Báo cáo Week 8 trên lớp - Thực hành Lập trình mạng - Kỳ 2025.1

Nguyễn Khánh Nam - 20225749

Github: https://github.com/nammannam/IT4062_Network_Programming

1) So sánh sự khác nhau giữa Asynchronous I/0 Model và signal-driven I/0 model

Đặc điểm	Signal-Driven I/O	Asynchronous I/O (AIO)
Cơ chế thông báo	Thông báo khi I/O operation có thể được bắt đầu.	Thông báo khi I/O operation đã hoàn thành trọn vẹn.
Giai đoạn Blocking	Có Blocking: Ứng dụng không bị chặn khi chờ dữ liệu, NHƯNG bị chặn (blocked) khi copy dữ liệu từ Kernel sang User space.	Không Blocking: Ứng dụng không bị chặn ở bất kỳ giai đoạn nào (kể cả lúc copy dữ liệu).
Trách nhiệm Copy Data	Ứng dụng (Process) phải gọi lệnh system call (như recvfrom) để tự copy dữ liệu.	Hệ điều hành (Kernel) tự động copy dữ liệu vào buffer của ứng dụng trong nền.
Quy trình hoạt động	**Thiết lập:** Ứng dụng thiết lập socket và đăng ký một bộ xử lý tín hiệu (signal handler). **Làm việc khác:** Ứng dụng tiếp tục chạy các tác vụ khác, không chờ đợi. **Thông báo:** Khi gói tin đến, Kernel gửi tín hiệu SIGIO cho ứng dụng. **Xử lý (Blocking phase):** Ứng dụng tạm dừng việc đang làm, chạy signal handler, và gọi hàm recvfrom. Lúc này, quá trình copy dữ liệu từ	**Yêu cầu:** Ứng dụng gọi hàm hệ thống (ví dụ aio_read), đưa cho Kernel địa chỉ buffer (bộ nhớ đệm) và yêu cầu đọc dữ liệu vào buffer. **Làm việc khác:** Ứng dụng tiếp tục chạy, hoàn toàn không quan tâm đến I/O nữa. **Kernel làm việc (Invisible phase):** Kernel chờ dữ liệu đến, **TỰ ĐỘNG** copy dữ liệu vào buffer mà ứng dụng đã đưa trước đó. **Thông báo:** Khi mọi thứ xong

	Kernel memory sang User memory diễn ra. **Trong lúc copy này, tiến trình bị block.** **Hoàn tất:** Sau khi copy xong, ứng dụng mới xử lý dữ liệu đó.	xuôi, Kernel gửi tín hiệu báo cho ứng dụng tự lấy dữ liệu trong buffer **Sử dụng:** Ứng dụng chỉ việc lấy dữ liệu trong buffer ra dùng (không tốn thời gian copy).

2) So sánh sự khác nhau giữa fast và slow system call?

Đặc điểm	Fast System Call	Slow System Call
Định nghĩa	Là các lời gọi hệ thống không bao giờ bị chặn vĩnh viễn.	Là các lời gọi hệ thống có khả năng bị chặn vĩnh viễn nếu sự kiện chờ đợi không xảy ra.
Trạng thái Block	Không block hoặc chỉ block trong thời gian xử lý phần cứng/kernel nội bộ.	Block cho đến khi có dữ liệu đến hoặc có sự kiện xảy ra.
Tác động của Signal	Thường không bị ngắt bởi Signal.	Bị ngắt (interrupted) bởi Signal khi đang chờ. Hàm sẽ trả về lỗi với errno = EINTR.

3) So sánh chi tiết lại điểm mạnh và điểm yếu giữa fork() và pthread _create(), nêu lên các trường hợp ứng dụng nên sử dụng fork() và pthread _create()

	fork()	pthread_create()
Điểm mạnh	Cách ly và Độ tin cậy cao: Một lỗi (crash) trong tiến trình con	Hiệu suất cao: Chi phí khởi tạo và chuyển đổi ngữ cảnh

	không làm hỏng tiến trình cha hoặc các tiến trình khác.	(context switching) cực kỳ thấp, giúp tận dụng tối đa CPU. -> Light weight
	Bảo mật: Dễ dàng áp dụng các chính sách bảo mật khác nhau (ví dụ: chroot, giảm quyền) cho từng tiến trình con độc lập.	Chia sẻ dữ liệu dễ dàng: Truy cập trực tiếp các biến chung (global) mà không cần cơ chế IPC (tập hợp các phương pháp và giao thức cho phép các tiến trình (process) khác nhau trên cùng một máy tính hoặc hệ thống máy tính phân tán trao đổi dữ liệu và phối hợp hoạt động). -> Chia sẻ dữ liệu giữa các luồng dễ dàng
	Dễ xử lý lỗi: Kernel tự động giải phóng tài nguyên khi một Process chết.	Tiết kiệm tài nguyên: Chia sẻ không gian bộ nhớ, tiết kiệm RAM.
Điểm yếu	Chi phí cao: Tốn thời gian tạo, tốn bộ nhớ RAM -> Heavy weight	Cần Đồng bộ hóa: Dễ xảy ra Race Condition, Deadlock do chia sẻ bộ nhớ. Yêu cầu lập trình viên phải sử dụng Mutexes, Semaphores để đồng bộ hóa.
	Giao tiếp phức tạp: Việc trao đổi dữ liệu giữa các tiến trình con tốn thời gian và phải qua API IPC phức tạp. -> Trao đổi từ cha xuống con thì dễ nhưng từ con lên cha thì khó.	Độ tin cậy thấp: Một lỗi truy cập bộ nhớ của một Luồng có thể làm sập toàn bộ ứng dụng.
	Quản lý tài nguyên: Khó kiểm soát việc quá tải tài nguyên hệ thống nếu quá nhiều Process được tạo ra.	Khó gỡ lỗi (Debugging): Việc theo dõi lỗi trong môi trường đa luồng, đặc biệt là lỗi đồng bộ hóa, rất khó khăn.
Sử dụng	**Web Server (Mô hình**	**Tính toán Song song (Parallel**

	Process-per-Client): ● Khi máy chủ nhận một kết nối mới, nó fork() ra một tiến trình con. ● **Lý do:** Nếu tiến trình con xử lý request bị lỗi (ví dụ: lỗi segmentation fault), tiến trình cha (Listener) vẫn hoạt động bình thường, đảm bảo tính liên tục của dịch vụ. (Ví dụ: Web server đời cũ hoặc các dịch vụ cần độ tin cậy cực cao). **Chạy các Chương trình bên ngoài (External Program Execution):** ● Dùng fork() sau đó gọi exec() để chạy một chương trình khác. Đây là cách chuẩn để khởi tạo một ứng dụng mới từ bên trong ứng dụng hiện tại. **Bảo mật và Tách biệt (Security & Isolation):** ● Các ứng dụng muốn giới hạn quyền của một phần code (ví dụ: sandbox, giảm quyền sau khi khởi tạo) sẽ fork() và thay đổi quyền của tiến trình con để tăng cường bảo mật.	**Computing):** ● Khi bạn muốn chia một tác vụ tính toán lớn (ví dụ: xử lý ma trận, render đồ họa) thành nhiều phần nhỏ để chạy đồng thời trên các nhân CPU khác nhau. ● **Lý do:** Các luồng chia sẻ bộ nhớ đầu vào/đầu ra, giúp tiết kiệm thời gian truyền dữ liệu (thay vì IPC). **Ứng dụng có Giao diện người dùng (Responsive UI):** ● Một luồng chính xử lý UI, các luồng khác xử lý các tác vụ dài (ví dụ: tải file, truy vấn cơ sở dữ liệu) ở chế độ nền. ● **Lý do:** Ngăn UI bị đóng băng (freezing), giúp ứng dụng phản hồi nhanh hơn với người dùng. **Web Server hiện đại (Worker Thread Pool):** ● Máy chủ duy trì một pool (nhóm) các luồng làm việc. Khi có request đến, nó được gán cho một luồng rỗi. ● **Lý do:** Chi phí tạo Luồng thấp hơn nhiều so với tạo Process, giúp xử lý đồng thời lượng lớn request

	Tác vụ cần tính toán độc lập, không cần chia sẻ dữ liệu nhiều: ● Ví dụ: Chạy các thuật toán khác nhau hoặc các module hoàn toàn độc lập.	một cách hiệu quả về tài nguyên.

4) Trình bày chi tiết interlock, Semaphore, và Mutex

1. Interlock (Cơ chế Khóa/Khoá liên động)

Khái niệm **Interlock** không phải là một cấu trúc dữ liệu cụ thể như Mutex hay Semaphore, mà thường được sử dụng để chỉ **nguyên tắc** hoặc **cơ chế ở cấp độ thấp** nhằm đảm bảo chỉ có một hoạt động có thể xảy ra tại một thời điểm, qua đó ngăn chặn sự xung đột.

Nguyên tắc

- Interlock đảm bảo rằng khi một tác vụ đang thực thi một đoạn mã quan trọng (Critical Section), các tác vụ khác phải bị khóa (locked out) hoặc bị chặn (blocked) cho đến khi tác vụ đó hoàn thành.

Cơ chế (Atomic Operations)

Trong lập trình cấp thấp, Interlock được thực hiện bằng các **Thao tác Nguyên tử (Atomic Operations)** được hỗ trợ trực tiếp bởi phần cứng (CPU). Các thao tác này là cơ sở để xây dựng Mutex và Semaphore:

- **Test-and-Set:** Kiểm tra một cờ và đặt nó thành TRUE (khóa) trong cùng một chu kỳ bộ nhớ không thể bị ngắt (uninterruptible).
- **Compare-and-Swap (CAS):** So sánh giá trị của một vị trí bộ nhớ với một giá trị dự kiến; nếu chúng khớp, nó sẽ ghi một giá trị mới vào vị trí đó, tất cả đều thực hiện nguyên tử.

2. Mutex (Mutual Exclusion - Độc quyền Tương hỗ)

Mutex là một cơ chế khóa (locking mechanism) đơn giản và phổ biến nhất, được thiết kế để đảm bảo rằng **chỉ một luồng (thread)** tại một thời điểm được phép truy cập vào một **vùng dữ liệu hoặc đoạn mã quan trọng (Critical Section)**.

Cơ chế hoạt động

- **Khóa (Acquire/Lock):** Một luồng muốn truy cập Critical Section phải gọi hàm lock() (hoặc wait()). Nếu Mutex đang mở, luồng đó sẽ khóa Mutex và đi vào Critical Section.
- **Mở khóa (Release/Unlock):** Sau khi hoàn thành công việc, luồng đó phải gọi hàm unlock() (hoặc signal()) để giải phóng Mutex, cho phép luồng khác có thể truy cập.

Đặc điểm quan trọng

- **Độc quyền sở hữu (Ownership):** Chỉ có luồng **đã khóa** Mutex mới có quyền **mở khóa** nó. Điều này giúp ngăn chặn lỗi lập trình nghiêm trọng.
- **Binary State:** Mutex chỉ có hai trạng thái: **Khóa** (locked) hoặc **Mở** (unlocked).
- **Ứng dụng:** Chủ yếu dùng để **bảo vệ dữ liệu chia sẻ** (ví dụ: một biến toàn cục, một đối tượng phức tạp) khỏi việc bị nhiều luồng sửa đổi cùng lúc.

3. Semaphore (Cấu trúc Tín hiệu)

Semaphore là một cấu trúc dữ liệu cơ bản hơn, hoạt động như một **bộ đếm (counter)**, được sử dụng để **kiểm soát quyền truy cập** vào một **tập hợp tài nguyên có giới hạn** hoặc để **điều phối/ra tín hiệu** giữa các luồng.

Các loại Semaphore

1. **Counting Semaphore (Semaphore Đếm):**
    - Cho phép N luồng truy cập tài nguyên cùng lúc, với N>1.
    - Ứng dụng: Kiểm soát số lượng kết nối tối đa đến một cơ sở dữ liệu (ví dụ: chỉ cho phép 100 luồng truy cập cùng lúc).
2. **Binary Semaphore (Semaphore Nhị phân):**
    - Giá trị tối đa là N=1. Nó hoạt động giống hệt như một Mutex.
    - Ứng dụng: Dùng để bảo vệ Critical Section hoặc làm tín hiệu đơn giản (ví dụ: báo hiệu một sự kiện đã xảy ra).

Cơ chế hoạt động

Semaphore được thao tác thông qua hai hàm:

Thao tác	Tên gọi phổ biến	Mô tả
P (Wait/Decrement)	wait() hoặc acquire()	Giảm giá trị của bộ đếm. Nếu bộ đếm ≤0 (tài nguyên đã hết), luồng bị chặn (block) cho đến khi bộ đếm tăng lên.
V (Signal/Increment)	signal() hoặc release()	Tăng giá trị của bộ đếm. Nếu có luồng nào đang bị chặn, một trong số chúng sẽ được giải phóng.

File account.txt:

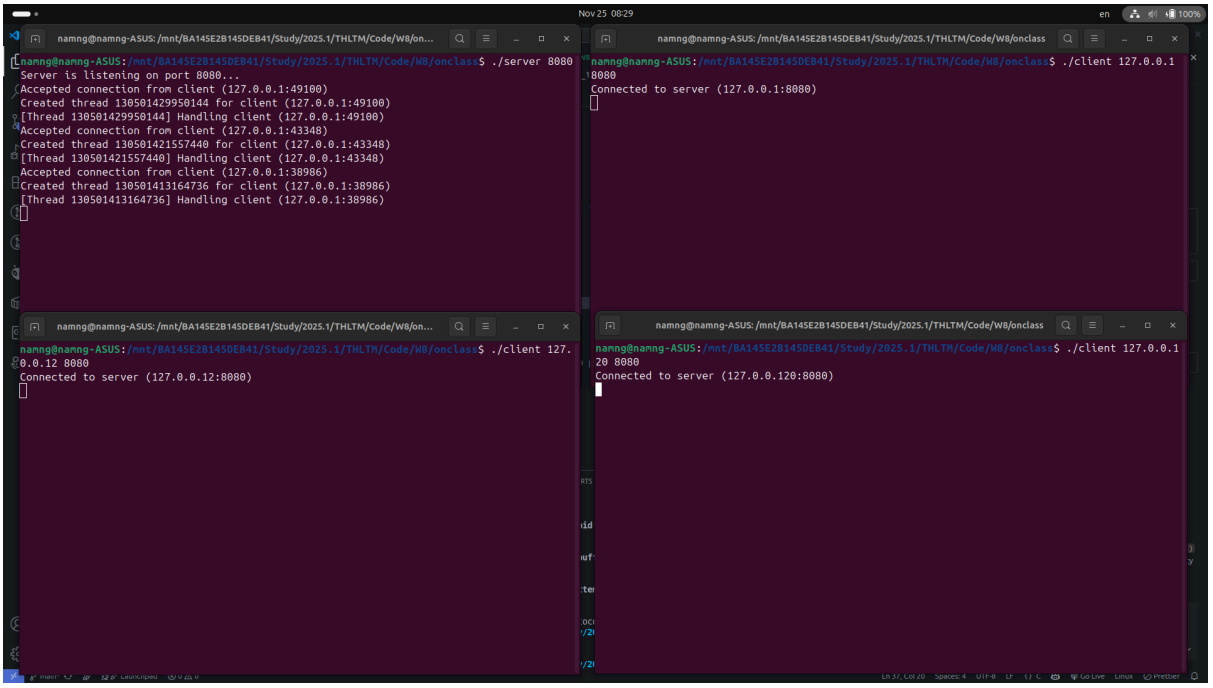
server.c | server_b2.c W4 | client_b2.c W4 | server_b2.c W5 | client_b2.c W5 | account.txt W5 M

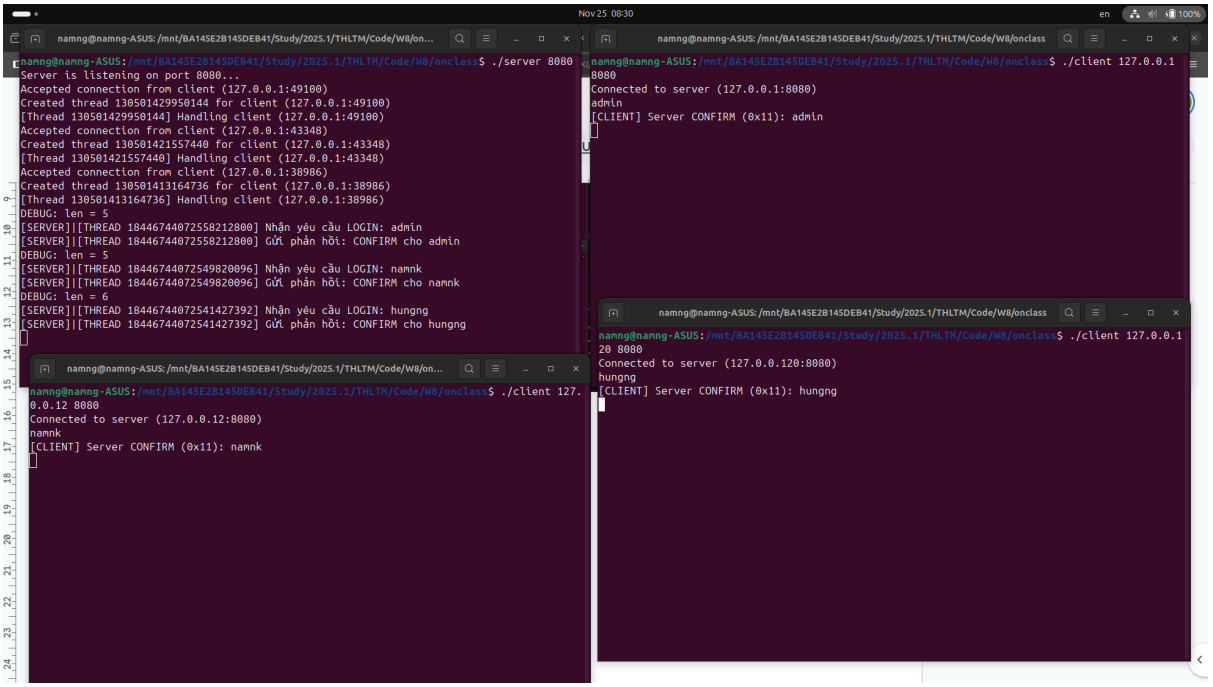
W5 > account.txt

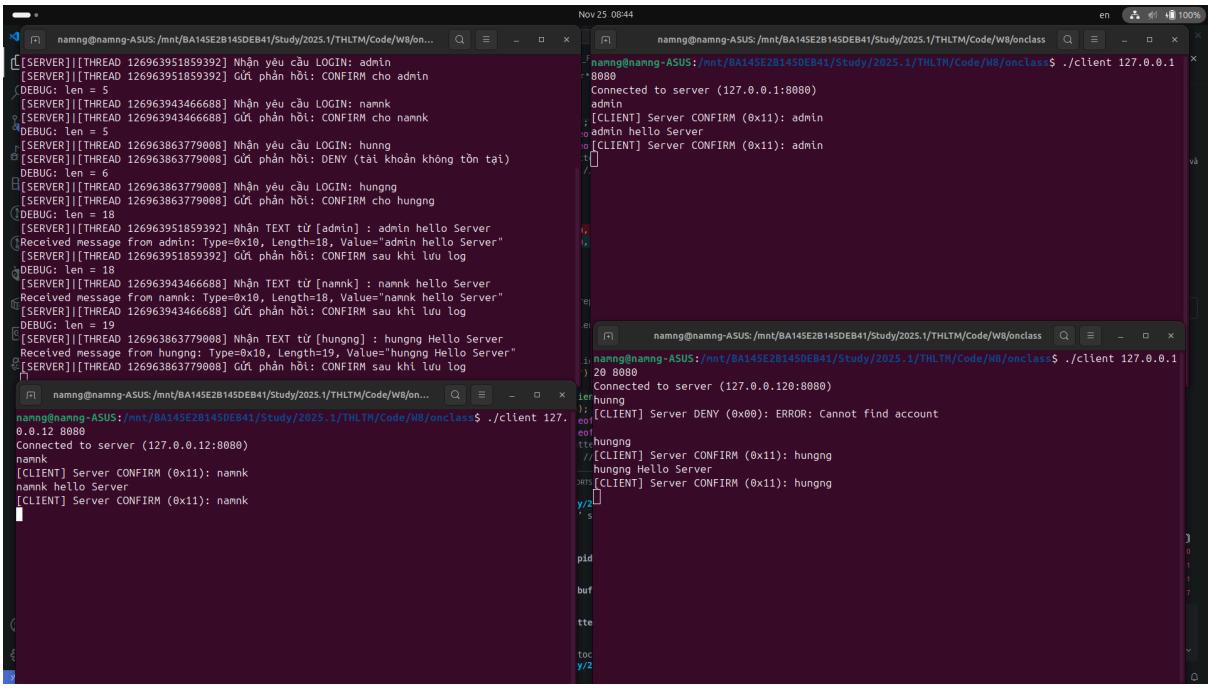
You, 30 seconds ago | 1 author (You)

```
admin admin123 admin@gmail.com google.com 0
namnk namnk123 namnk@gmail.com youtube.com 0
hungng hung2004 hungng@hust.edu soict.hust.edu.vn 1
locdinh loc2004 123@gmail.com daotao.ai 1
quangvn vnquang12 quangvn22@gmail.com youtube.com 1
test2 test123 test2new@gmail.com youtube.com 1
```

Kết quả: **Mô hình 1 Server - 3 Clients**




namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./server 8080
Server is listening on port 8080...
Accepted connection from client (127.0.0.1:49100)
Created thread 130501429950144 for client (127.0.0.1:49100)
[Thread 130501429950144] Handling client (127.0.0.1:49100)
Accepted connection from client (127.0.0.1:43348)
Created thread 130501421557440 for client (127.0.0.1:43348)
[Thread 130501421557440] Handling client (127.0.0.1:43348)
Accepted connection from client (127.0.0.1:38986)
Created thread 130501413164736 for client (127.0.0.1:38986)
[Thread 130501413164736] Handling client (127.0.0.1:38986)
DEBUG: len = 5
[SERVER]|[THREAD 18446744072558212800] Nhận yêu cầu LOGIN: admin
[SERVER]|[THREAD 18446744072558212800] Gửi phản hồi: CONFIRM cho admin
DEBUG: len = 5
[SERVER]|[THREAD 18446744072549820096] Nhận yêu cầu LOGIN: namnk
[SERVER]|[THREAD 18446744072549820096] Gửi phản hồi: CONFIRM cho namnk
DEBUG: len = 6
[SERVER]|[THREAD 18446744072541427392] Nhận yêu cầu LOGIN: hungng
[SERVER]|[THREAD 18446744072541427392] Gửi phản hồi: CONFIRM cho hungng
namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./client 127.0.0.1
8080
Connected to server (127.0.0.1:8080)
admin
[CLIENT] Server CONFIRM (0x11): admin
namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./client 127.0.0.12 8080
Connected to server (127.0.0.12:8080)
namnk
[CLIENT] Server CONFIRM (0x11): namnk
namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./client 127.0.0.120 8080
Connected to server (127.0.0.120:8080)
hungng
[CLIENT] Server CONFIRM (0x11): hungng


[SERVER]|[THREAD 126963951859392] Nhận yêu cầu LOGIN: admin
[SERVER]|[THREAD 126963951859392] Gửi phản hồi: CONFIRM cho admin
DEBUG: len = 5
[SERVER]|[THREAD 126963943466688] Nhận yêu cầu LOGIN: namnk
[SERVER]|[THREAD 126963943466688] Gửi phản hồi: CONFIRM cho namnk
DEBUG: len = 5
[SERVER]|[THREAD 126963863779008] Nhận yêu cầu LOGIN: hunng
[SERVER]|[THREAD 126963863779008] Gửi phản hồi: DENY (tài khoản không tồn tại)
DEBUG: len = 6
[SERVER]|[THREAD 126963863779008] Nhận yêu cầu LOGIN: hungng
[SERVER]|[THREAD 126963863779008] Gửi phản hồi: CONFIRM cho hungng
DEBUG: len = 18
[SERVER]|[THREAD 126963951859392] Nhận TEXT từ [admin] : admin hello Server
Received message from admin: Type=0x10, Length=18, Value="admin hello Server"
[SERVER]|[THREAD 126963951859392] Gửi phản hồi: CONFIRM sau khi lưu log
DEBUG: len = 18
[SERVER]|[THREAD 126963943466688] Nhận TEXT từ [namnk] : namnk hello Server
Received message from namnk: Type=0x10, Length=18, Value="namnk hello Server"
[SERVER]|[THREAD 126963943466688] Gửi phản hồi: CONFIRM sau khi lưu log
DEBUG: len = 19
[SERVER]|[THREAD 126963863779008] Nhận TEXT từ [hungng] : hungng Hello Server
Received message from hungng: Type=0x10, Length=19, Value="hungng Hello Server"
[SERVER]|[THREAD 126963863779008] Gửi phản hồi: CONFIRM sau khi lưu log
namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./client 127.0.0.1
8080
Connected to server (127.0.0.1:8080)
admin
[CLIENT] Server CONFIRM (0x11): admin
admin hello Server
[CLIENT] Server CONFIRM (0x11): admin
namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./client 127.0.0.12 8080
Connected to server (127.0.0.12:8080)
namnk
[CLIENT] Server CONFIRM (0x11): namnk
namnk hello Server
[CLIENT] Server CONFIRM (0x11): namnk
namng@namng-ASUS:/mnt/BA145E2B145DEB41/Study/2025.1/THLTM/Code/W8/onclass$ ./client 127.0.0.120 8080
Connected to server (127.0.0.120:8080)
hunng
[CLIENT] Server DENY (0x00): ERROR: Cannot find account
hungng
[CLIENT] Server CONFIRM (0x11): hungng
hungng Hello Server
[CLIENT] Server CONFIRM (0x11): hungng

- **File log user.txt:**