**BÁO CÁO THỰC HÀNH LẬP TRÌNH MẠNG – TUẦN 05**

**BÀI TẬP TRÊN LỚP**

**Họ và tên:** Đoàn Nhật Quang

**MSSV:** 20225911

**Lớp:** 161266

1. **UDP**
2. **Trình bày rõ hơn ý nghĩa và chi tiết từng cờ trong lệnh recvfrom(), sendto()**

Các cờ (flags) được sử dụng để điều khiển cách hoạt động của các hàm recvfrom() và sendto(). Nhiều cờ có thể kết hợp với nhau bằng toán tử bitwise OR.

- Đối với hàm recvfrom():

+ MSG\_PEEK: Cho phép “xem trước” dữ liệu đến mà không xóa nó khỏi hàng đợi. Nghĩa là, lần gọi recvfrom() tiếp theo vẫn nhận lại cùng dữ liệu đó.

+ MSG\_OOB: Dùng để nhận dữ liệu ngoài luồng (out-of-band data).

+ MSG\_WAITALL: Khi dùng với socket kiểu SOCK\_STREAM, cờ này buộc hàm phải chờ cho đến khi nhận đủ lượng dữ liệu yêu cầu mới trả kết quả.

- Đối với hàm sendto():

+ MSG\_OOB: Dùng để gửi dữ liệu ngoài luồng trên các socket có hỗ trợ tính năng này.

+ MSG\_DONTROUTE: Yêu cầu không dùng gateway để truyền gói tin, chỉ gửi đến các máy chủ nằm trực tiếp trong cùng mạng kết nối.

1. **EMSGSIZE trả về trong sendto() ý nghĩa là gì**

Khi hàm sendto() trả về lỗi EMSGSIZE, điều đó cho biết datagram mà ứng dụng muốn gửi vượt quá kích thước bộ đệm gửi (send buffer) của socket — tức là kích thước gói tin quá lớn so với giới hạn mà hệ thống cho phép trong bộ đệm gửi.

1. **Trường hợp nào nên dùng UDP và tại sao**

Nên sử dụng UDP trong những trường hợp không cần độ tin cậy cao và không yêu cầu cơ chế kiểm soát luồng dữ liệu, vì UDP là giao thức không đảm bảo độ tin cậy (unreliable) và không có kiểm soát luồng (no flow control).

Ví dụ: UDP thường được dùng trong các ứng dụng như DNS, truyền phát (streaming), hoặc các dịch vụ thời gian thực nơi tốc độ quan trọng hơn tính chính xác tuyệt đối của dữ liệu

1. **Miêu tả quy trình hoạt động của UDP client và UDP server trong việc sử dụng UDP socket**

Phía Server:

* socket(): Tạo một socket UDP.
* bind(): Gán địa chỉ IP và cổng cho socket để server có thể nhận dữ liệu.
* recvfrom(): Chờ (block) cho đến khi nhận được dữ liệu từ client.
* Xử lý yêu cầu: Thực hiện các thao tác cần thiết dựa trên dữ liệu nhận được.
* sendto(): Gửi gói tin phản hồi lại cho client.

Phía Client:

* socket(): Tạo một socket UDP.
* bind() *(tuỳ chọn)*: Gán địa chỉ cho socket nếu cần (thường có thể bỏ qua, hệ thống sẽ tự chọn).
* sendto(): Gửi dữ liệu yêu cầu (request) đến server; hàm có thể block cho đến khi dữ liệu được gửi đi.
* recvfrom(): Chờ (block) để nhận phản hồi (reply) từ server.

1. **TCP**
2. **Trình bày sự khác nhau giữa recvfrom() với recv(), sendto() với send()**

Sự khác biệt chủ yếu giữa các cặp hàm này nằm ở việc có hay không có kết nối trước khi truyền dữ liệu, và do đó có cần tham số địa chỉ hay không:

- recv() và send():

+ Được sử dụng với socket đã được kết nối (connected socket).

+ Vì kết nối giữa hai host đã được thiết lập trước nên không cần truyền thêm tham số địa chỉ trong mỗi lần gửi hoặc nhận dữ liệu.

- recvfrom() và sendto():

+ Dùng cho socket không kết nối (connectionless), điển hình là UDP.

+ Các hàm này cần tham số địa chỉ để xác định nguồn gửi hoặc đích nhận.

+ Đây là kiểu giao tiếp không cần thiết lập kết nối trước và không đảm bảo độ tin cậy trong việc truyền dữ liệu.

1. **Trình bày rõ hơn ý nghĩa và chi tiết cờ trong lệnh recv() và send()**

Có thể kết hợp nhiều cờ (flags) bằng toán tử bitwise OR (|) để điều chỉnh hành vi của các hàm send() và recv().

Các cờ dành cho send():

* MSG\_OOB: Gửi dữ liệu ngoài luồng (out-of-band). Bên nhận sẽ nhận được tín hiệu SIGURG và có thể xử lý dữ liệu này mà không cần đọc hết dữ liệu thông thường trong hàng đợi.
* MSG\_DONTROUTE: Chỉ gửi dữ liệu trong mạng nội bộ, không chuyển qua router.
* MSG\_DONTWAIT: Nếu thao tác gửi có thể bị block, hàm sẽ trả về lỗi EAGAIN ngay lập tức thay vì chờ.
* MSG\_NOSIGNAL: Ngăn tín hiệu SIGPIPE được gửi đến tiến trình khi cố gửi dữ liệu đến host đã ngắt kết nối.

Các cờ dành cho recv():

* MSG\_PEEK: Cho phép xem trước dữ liệu mà không xóa nó khỏi hàng đợi; dữ liệu có thể được đọc lại trong lần gọi recv() kế tiếp.
* MSG\_OOB: Dùng để nhận dữ liệu ngoài luồng (out-of-band).
* MSG\_WAITALL: Buộc hàm phải block cho đến khi nhận đủ số byte yêu cầu, trừ khi kết nối bị đóng, có lỗi, hoặc có tín hiệu bị ngắt.

1. **Miêu tả rõ hơn slide 18**

Slide mô tả quá trình **xử lý kết nối (Process connections)** trong TCP, hay còn gọi là bắt tay ba bước.

*1. Server khởi tạo và chờ kết nối:*

Server gọi hàm listen(), chuyển socket sang trạng thái LISTENING để chờ yêu cầu kết nối từ Client.

*2. Client gửi yêu cầu kết nối:*

Client gọi connect(), tạo và gửi một gói tin SYN đến Server — yêu cầu thiết lập kết nối.

*3. Server phản hồi yêu cầu:*

Khi nhận được gói SYN, Server chuyển sang trạng thái SYN\_RECEIVED.

Server gửi ngược lại cho Client một gói SYN/ACK nhằm xác nhận đã nhận yêu cầu và đồng thời gửi lại yêu cầu thiết lập kết nối của chính nó.

*4. Client xác nhận lại:*

Client nhận được SYN/ACK, sau đó gửi gói ACK cuối cùng để xác nhận đã nhận phản hồi từ Server.

Khi đó, hàm connect() của Client trả về, báo hiệu kết nối đã được thiết lập thành công.

*5. Server hoàn tất kết nối:*

Khi Server nhận được gói ACK, nó chuyển sang trạng thái ESTABLISHED.

Kết nối mới được thêm vào hàng đợi kết nối (new connection queue).

*6. Server chấp nhận kết nối:*

Cuối cùng, Server gọi hàm accept() để lấy kết nối hoàn chỉnh ra khỏi hàng đợi,

sau đó bắt đầu quá trình trao đổi dữ liệu với Client.

1. **Trường hợp nào nên dùng TCP và tại sao**

Nên sử dụng TCP trong các trường hợp cần đảm bảo độ tin cậy và kiểm soát chặt chẽ quá trình truyền dữ liệu, vì TCP cung cấp cơ chế truyền thông đáng tin cậy (reliable communication) và kiểm soát tốc độ truyền dữ liệu (data rate control) giữa hai đầu kết nối.

Ví dụ: TCP thường được dùng trong các ứng dụng như Email, Web (HTTP/HTTPS), truyền tải hình ảnh (Image transfer), hoặc bất kỳ dịch vụ nào yêu cầu dữ liệu phải đến đầy đủ và theo đúng thứ tự.

1. **Miêu tả sử dụng quy trình hoạt động của TCP client và TCP server trong việc sử dụng TCP socket**

Phía Server:

* socket() – Tạo một socket TCP.
* bind() – Gán địa chỉ IP và cổng cho socket.
* listen() – Đưa socket vào chế độ lắng nghe (LISTEN) để chờ yêu cầu kết nối từ client.
* accept() – Chấp nhận một yêu cầu kết nối đến và tạo một socket mới dành riêng cho kết nối đó.
* recv() / send() – Thực hiện truyền và nhận dữ liệu với client.
* close() – Đóng kết nối với client sau khi hoàn tất giao tiếp.
* (Lặp lại các bước từ accept() đến close() để xử lý các kết nối mới.)

Phía Client:

* socket() – Tạo một socket TCP.
* connect() – Gửi yêu cầu kết nối đến server.
* send() / recv() – Thực hiện giao tiếp: gửi và nhận dữ liệu với server.
* close() – Đóng kết nối sau khi hoàn thành trao đổi dữ liệu.