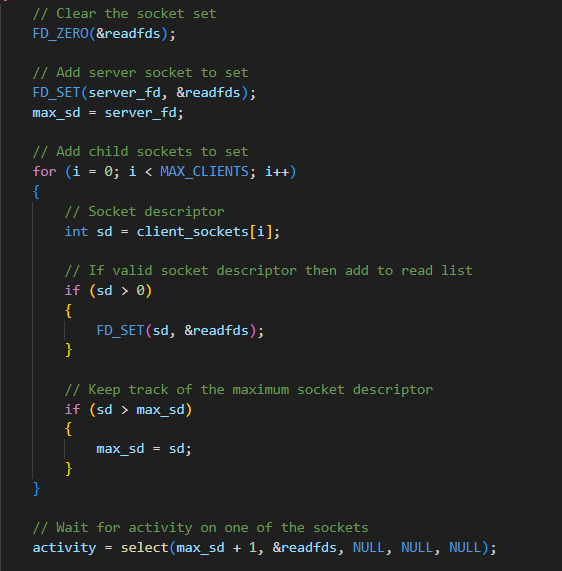


# 

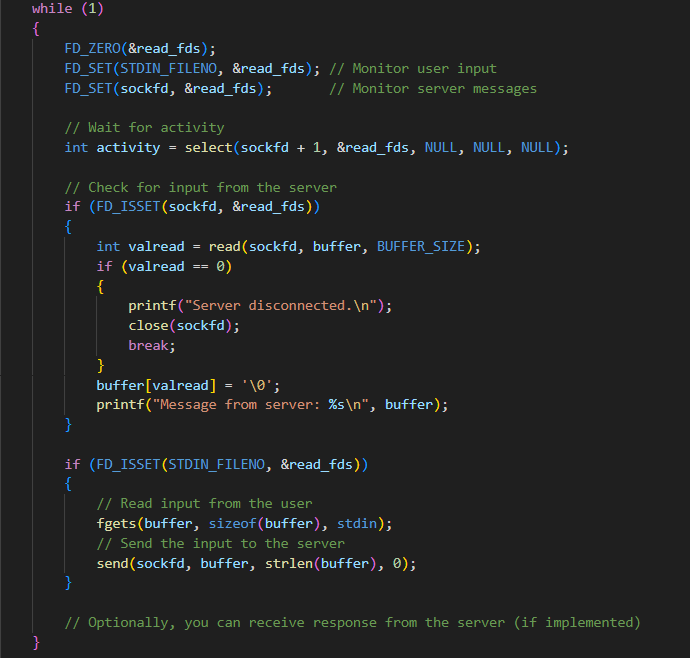
# **Phần 3: I/O Multiplexing để Xử Lý Nhiều Client**

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

**Server:**



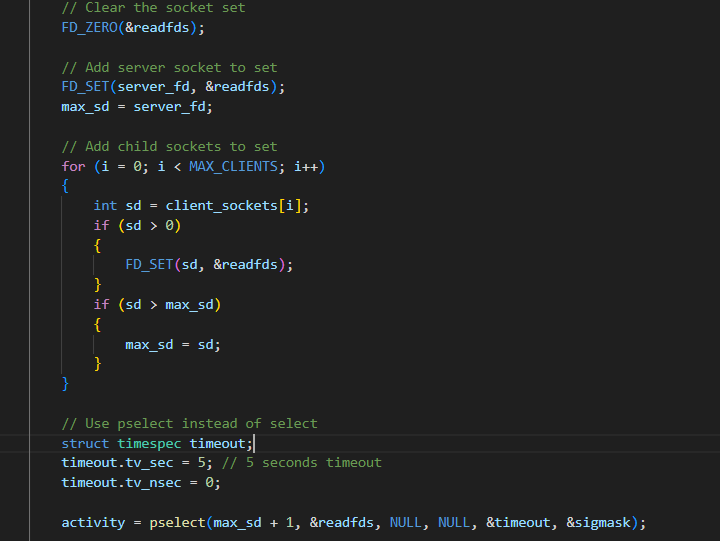
**Client:**

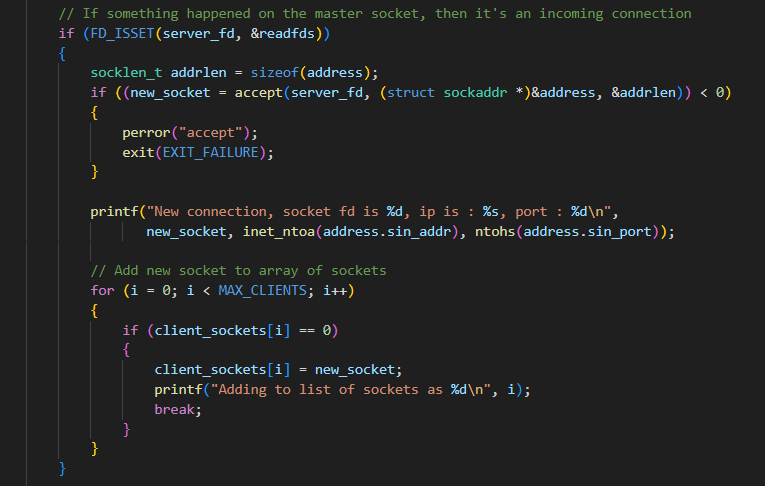
****

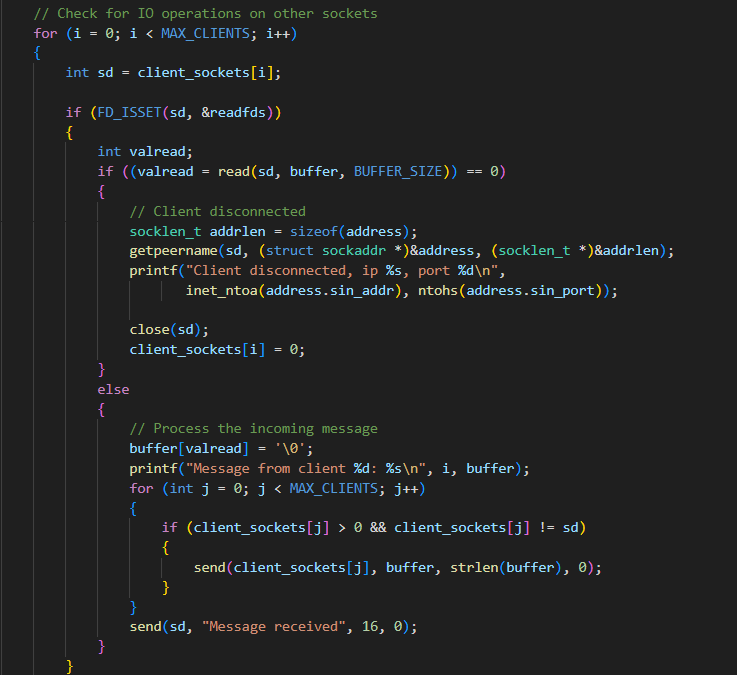
## 

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

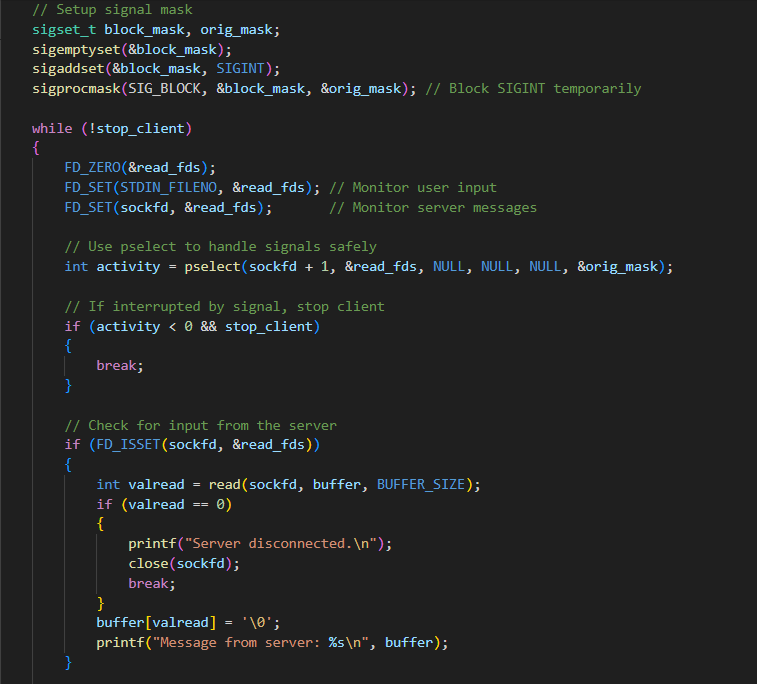
**Server:**

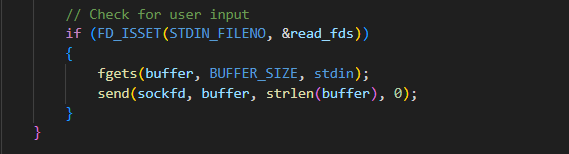




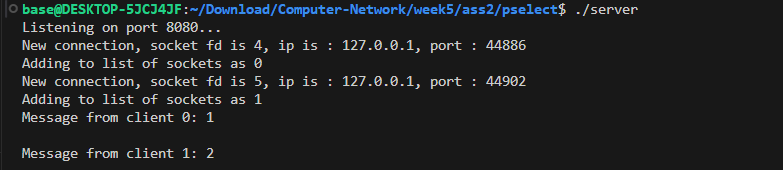


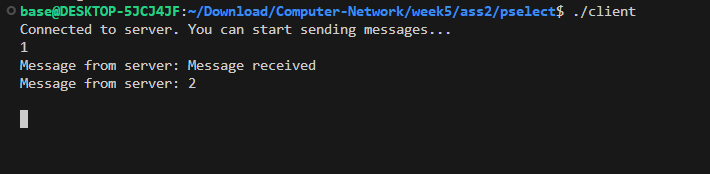
**Client:**

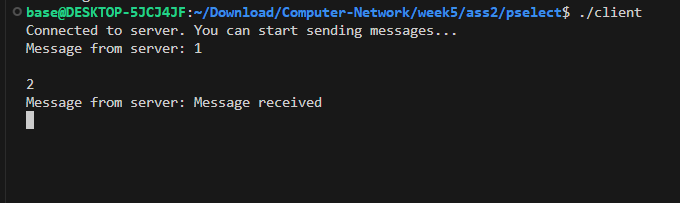




Kết quả:

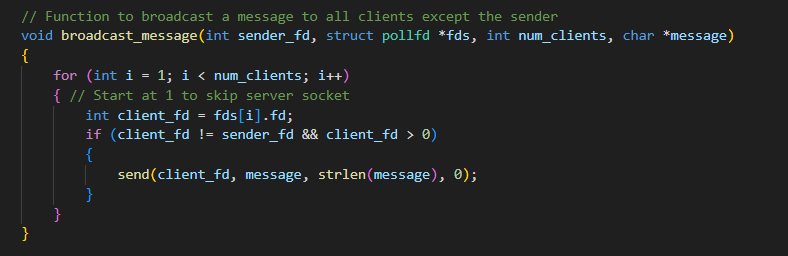


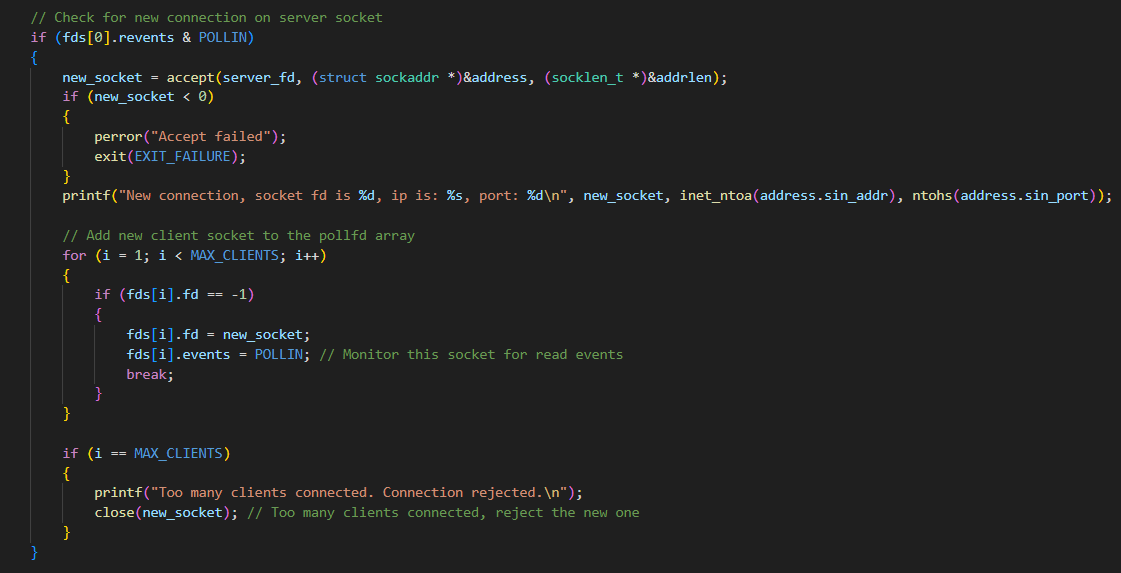




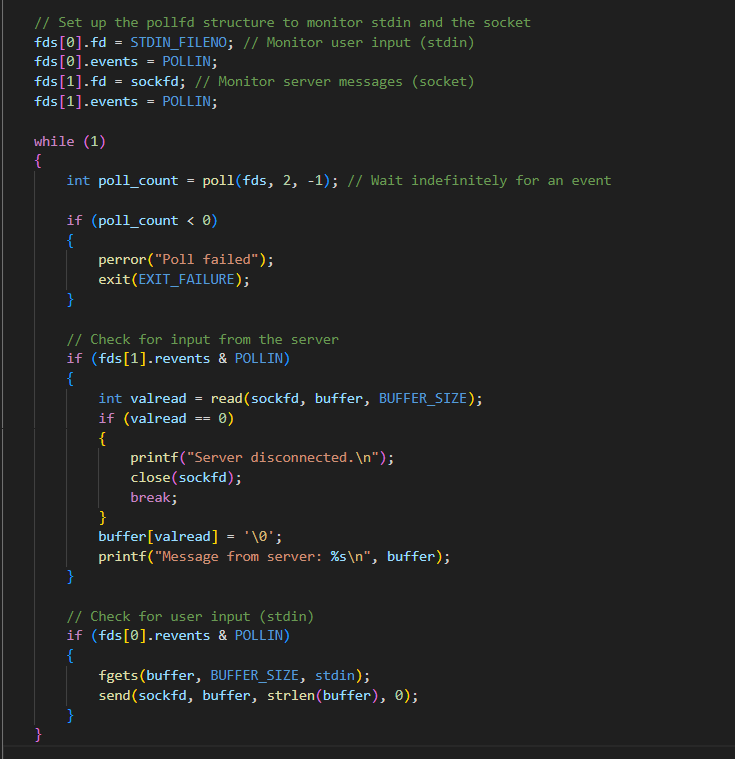
## 3.3: Sử Dụng poll() để Xử Lý Nhiều Client

Server:





Client:



## 3.4 Cách mà các cơ chế I/O multiplexing (select(), pselect(), poll())

## được sử dụng để quản lý nhiều client.

### **1. select()**

* **Cách hoạt động**: select() cho phép tiến trình theo dõi nhiều file descriptor (socket, file, stdin...) để xem khi nào chúng sẵn sàng cho các hoạt động I/O (đọc, ghi, hay xảy ra lỗi).
  + Bạn tạo ba tập hợp các file descriptor (fd\_set) cho các thao tác đọc, ghi và lỗi.
  + select() sẽ chặn (block) cho đến khi ít nhất một file descriptor trong tập hợp có sẵn.
  + Sau đó, bạn sẽ kiểm tra tập hợp để biết socket nào có dữ liệu cần xử lý.
* **Ưu điểm**: Đơn giản, dễ triển khai.
* **Nhược điểm**: Giới hạn số lượng file descriptor mà nó có thể quản lý (thường là 1024 trên hệ thống Unix).

### **2. pselect()**

* **Cách hoạt động**: pselect() tương tự như select(), nhưng nó cho phép bạn quản lý thêm một **mask tín hiệu** (signal mask), nghĩa là bạn có thể tạm thời thay đổi các tín hiệu được tiến trình xử lý trong khi chờ đợi I/O.
  + Lợi ích của việc này là giúp bạn tránh được race condition khi làm việc với tín hiệu (signal) và I/O.
* **Ưu điểm**: Bổ sung thêm khả năng xử lý tín hiệu cùng với I/O multiplexing.
* **Nhược điểm**: Giống như select(), bị giới hạn bởi số lượng file descriptor.

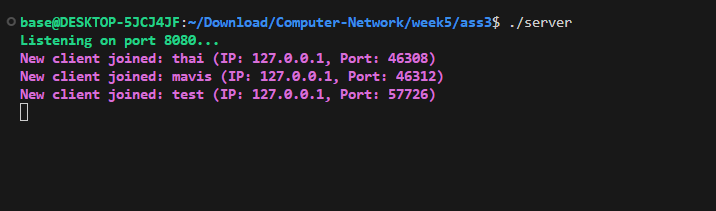
### **3. poll()**

* **Cách hoạt động**: poll() không có giới hạn cố định về số lượng file descriptor mà nó có thể quản lý. Bạn cung cấp một mảng các pollfd cấu trúc, mỗi cái đại diện cho một socket hoặc file descriptor và trạng thái mà bạn muốn giám sát (đọc, ghi, lỗi).
  + poll() sẽ chặn cho đến khi có ít nhất một file descriptor sẵn sàng, sau đó bạn kiểm tra từng phần tử của mảng pollfd để xem socket nào sẵn sàng cho thao tác I/O.
* **Ưu điểm**:
  + Không có giới hạn cứng về số lượng file descriptor.
  + Linh hoạt hơn select() khi quản lý một lượng lớn file descriptor.
* **Nhược điểm**:
  + Mặc dù không có giới hạn cứng về số lượng file descriptor, nhưng với số lượng lớn fd, việc kiểm tra sẽ gây ra độ trễ.

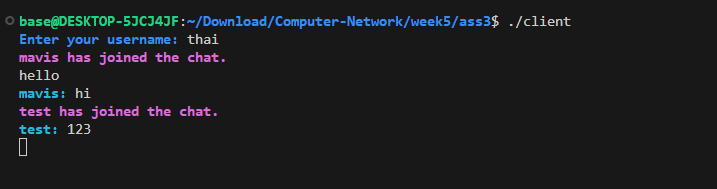
# **Phần 4: Tính Năng Phòng Chat và Trải Nghiệm Người Dùng**

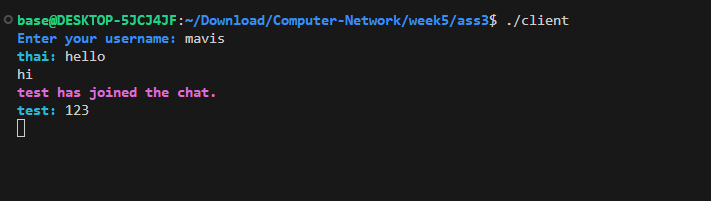
Hoạt động khi có 3 người trong phòng chat:

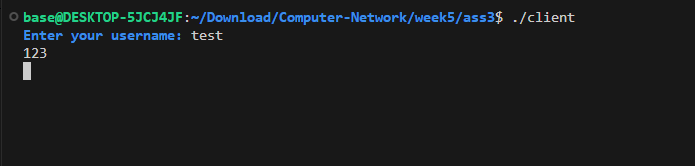
Server thông báo có người tham gia vào phòng chat và thông báo đến cho các thành viên còn lại:



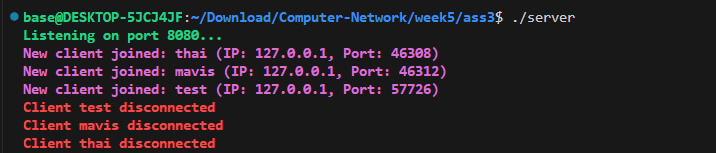
Client:

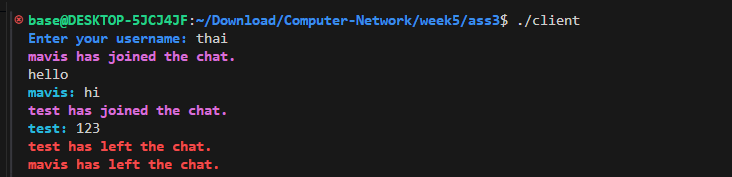


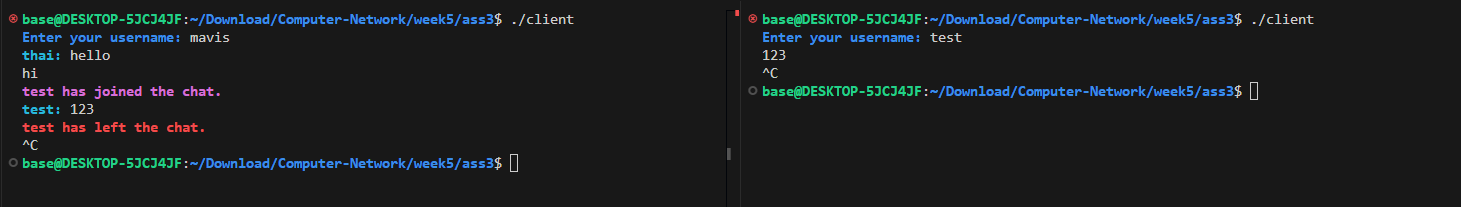




Xử lý khi thành viên dời đoạn chat:







## Cách chức năng phòng chat được triển khai

#### **1. Phát tin nhắn (broadcast)**

* Khi một client gửi tin nhắn, tin nhắn sẽ được server nhận qua socket. Sau đó, server thực hiện việc phát (broadcast) tin nhắn đó đến tất cả các client khác trong phòng.
* **Cách triển khai**:
  + **Client**:
    - Người dùng nhập tin nhắn qua stdin.
    - Tin nhắn được gửi đến server qua socket bằng hàm send().
  + **Server**:
    - Server sử dụng cơ chế I/O multiplexing như select() hoặc poll() để kiểm tra dữ liệu từ các socket client.
    - Khi nhận được tin nhắn từ một client, server sẽ sử dụng một vòng lặp để gửi tin nhắn đó đến tất cả các client khác thông qua hàm send().

#### **2. Tên người dùng**

* Khi một client kết nối đến server, server yêu cầu client cung cấp tên người dùng. Tên người dùng sẽ được lưu trữ tại server và đi kèm với các tin nhắn mà client gửi đi.
* **Cách triển khai**:
  + **Client**:
    - Khi kết nối đến server, client gửi tên người dùng đầu tiên qua socket.
  + **Server**:
    - Sau khi chấp nhận kết nối, server sẽ đọc tên người dùng từ socket và lưu tên đó vào một cấu trúc Client tương ứng.
    - Mỗi khi client gửi tin nhắn, server sẽ gắn tên người dùng vào tin nhắn trước khi phát cho các client khác.

#### **3. Phát thông báo khi có người dùng mới tham gia hoặc rời đi**

* Khi một người dùng tham gia phòng chat, server sẽ phát thông báo cho tất cả các client khác biết rằng người này đã tham gia.
* Khi một người dùng ngắt kết nối, server phát thông báo rằng người dùng này đã rời đi.
* **Cách triển khai**:
  + Khi một client mới kết nối, server nhận tên người dùng, sau đó phát một thông báo như "User X has joined the chat" đến tất cả các client khác.
  + Khi một client ngắt kết nối, server phát một thông báo như "User X has left the chat" đến các client còn lại.

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

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

* **Khó khăn**: Khi có nhiều client kết nối đến server, cần phải quản lý được tất cả các kết nối này mà không gây quá tải cho server. Server cần phải lắng nghe dữ liệu từ nhiều socket cùng lúc và phát tin nhắn đến đúng các client.
* **Cách khắc phục**: Sử dụng các cơ chế I/O multiplexing như **select()** hoặc **poll()** giúp server theo dõi nhiều socket client trong một luồng duy nhất. Điều này giúp server không phải tạo ra nhiều luồng con cho mỗi client (tiết kiệm tài nguyên hệ thống) và có thể xử lý nhiều kết nối cùng lúc một cách hiệu quả.

#### **2. Đồng bộ hóa việc phát tin nhắn**

* **Khó khăn**: Trong một hệ thống có nhiều client, nếu một client gửi quá nhiều tin nhắn liên tục, các client khác có thể gặp khó khăn trong việc nhận tin nhắn hoặc mất đồng bộ.
* **Cách khắc phục**:
  + Sử dụng các bộ đệm (buffer) lớn để đảm bảo rằng tin nhắn được lưu tạm thời trước khi được phát.
  + Kiểm tra và quản lý kích thước tin nhắn gửi từ mỗi client để tránh tình trạng quá tải mạng.
  + Triển khai các biện pháp kiểm soát lưu lượng để giới hạn số lượng tin nhắn mà mỗi client có thể gửi trong một khoảng thời gian.

#### **3. Xử lý ngắt kết nối đột ngột từ client**

* **Khó khăn**: Khi một client ngắt kết nối đột ngột mà không gửi tín hiệu nào, server có thể không phát hiện ngay lập tức. Điều này dẫn đến việc server vẫn giữ kết nối không còn hợp lệ, gây lãng phí tài nguyên.
* **Cách khắc phục**:
  + Sử dụng hàm **read()** hoặc **recv()** trên mỗi socket để phát hiện khi read() trả về giá trị 0 (nghĩa là client đã ngắt kết nối). Khi đó, server sẽ đóng socket tương ứng và phát thông báo ngắt kết nối đến các client còn lại.
  + Theo dõi trạng thái kết nối của từng client bằng cách sử dụng cơ chế timeout để kiểm tra xem client có gửi dữ liệu trong một khoảng thời gian nhất định không.

#### **4. Truyền tải tin nhắn lớn**

* **Khó khăn**: Nếu một client gửi tin nhắn có kích thước quá lớn (vượt quá kích thước bộ đệm), có thể gây ra lỗi hoặc mất một phần tin nhắn.
* **Cách khắc phục**:
  + Thiết lập kích thước bộ đệm hợp lý (ví dụ: BUFFER\_SIZE), và kiểm tra giới hạn kích thước tin nhắn từ client.
  + Chia nhỏ tin nhắn lớn thành các phần nhỏ hơn và gửi từng phần một nếu kích thước vượt quá giới hạn.

#### **5. Bảo mật**

* **Khó khăn**: Trong các ứng dụng phòng chat, bảo mật là vấn đề quan trọng để tránh việc rò rỉ thông tin hoặc bị tấn công từ bên ngoài.
* **Cách khắc phục**:
  + Sử dụng giao thức bảo mật như **SSL/TLS** để mã hóa dữ liệu giữa client và server.
  + Xác thực người dùng trước khi cho phép tham gia phòng chat, tránh những client không xác định truy cập.