

Ồ CẨM TIỂU HỌC

Tổng Văn Vân, SoICT, ĐHBKHN

Nội dung

- `socket()`
- UDP Socket API
- API ổ cắm TCP
- Máy chủ TCP lặp lại
- Thiết kế giao thức ứng dụng

Ổ cắm()

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int giao thức);
```

- Tạo điểm cuối cho giao tiếp • Miền [IN]: AF_INET, AF_INET6 hoặc AF_UNSPEC, .
- Đối số loại [IN] có thể là:
 - SOCK_STREAM: Cung cấp các luồng byte dựa trên kết nối, theo trình tự, đáng tin cậy, hai chiều
 - SOCK_DGRAM: Hỗ trợ datagram
 - SOCK_RAW: Cung cấp quyền truy cập giao thức mạng thô
- Giao thức [IN] thường là 0
- Trả về giá trị
 - Một bộ mô tả ổ cắm mới mà bạn có thể sử dụng để thực hiện những việc liên quan đến ổ cắm
 - Nếu xảy ra lỗi, hãy trả về -1 (nhớ errno)

tróí buộc()

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

- Liên kết ổ cắm với địa chỉ IP và số cổng • Ở đâu
 - [IN] sockfd : bộ mô tả socket • [IN]
 - addr : con trỏ tới cấu trúc sockaddr được gán cho sockfd • [IN] addrlen : chỉ định kích thước, tính bằng byte của cấu trúc địa chỉ được trỏ tới bởi addr
- Giá trị trả về
 - Trả về 0 nếu không có lỗi xảy ra.
 - Ngược lại, trả về -1 (và errno sẽ được đặt tương ứng)

tắt()

```
#include <sys/socket.h>  
int shutdown(int socket, int how);
```

- Tắt các hoạt động gửi và nhận ổ cắm • Ở đâu
 - [IN] sockfd: bộ mô tả xác định ổ cắm. • [IN] cách:
SHUT_RD, SHUT_WR, SHUT_RDWR
- Giá trị trả về
 - Trả về 0 nếu không có lỗi xảy ra.
 - Ngược lại, trả về -1 (và errno sẽ được đặt tương ứng)

gần()

```
#include <unistd.h>
int close(int sockfd);
```

- Đóng bộ mô tả ổ cắm •

[IN] sockfd: bộ mô tả xác định ổ cắm. • Giá trị trả về

- Trả về 0 nếu không có lỗi xảy ra.
- Ngược lại, trả về -1 (và errno sẽ được đặt tương ứng) •

close() so với shutdown()

- close() cố gắng hoàn thành quá trình truyền này trước khi đóng, giải phóng bộ mô tả ổ cắm • shutdown(): dừng ngay việc nhận và truyền dữ liệu, không giải phóng bộ mô tả ổ cắm

tùy chọn ổ cắm

```
#include <sys/socket.h>
int setsockopt (int sockfd, int level, int optname, void
                *optval, int optlen);
```

- Đặt các tùy chọn kiểm soát việc truyền dữ liệu trên socket •

Các thông số:

- [IN] sockfd: tham chiếu đến bộ mô tả ổ cắm mở •
- [IN] level: chỉ định cấp độ giao thức mà tùy chọn cư trú • [IN] optname: chỉ định một tùy chọn để đặt • [IN] optval: trỏ đến tùy chọn đã đặt giá trị • [IN] optlen: kích thước của giá trị tùy chọn được chỉ định bởi optval
- Trả lại:
 - Trả về 0 nếu không có lỗi xảy ra.
 - Ngược lại, trả về -1 (và errno sẽ được đặt tương ứng)

Tùy chọn ổ cắm (tiếp)

```
#include <sys/socket.h>
int getsockopt (int sockfd, int level, int optname, void
                *optval, int *optlen);
```

- Nhận các tùy chọn điều khiển truyền dữ liệu trên socket • Tham số:
 - [IN] sockfd: tham chiếu đến bộ mô tả ổ cắm mở •
 - [IN] level: chỉ định cấp độ giao thức mà tùy chọn cư trú • [IN] optname: chỉ định một tùy chọn để đặt • [OUT] optval: trỏ đến tùy chọn đã đặt giá trị • [IN, OUT] optlen: kích thước của giá trị tùy chọn được chỉ định bởi optval
- Trả lại:
 - Trả về 0 nếu không có lỗi xảy ra.
 - Ngược lại, trả về -1 (và errno sẽ được đặt tương ứng)

cấp độ = SOL_SOCKET

tên giá trị	Loại hình	Sự mô tả
SO_BROADCAST	int	Định cấu hình ổ cắm để gửi dữ liệu quảng bá. (Chỉ ổ cắm UDP)
SO_DONTROUTE	int	Đặt xem dữ liệu gửi đi có được gửi trên giao diện mà ổ cắm được liên kết và không được định tuyến trên một số giao diện khác hay không
SO_KEEPALIVE	int TCP	tự động gửi một thăm dò liên tục đến ngang hàng
SO_LINGER	nán lại	chỉ định cách chức năng đóng hoạt động cho một giao thức hướng kết nối
SO_REUSEADDR	int	Cho phép ổ cắm được liên kết với một địa chỉ đã được sử dụng timeval Đặt thời gian chờ để
SO_RCVTIMEO	chặn các	cuộc gọi nhận
SO_SNDTIMEO	timeval	Đặt thời gian chờ để chặn cuộc gọi gửi

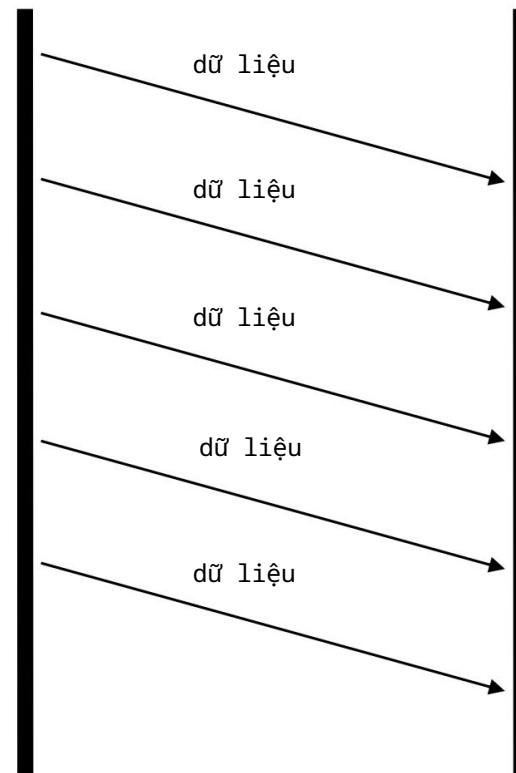
Ổ CẮM UDP

UDP (Giao thức gói dữ liệu người dùng)

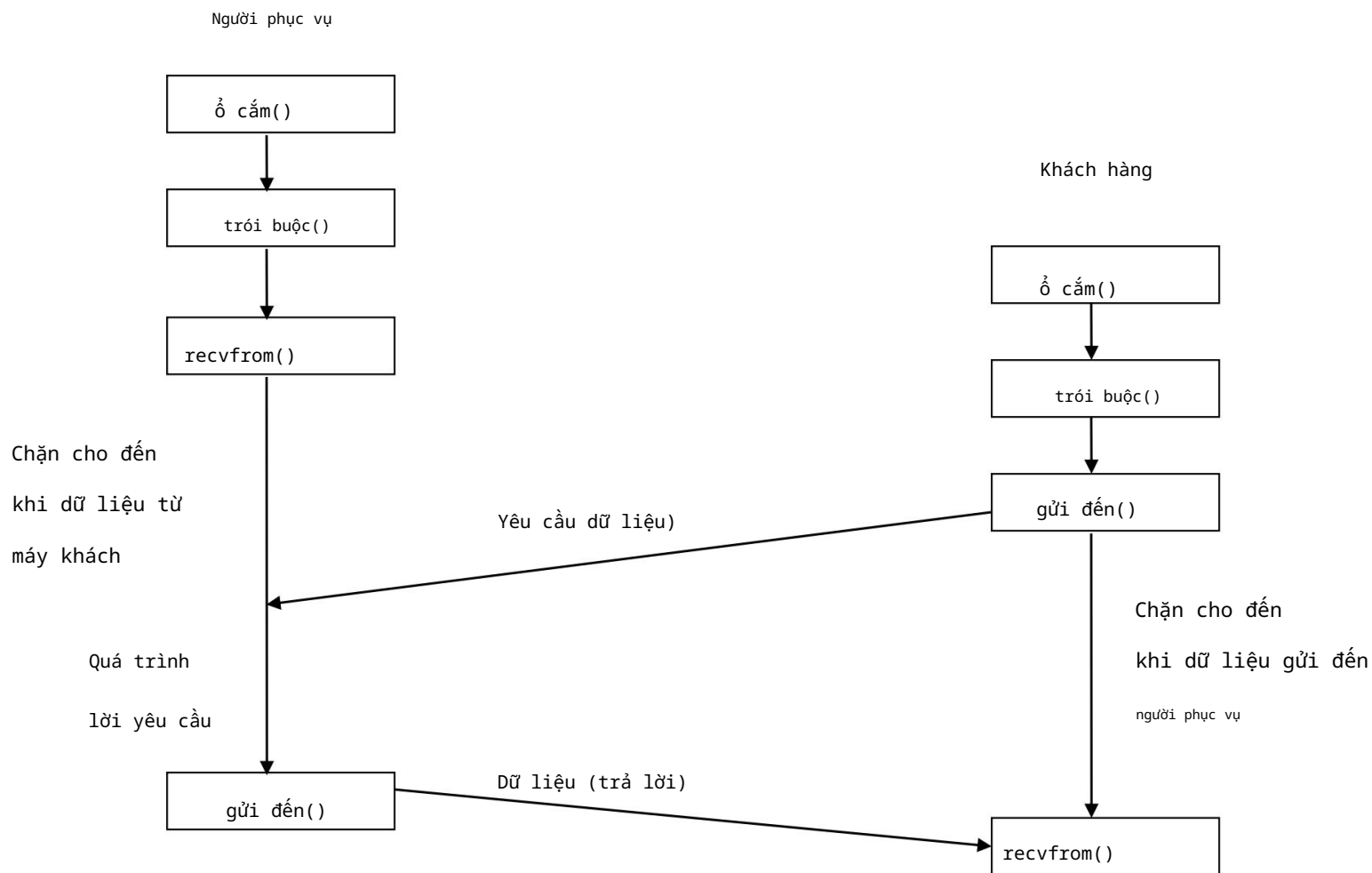
- Không đáng tin cậy
- Không kiểm soát dòng chảy
- Ví dụ quen thuộc •
 - DNS
 - Truyền trực tuyến
 - Hình ảnh
 - Trao đổi bưu thiếp

người phục vụ

khách hàng



Máy khách/máy chủ UDP



recvfrom()

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct
sockaddr *from, socklen_t *fromlen );
```

- Đã nhận dữ liệu từ một ổ cắm
- Thông số:
 - [IN] sockfd: bộ mô tả tập tin ổ cắm • [OUT] buf: bộ đệm nơi lưu trữ thông báo • [IN] len: kích thước của bộ đệm • [IN] flags: cách điều khiển chức năng recvfrom hoạt động • [OUT] from: địa chỉ người gửi
 - [OUT] fromlen: kích thước địa chỉ người gửi • Return:
- Thành công: trả về độ dài của dữ liệu nhận được theo byte. Nếu đến thông báo quá dài để vừa với bộ đệm được cung cấp, các byte thừa sẽ bị loại bỏ.
- Lỗi: 1 và đặt errno để báo lỗi.

recvfrom() - Cờ

- MSG_PEEK: Xem một tin nhắn đến. Dữ liệu được coi là chưa đọc và hàm `recvfrom()` hoặc chức năng tương tự tiếp theo sẽ vẫn trả về dữ liệu này.
- MSG_OOB: Yêu cầu dữ liệu ngoài băng tần. Tầm quan trọng và ngữ nghĩa của dữ liệu ngoài băng tần là dành riêng cho giao thức.
- MSG_WAITALL: Trên ổ cắm `SOCK_STREAM`, yêu cầu này chặn chức năng cho đến khi có thể trả về toàn bộ lượng dữ liệu, ngoại trừ:
 - kết nối bị ngắt
 - MSG_PEEK đã được chỉ định
 - một lỗi đang chờ xử lý cho ổ cắm
 - một tín hiệu được bắt

toán tử OR (`|`) theo bit (`|`) để kết hợp nhiều hơn một cờ

gửi đến()

```
ssize_t sendto(int sockfd, void *buf, size_t len, int flags,
               struct sockaddr *to, socklen_t *tolen );
```

- Gửi dữ liệu trên ổ cắm
- Thông số:
 - [IN] sockfd: bộ mô tả tập tin socket
 - [IN] buf: trỏ đến vùng đệm chứa tin nhắn sẽ được gửi
 - [IN] len: kích thước của tin nhắn
 - [IN] flags: cách điều khiển chức năng sendto hoạt động
 - [IN] to: địa chỉ của người nhận
 - [IN] tolen: độ dài của cấu trúc sockaddr được trỏ tới bởi

để tranh

luận • Trả về:

- Thành công: sẽ trả về độ dài của tin nhắn đã gửi theo byte
- Lỗi: 1 và đặt errno để chỉ báo lỗi.

sendto() - Cờ

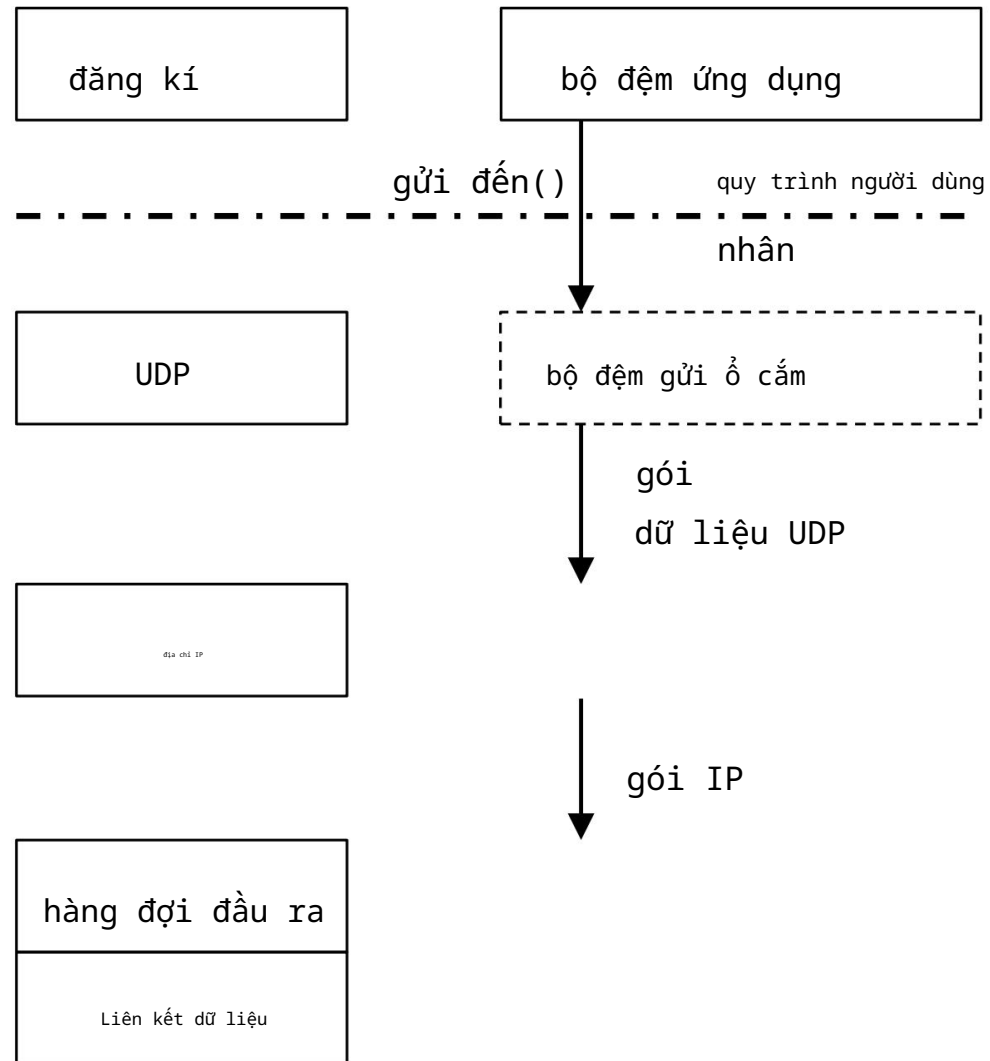
- MSG_OOB: Gửi dữ liệu ngoài băng tần trên các ổ cắm hỗ trợ dữ liệu ngoài băng tần.
- MSG_DONTROUTE:

Không sử dụng cổng để gửi gói, chỉ gửi đến máy chủ trên các mạng được kết nối trực tiếp

- Sử dụng toán tử OR (|) theo bit (|) để kết hợp nhiều cờ

gửi đến()

- Bộ đệm ổ cắm UDP không thực sự tồn tại
- Bộ đệm ổ cắm UDP có kích thước bộ đệm gửi
- Nếu một ứng dụng ghi một datagram lớn hơn kích thước bộ đệm gửi socket , EMSGSIZE sẽ được trả về



Thí dụ

- Máy khách và máy chủ UDP đơn giản •
Máy chủ nhận dữ liệu từ máy khách
 - Server gửi lại dữ liệu cho client
 - Nó có trong `udpserv01.c` và `dg_echo.c`



Ví dụ – Máy chủ UDP Echo

```
int sockfd, rcvBytes, sendBytes; socklen_t len;
buff char [BUFF_SIZE+1]; cấu trúc sockaddr_in
servaddr, cliaddr;

//Bước 1: Xây dựng socket if((sockfd =
socket(AF_INET, SOCK_DGRAM, 0)) < 0){ perror("Lỗi: "); trả về 0;

}

//Bước 2: Liên kết địa chỉ với socket
bzero(&servaddr, sizeof(servaddr)); servaddr.sin_family
= AF_INET; servaddr.sin_addr.s_addr =
htonl(INADDR_ANY); servaddr.sin_port = htons(SERV_PORT);
if(bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)))
{ perror("Lỗi: "); trả về 0;

}

printf("Máy chủ bắt đầu.");
```

Ví dụ - Máy chủ UDP Echo(tiếp)

```
//Bước 3: Giao tiếp với client
for ( ; ; ) len =
sizeof(cliAddr); rcvBytes = recvfrom(sockfd, buff,
    &len);

if(rcvBytes < 0)
    { perror("Lỗi: "); trả về
    0;

} buff[rcvBytes] = '\0';
printf("[%s.%d]: %s", inet_ntoa(cliAddr.sin_addr), ntohs(cliAddr.sin_port),
    msg);

sendBytes = sendto(sockfd, buff, rcvBytes, 0,
    (cấu trúc sockaddr *) &cliAddr, len);

if(sendBytes < 0)
    { perror("Lỗi: "); trả về
    0;

}
}
```

Ví dụ - Máy khách UDP Echo

```
int sockfd, rcvBytes, sendBytes; socklen_t len;
buff char [BUFF_SIZE+1]; cấu trúc sockaddr_in
servaddr;

//Bước 1: Xây dựng socket if((sockfd =
socket(AF_INET, SOCK_DGRAM, 0)) < 0){
    perror("Lỗi: ");
    trả về 0;
}

//Bước 2: Xác định địa chỉ của máy chủ
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr = inet_aton(SERV_ADDR, &servaddr.sin_addr); servaddr.sin_port =
htons(SERV_PORT);
```

Ví dụ – Máy khách UDP Echo(tiếp)

```
//Bước 3: Giao tiếp với máy chủ printf("Gửi tới máy  
chủ: "); get_s(buff, BUFF_SIZE);  
  
len = sizeof(servaddr); sendBytes  
= sendto(sockfd, buff, strlen(buff), 0, (struct sockaddr *) &seraddr, len);  
  
nếu (gửiBytes < 0){  
    perror("Lỗi: "); trả về 0;  
  
}  
  
rcvBytes = recvfrom(sockfd, buff, BUFF_SIZE, 0, (struct sockaddr *)  
                &seraddr, &len);  
  
if(rcvBytes < 0)  
    { perror("Lỗi: "); trả về 0;  
  
} buff[rcvBytes] = '\0';  
printf("Trả lời từ máy chủ: %s", buff);
```

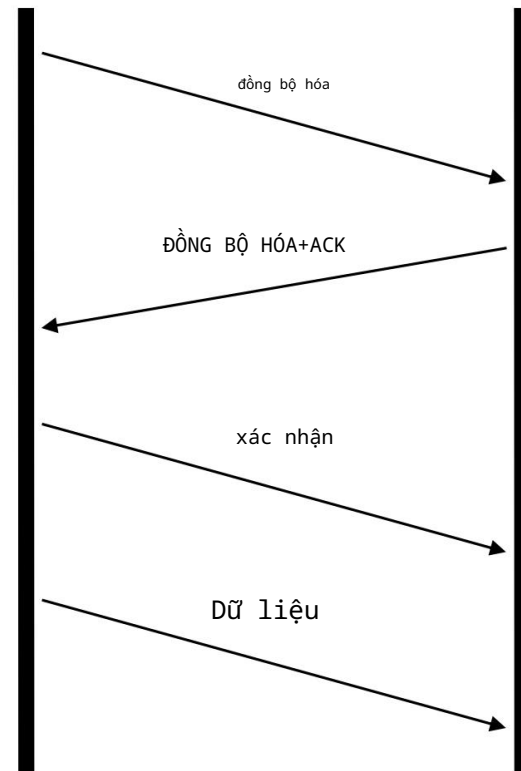
Ổ CẮM TCP

TCP (Giao thức điều khiển truyền dẫn)

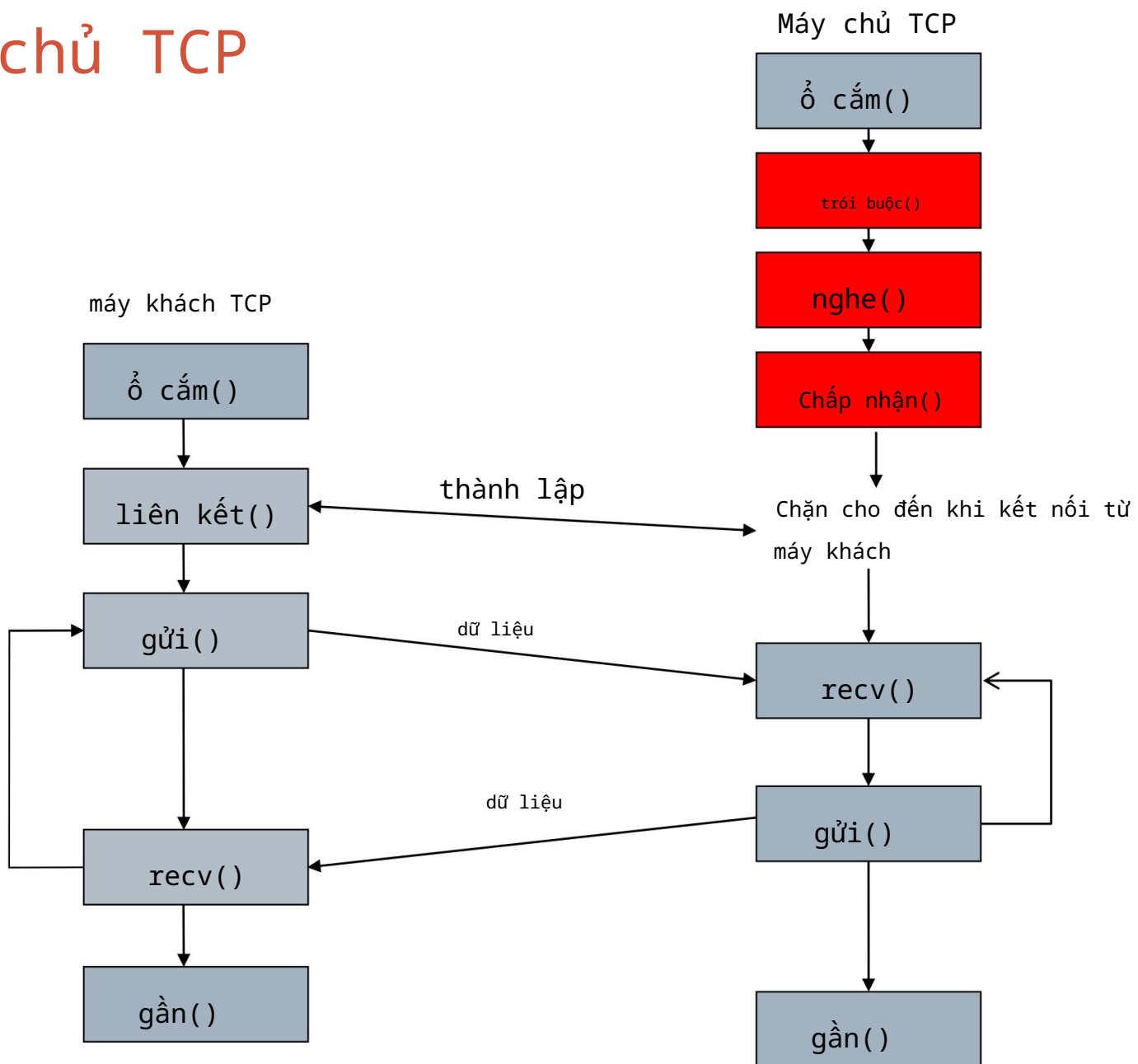
- Cung cấp thông tin liên lạc đáng tin cậy
- Kiểm soát tốc độ dữ liệu
- Ví dụ
 - Thư
 - TRANG WEB
 - Hình ảnh

khách hàng

người phục vụ



Máy chủ TCP



phía máy chủ TCP

1. Tạo một ổ cắm - `socket()`.
2. Liên kết ổ cắm - `bind()`.
3. Nghe trên socket - `listen()`.
4. Chấp nhận kết nối - `accept()`.
5. Gửi và nhận dữ liệu - `recv()`, `send()`.
6. Ngắt kết nối- `close()`
7. Đóng ổ LISTENING

} nhiều lần

nghe ()

```
#include <sys/socket.h>
int nghe(int sockfd, int backlog);
```

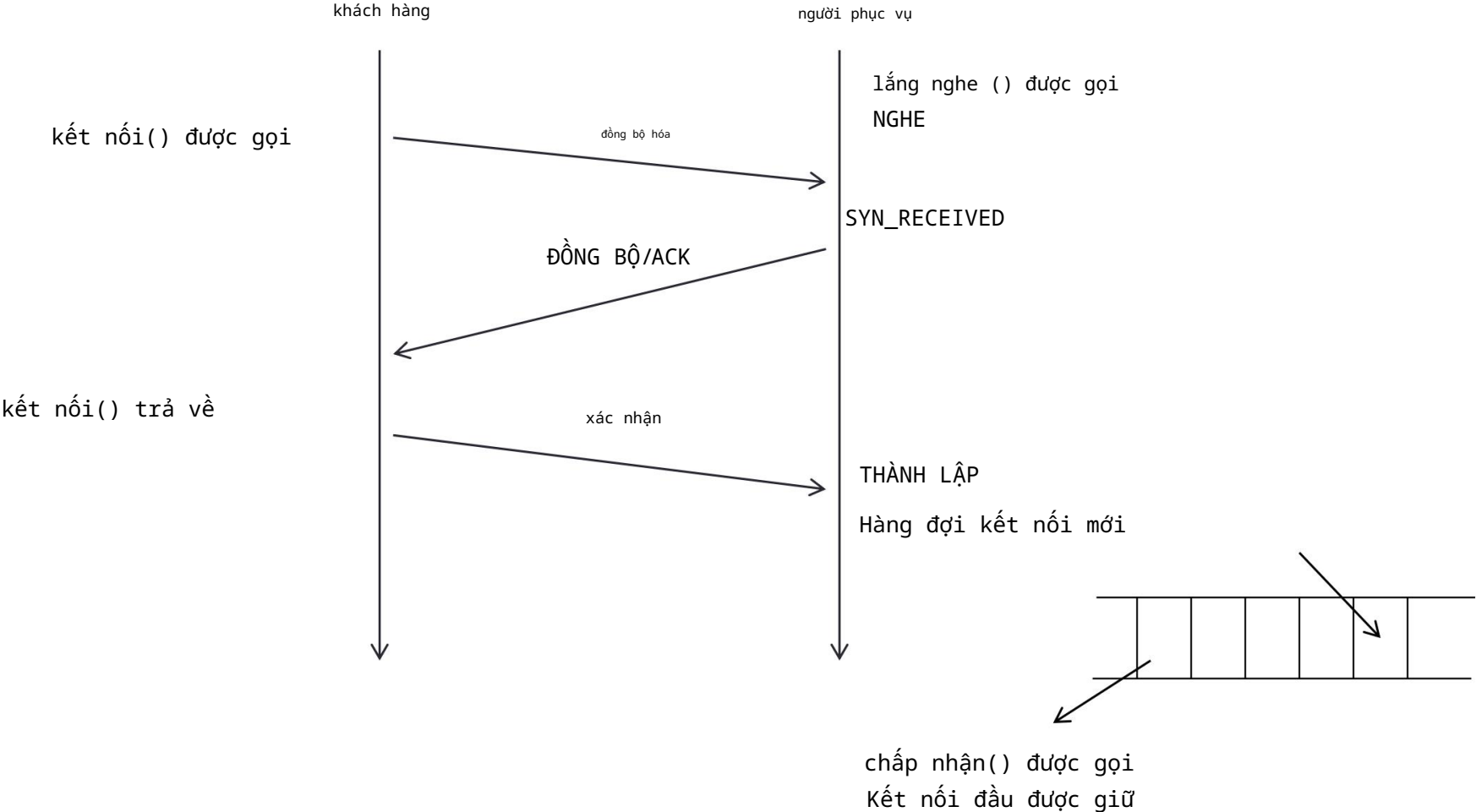
- Thiết lập một ổ cắm để LẮNG NGHE kết nối đến.
- Thông số:
 - [IN] sockfd: bộ mô tả xác định ổ cắm bị ràng buộc, không được kết nối
 - [IN] tồn đọng: độ dài hàng đợi cho các ổ cắm được thiết lập hoàn chỉnh đang chờ được chấp nhận
 - Giá trị trả về
 - Khi thành công, 0 được trả về
 - Khi có lỗi, -1 được trả về và errno được đặt phù hợp

Chấp nhận ()

```
#include <sys/types.h>
#include <sys/socket.h> int
accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- Chấp nhận kết nối đến trên ổ cắm LISTENING • Tham số:
 - [IN] sockfd: Một bộ mô tả xác định một ổ cắm đang lắng nghe các kết nối sau khi lắng nghe().
 - [OUT] addr: con trỏ tới cấu trúc sockaddr được điền địa chỉ của ổ cắm ngang hàng
 - [IN, OUT] addrlen: người gọi phải khởi tạo nó để chứa kích thước (tính bằng byte) của cấu trúc được trỏ tới bởi addr; khi trả lại, nó sẽ chứa kích thước thực của địa chỉ ngang hàng. • Giá trị trả về
 - Bộ mô tả ổ cắm mới được kết nối nếu không có lỗi
 - -1 nếu có lỗi

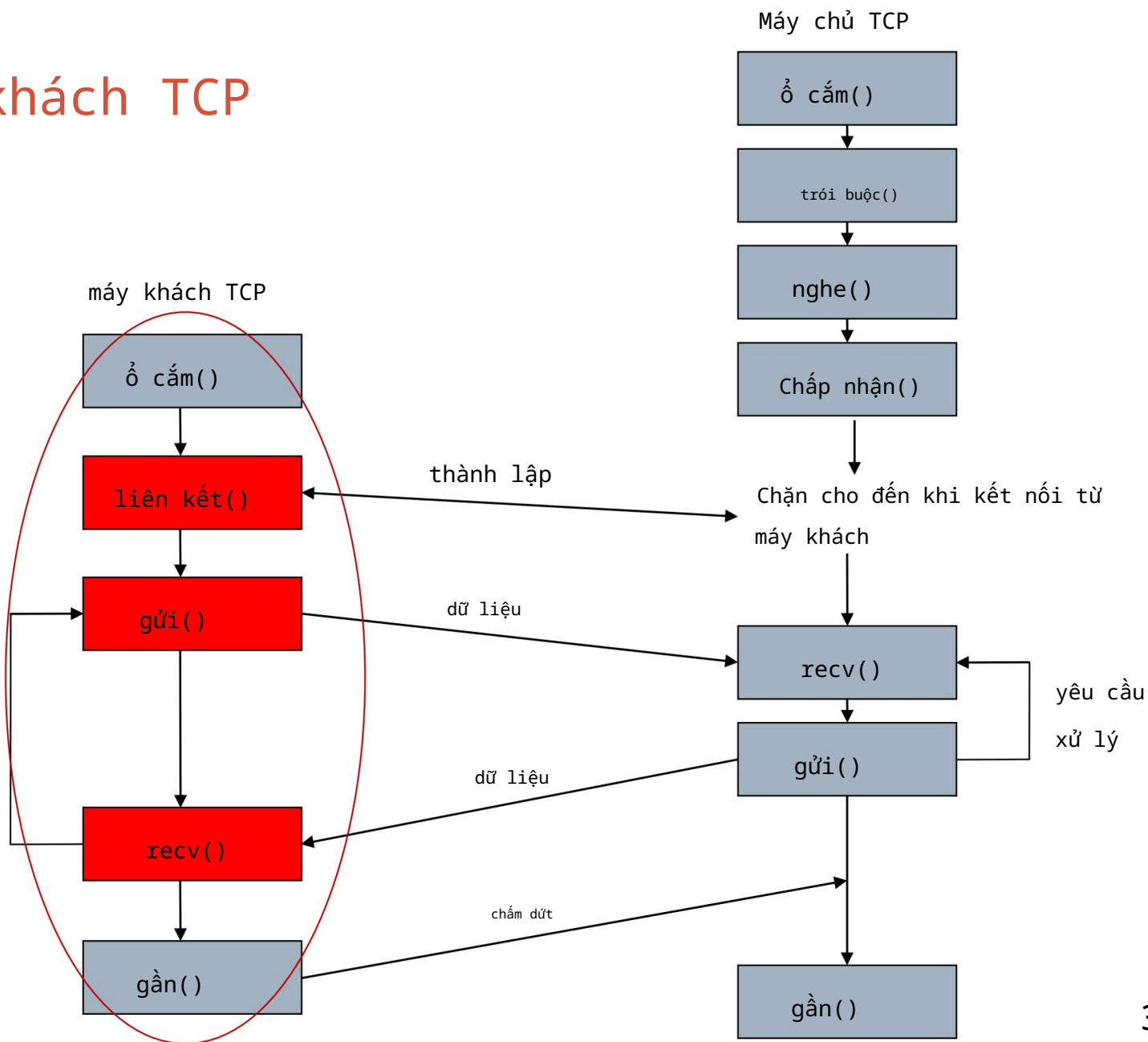
quá trình kết nối



Chế độ ổ cắm

- Các loại socket máy chủ
 - Máy chủ lặp: Mỗi lần chỉ mở một socket.
 - Máy chủ đồng thời: Sau khi chấp nhận, một tiến trình/luồng con được sinh ra để xử lý kết nối.
 - Máy chủ ghép kênh: sử dụng select để chờ đồng thời trên tất cả các socketId đang mở và chỉ đánh thức quy trình khi có dữ liệu mới

Máy khách TCP



phía máy khách TCP

- Giao tiếp của máy khách TCP điển hình bao gồm bốn bước cơ bản:
 - Tạo ổ cắm TCP bằng cách sử dụng `socket()`.
 - Thiết lập kết nối đến máy chủ bằng `connect()`.
 - Giao tiếp bằng `send()` và `recv()`.
 - Đóng kết nối với `close()`.
- Tại sao “client” không cần `bind()` ?

liên kết()

```
#include <sys/types.h> #include  
<sys/socket.h> int connect(int  
sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

- Kết nối ổ cắm với máy chủ
- Thông số:
 - [IN] sockfd: Bộ mô tả xác định ổ cắm chưa được kết nối.
 - [IN] serv_addr: Địa chỉ của máy chủ chứa ổ cắm kết nối.
 - [IN] addrlen: Độ dài của tên.
- Giá trị trả về
 - Nếu không xảy ra lỗi, trả về 0.
 - Ngược lại, trả về -1

gửi ()

```
#include <sys/types.h> #include  
<sys/socket.h> ssize_t send(int  
sockfd, const void *buf, size_t len, int flags);
```

- Gửi dữ liệu trên ổ cắm được kết nối
- Tham số:
 - [IN] sockfd: bộ mô tả xác định ổ cắm được kết nối.
 - [IN] buf: trỏ tới vùng đệm chứa thông điệp cần gửi.
 - [IN] len: chỉ định độ dài của tin nhắn
 - [IN] flags: chỉ định kiểu truyền tin nhắn, thường là 0
- Giá trị trả về:
 - Nếu không có lỗi xảy ra, send() trả về tổng số ký tự đã gửi
 - Nếu không, trả về -1

`gửi()` - Cờ

- `MSG_OOB`: Gửi dưới dạng dữ liệu “ngoài băng thông”. Bộ thu sẽ nhận được tín hiệu `SIGURG` và sau đó nó có thể nhận dữ liệu này mà không cần nhận trước tất cả phần còn lại của dữ liệu bình thường trong hàng đợi.
- `MSG_DONTROUTE`: Không gửi dữ liệu này qua bộ định tuyến, chỉ cần giữ lại địa phương.
- `MSG_DONTWAIT`: Nếu `send()` bị chặn vì lưu lượng gửi đi bị tắc, hãy trả lại `EAGAIN`. Điều này giống như “bật tính năng không chặn chỉ dành cho lần gửi này.”
- `MSG_NOSIGNAL`: Nếu bạn `gửi()` tới một máy chủ từ xa không còn `recv()`, thông thường bạn sẽ nhận được tín hiệu `SIGPIPE`. Việc thêm cờ này sẽ ngăn tín hiệu đó tăng lên.

send() - Kích thước dữ liệu lớn hơn bộ đệm

```
char sendBuff[2048]; int
dataLength, nLeft, idx;

// Điền vào sendbuff với 2048 byte dữ liệu nLeft = dataLength;
idx = 0;

trong khi (nLeft > 0){
    // Giả sử s là một ổ cắm luồng được kết nối, hợp lệ ret = send(s,
    &sendBuff[idx], nLeft, 0); nếu (ret == -1) {

        // Trình xử lý lỗi

    } nLeft -= ret; idx
    += ret;
}
```

recv()

```
#include <sys/types.h> #include  
<sys/socket.h> ssize_t recv(int  
sockfd, void *buf, size_t len, int flags);
```

- Nhận dữ liệu trên socket •

Tham số:

- [IN] sockfd: bộ mô tả xác định ổ cắm được kết nối. • [IN,
OUT] buf: trỏ đến vùng đệm chứa tin nhắn • [IN] len: chỉ định độ dài tính
bằng byte của vùng đệm • [IN] flags: chỉ định kiểu nhận tin nhắn, thường là 0
- Giá trị trả về:
 - Nếu không có lỗi xảy ra, trả về độ dài của tin nhắn đã nhận theo byte
 - Nếu ngang hàng đã thực hiện tất máy theo thứ tự, trả về 0 • Nếu không,
trả về -1

recv() - Cờ

- MSG_PEEK: Xem một tin nhắn đến. Dữ liệu được coi là chưa đọc và hàm `recvfrom()` hoặc chức năng tương tự tiếp theo sẽ vẫn trả về dữ liệu này.
- MSG_OOB: Yêu cầu dữ liệu ngoài băng tần. Tầm quan trọng và ngữ nghĩa của dữ liệu ngoài băng tần là dành riêng cho giao thức.
- MSG_WAITALL: Trên ổ cắm `SOCK_STREAM`, yêu cầu này chặn chức năng cho đến khi có thể trả về toàn bộ lượng dữ liệu, ngoại trừ:
 - kết nối bị ngắt
 - MSG_PEEK đã được chỉ định
 - một lỗi đang chờ xử lý cho ổ cắm
 - một tín hiệu được bắt

toán tử OR (`|`) theo bit (`|`) để kết hợp nhiều hơn một cờ

Ví dụ - Máy chủ TCP Echo

```
int nghefd, conncfd; buff char
[BUFF_SIZE+1]; cấu trúc sockaddr_in
servAddr;

//Bước 1: Xây dựng socket listenfd =
socket(AF_INET, SOCK_STREAM, 0);

//Bước 2: Liên kết địa chỉ với socket
bzero(&servaddr, sizeof(servaddr)); servaddr.sin_family
= AF_INET; servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT); if(bind(listenfd, (struct
sockaddr *) &servAddr, sizeof(servAddr))){

    perror("Lỗi: "); trả về 0;

}
```

Ví dụ - TCP Echo Server(tiếp)

```
//Bước 3: Lắng nghe yêu cầu từ client
if(nghe(listenfd, 10)){
    perror("Lỗi! Không nghe được.");
    trả về 0;
}

printf("Máy chủ bắt đầu!");

//Bước 4: Giao tiếp với client sockaddr_in
clientAddr; int rcvBytes, sendBytes, clientAddrLen
= sizeof(clientAddr); trong khi(1){

    // chấp nhận yêu cầu
    connfd = accept(listenfd, (sockaddr *) & clientAddr,
                    &clientAddrLen);
```


Ví dụ - TCP Echo Server(tiếp)

```
//nhận tin nhắn từ client rcvBytes =  
recv(connfd, buff, BUFF_SIZE, 0); if(rcvBytes < 0){ perror("Lỗi :");  
  
}  
khác{  
    buff[rcvBytes] = '\\0';  
    printf("Nhận từ client[%s:%d] %s\\n", inet_ntoa(clientAddr.sin_addr),  
           ntohs(clientAddr.sin_port), buff); // Tiếng vang  
           cho khách hàng  
  
    sendBytes = send(connfd, buff, strlen(buff), 0); nếu (gửiBytes < 0)  
  
        perror("Lỗi: ",);  
}  
closesocket(connfd);  
} // kết thúc khi
```

Ví dụ - Máy khách TCP Echo

```
int clientfd; buff
char [BUFF_SIZE+1]; cấu trúc
sockaddr_in servaddr;

//Bước 1: Xây dựng socket clientfd =
socket(AF_INET, SOCK_STREAM, 0);

//Bước 2: Chỉ định địa chỉ máy chủ bzero(&servaddr,
sizeof(servaddr)); servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(SERV_ADDR);
servaddr.sin_port = htons(SERV_PORT);

//Bước 4: Kết nối máy chủ
if(connect(clientfd, (sockaddr *) &serverAddr,
                                sizeof(serverAddr))){
    perror("Lỗi: "); trả về 0;
}
```

Ví dụ - TCP Echo Client(tiếp)

```
//Bước 5: Giao tiếp với server char
buff[BUFF_SIZE]; int ret;

//Gửi tin nhắn
printf("Gửi tới máy chủ: "); get_s(buff,
BUFF_SIZE); ret = send(clientfd, buff,
strlen(buff), 0); if(ret < 0){ perror("Lỗi: "); trả về 0;

}

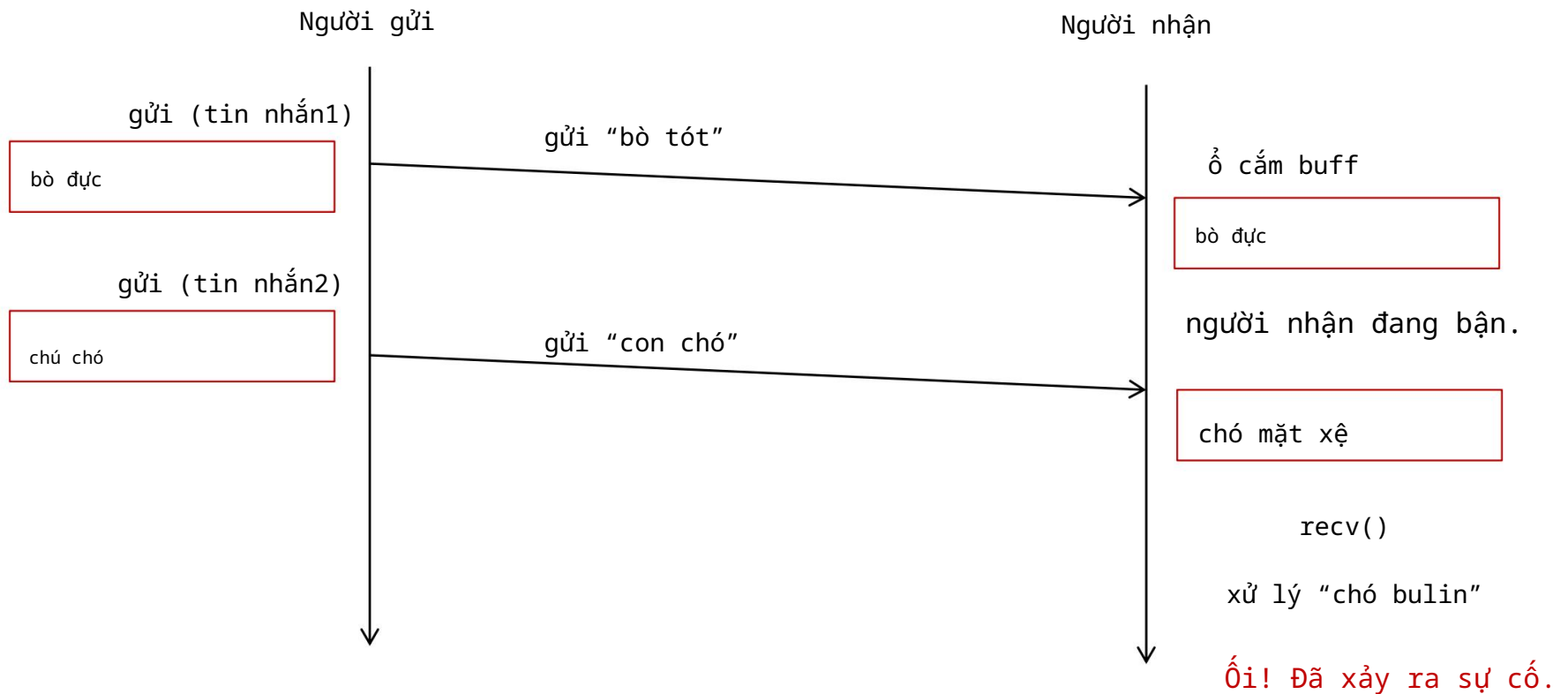
//Nhận tin nhắn echo ret =
recv(clientfd, buff, BUFF_SIZE, 0); if(ret < 0){ perror("Lỗi:
"); trả về 0;

} printf("Nhận từ máy chủ: %s\n", buff); đóng (khách hàngfd);
trả về 0;
```

Sự cố luồng byte

TCP không hoạt động trên các gói dữ liệu.

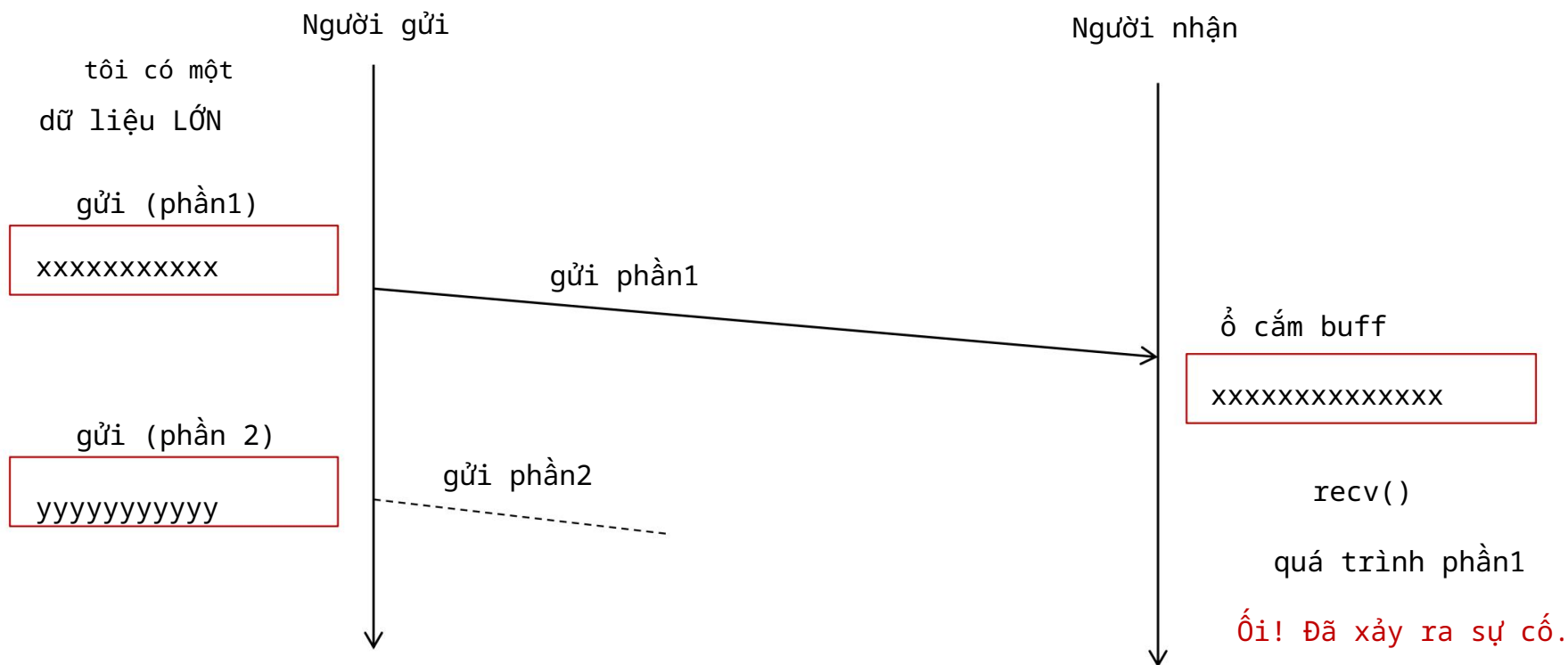
TCP hoạt động trên các luồng dữ liệu.



Vấn đề dòng byte(tiếp)

TCP không hoạt động trên các gói dữ liệu.

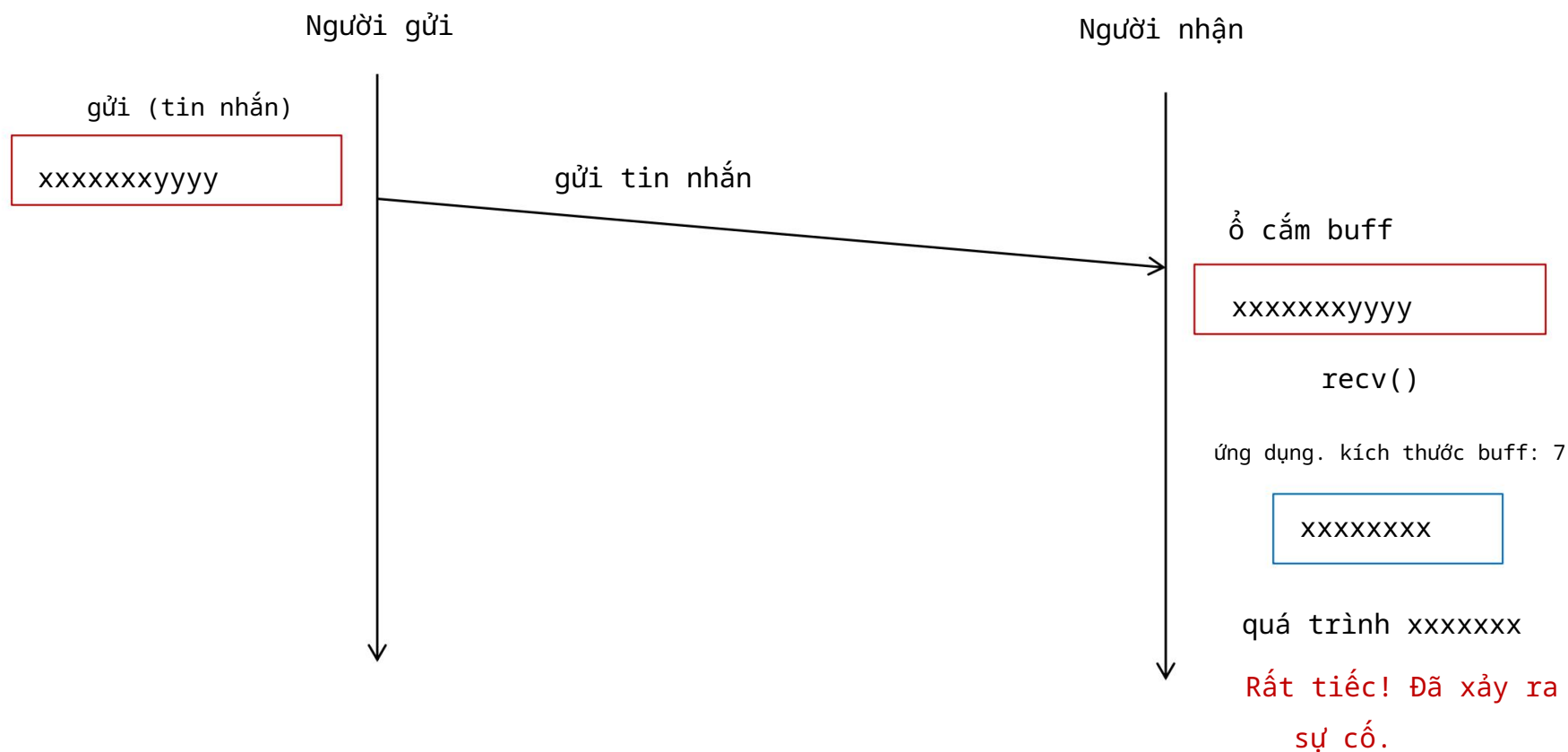
TCP hoạt động trên các luồng dữ liệu.



Vấn đề dòng byte(tiếp)

TCP không hoạt động trên các gói dữ liệu.

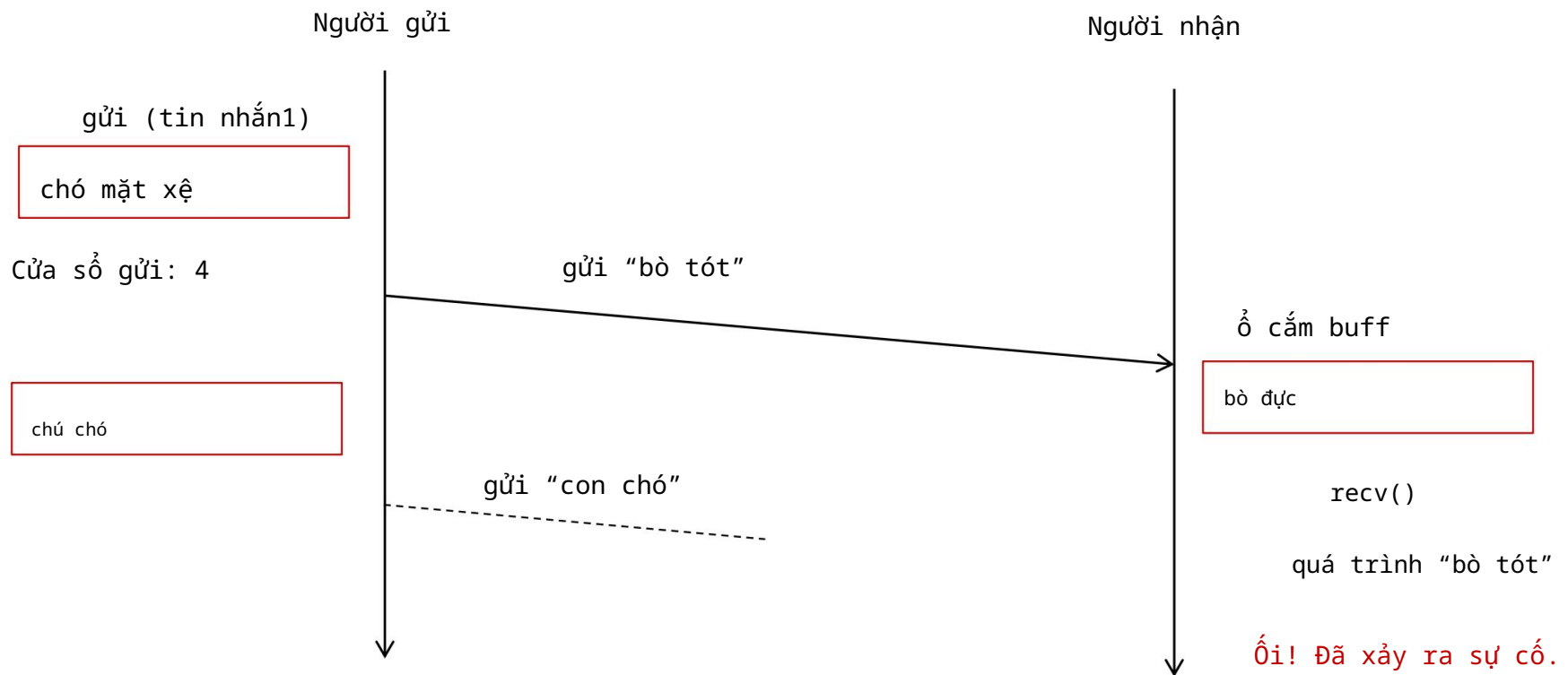
TCP hoạt động trên các luồng dữ liệu.



Vấn đề dòng byte(tiếp)

TCP không hoạt động trên các gói dữ liệu.

TCP hoạt động trên các luồng dữ liệu.



Vấn đề dòng byte(tiếp)

- Người nhận không biết kích thước của tin nhắn mà người gửi đã gửi

- Giải pháp 1: Tin nhắn có độ dài cố định • Độ dài là bao nhiêu? Làm thế nào để đếm?

- Giải pháp 2: Dấu phân cách
- | | | | |
|----------|--|------------|--|
| Tin nhắn | | Tin nhắn 2 | |
|----------|--|------------|--|

- 1 • Nhưng tin nhắn cũng có thể chứa dấu phân cách • Phức tạp! • Giải pháp 3: Thêm tiền tố

- độ dài • Gửi tin nhắn kèm theo độ dài • Người - n byte

Người - n byte	Thông điệp
----------------	------------

nhận:

- `recv(.,n, MSG_WAITALL)` trả về độ dài của tin nhắn • Nhận dữ liệu (slide tiếp theo)

Vấn đề dòng byte(tiếp)

```
than      recvBuff[BUFF_SIZE], *dữ liệu; ret,  
int       nLeft;  
nLeft = msgLength; //độ dài dữ liệu cần nhận data = (char *) malloc(msgLength); bộ nhớ  
(dữ liệu, 0, độ dài msg) idx = 0;  
  
trong khi (nLeft > 0) {  
  
    ret = recv(s, &recvBuff, BUFF_SIZE, 0); nếu (ret == -1){  
  
        // Trình xử lý lỗi ngắt;  
  
    } idx += ret;  
    memcpy(data + idx, recvBuff, ret) nLeft -= ret;  
  
}
```

kết nối() với UDP

- Nếu máy chủ không chạy, máy khách sẽ chặn vĩnh viễn lệnh gọi `recvfrom()` lỗi không đồng bộ
- Sử dụng `connect()` cho ổ cắm UDP
 - Nhưng nó khác với gọi `connect()` trên ổ cắm TCP
 - Gọi `connect()` trên ổ cắm UDP không tạo kết nối • Hạt nhân chỉ kiểm tra bất kỳ lỗi nào ngay lập tức và trả về ngay lập tức đến quá trình gọi
- Chúng ta không sử dụng `sendto()` mà thay vào đó là `write()` hoặc `send()` • Chúng ta không cần sử dụng `recvfrom()` để tìm hiểu người gửi của một datagram mà thay vào đó là `read()`, `recv()`
- Các lỗi không đồng bộ được trả về quy trình cho các ổ cắm UDP được kết nối

Thí dụ

```
số nguyên ;  
dòng gửi char [MAXLINE], recvline[MAXLINE + 1]; cấu trúc  
sockaddr_in servaddr; kết nối (sockfd, (struct sockaddr *)  
&servaddr, servlen);  
while (fgets(sendline, MAXLINE, fp) != NULL) {  
    gửi(sockfd, sendline, strlen(sendline));  
    n = recv(sockfd, recvline, MAXLINE); recvline[n] =  
    0; /* null chấm dứt */  
    printf("%s", recvline);  
}
```

ĐĂNG KÍ THIẾT KẾ GIAO THỨC

giao thức

- Bộ quy tắc:
 - Định dạng thông báo
 - Trình tự thông báo
 - Xử lý thông báo
- Mục tiêu
 - Mọi người phải biết
 - Mọi người phải đồng ý
 - Rõ ràng
 - Hoàn thành

Ví dụ: phiên POP

C: <máy khách kết nối với cổng dịch vụ 110>

S: +OK Máy chủ POP3 đã sẵn sàng <1896.6971@mailgate.dobbs.org>

C: NGƯỜI DÙNG

S: +Được rồi anh bạn

C: VƯỢT QUA nữ hoàng đỏ

S: +OK hộp thư của bob có 2 thư (320 octet)

C: DANH SÁCH

S: +OK 2 thông báo (320 octet)

S: 1 120

S: 2 200

Đ: .

C: BỎ CUỘC

S: +OK dewey máy chủ POP3 đăng xuất (maildrop trống)

C: <client hang u>

Ví dụ: xác thực FTP

> ftp 202.191.56.65

C: Đã kết nối với 202.91.56.65

S: 220 Chuỗi xác định máy chủ

Người dùng: vantv (C: USER vantv)

S: 331 Cần mật khẩu cho vantv

Mật khẩu:(C: PASS)

S: 530 Đăng nhập không chính xác

C: là

S: 530 Vui lòng đăng nhập bằng USER và PASS

C: NGƯỜI DÙNG vantv

S: 331 Cần mật khẩu cho vantv

Mật khẩu:(C: PASS)

S: 230 Người dùng vantv đã đăng nhập

Các bước trong thiết kế

1. Định nghĩa dịch vụ
 2. Chọn mô hình ứng dụng (client/server, P2P,..)
 3. Thiết lập các mục tiêu thiết kế
 4. Thiết kế cấu trúc thông báo: định dạng, trường, loại tin nhắn, mã hóa, .
 5. Xử lý giao thức 6.
- Tương tác với môi trường (DNS, DHCP.)

Mục tiêu thiết kế

- Chúng ta có cần những sàn giao dịch đáng tin cậy không?
- Có bao nhiêu loại hình tham gia? tất cả họ có thể giao tiếp với nhau?
- Việc xác thực các bên có cần thiết không?
- Việc xác thực các bên quan trọng như thế nào?
- Dữ liệu được chuyển có được bảo mật không? Mức độ ủy quyền nào là cần thiết?
- Chúng ta có cần xử lý lỗi phức tạp không?

Vấn đề thiết kế

- Nó có trạng thái hay không trạng thái?
- Giao thức vận chuyển đáng tin cậy hay không đáng tin cậy?
- Có cần trả lời không? • Làm thế nào để trả lời để mất trả lời?
- Nó sẽ được quảng bá, phát đa hướng hay đơn hướng? •
Broadcast, multicast: phải dùng UDP Socket
- Có nhiều kết nối không? • Đồng bộ như thế nào?
- Có bao nhiêu loại hình tham gia? tất cả họ có thể giao tiếp với nhau?
- Quản lý phiên làm việc •

Bảo mật: xác thực, cấp quyền, bảo mật.

Thiết kế thông điệp

- Tiêu đề: chứa các trường có cấu trúc mô tả dữ liệu thực trong thông báo, chẳng hạn như • loại thông báo • lệnh

- kích

thước cơ thể • thông

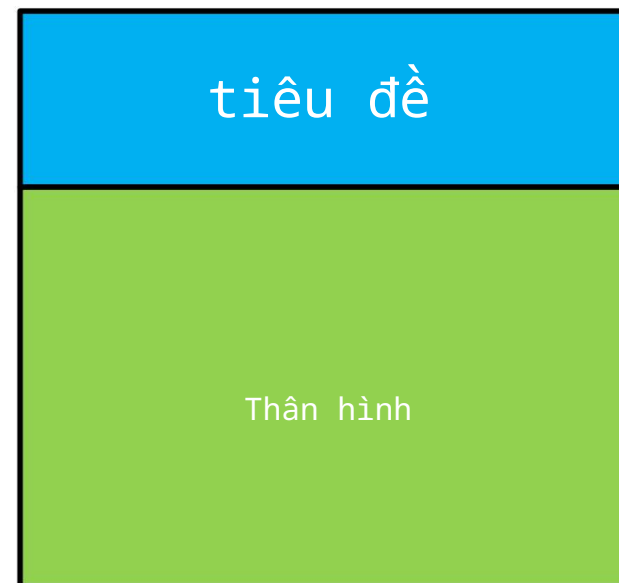
tin người nhận • thông

tin trình tự • số lần truyền lại.

- Body: dữ liệu thực sự được truyền đi:

- tham số lệnh • tải trọng

dữ liệu



Các định dạng đơn giản

nhất: • Loại - Độ dài - Giá trị (TLV) • Loại - Giá trị

Tin nhắn kiểm soát

- Xác định các giai đoạn đối thoại giữa các bên
- Kiểm soát đối thoại giữa các bên
- Giải quyết các khía cạnh giao tiếp khác nhau:
 - bắt đầu hoặc kết thúc giao tiếp
 - mô tả giai đoạn giao tiếp (ví dụ: xác thực, trạng thái

yêu cầu, truyền dữ

liệu) • phối hợp (ví dụ: xác nhận đã nhận, thử lại yêu

cầu) • thay đổi tài nguyên (ví dụ: yêu cầu cho các kênh liên lạc mới)

- Định dạng thông thường:

Tham số lệnh	
--------------	--

- Lệnh: NÊN có độ dài cố định hoặc sử dụng dấu phân cách
- Ví dụ: USER, PASS, PWD (FTP),

Truyền dữ liệu

- Tin nhắn mang dữ liệu qua mạng • Chúng thường được gửi dưới dạng phản hồi cho các lệnh cụ thể
- Dữ liệu thường bị phân mảnh trong nhiều thông điệp • Tiêu đề mô tả:
 - loại định dạng dữ liệu nhị phân • đầu mối cho bố cục của dữ liệu có cấu trúc (khi cấu trúc linh hoạt/động) • kích thước dữ liệu, thông tin về độ lệch hoặc trình tự • loại khối dữ liệu: cuối cùng/trung gian

Định dạng tin nhắn

Định hướng

theo byte • Phần đầu tiên của thông báo thường là một byte để phân biệt giữa các loại thông báo. •

Các byte tiếp theo trong tin nhắn sẽ chứa nội dung tin nhắn theo định dạng được xác định trước

• Ưu điểm: nhỏ gọn • Nhược

điểm: khó xử lý, gỡ lỗi hoặc kiểm tra hơn • Ví dụ:

DHCP, DNS


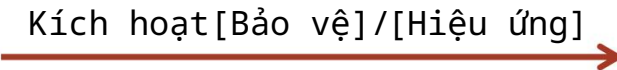

Định dạng dữ liệu

định hướng văn bản

- Một tin nhắn là một chuỗi gồm một hoặc nhiều dòng • Bắt đầu dòng đầu tiên của tin nhắn thường là một từ đại diện cho loại tin nhắn.
- Phần còn lại của dòng đầu tiên và các dòng tiếp theo chứa dữ liệu.
- Ưu điểm:
 - dễ hiểu, dễ theo dõi • linh hoạt
 - dễ kiểm tra
- Ví dụ: giao thức HTTP, FTP, email
- Nhược điểm • có thể làm cho thông báo lớn một cách vô lý • có thể trở nên phức tạp

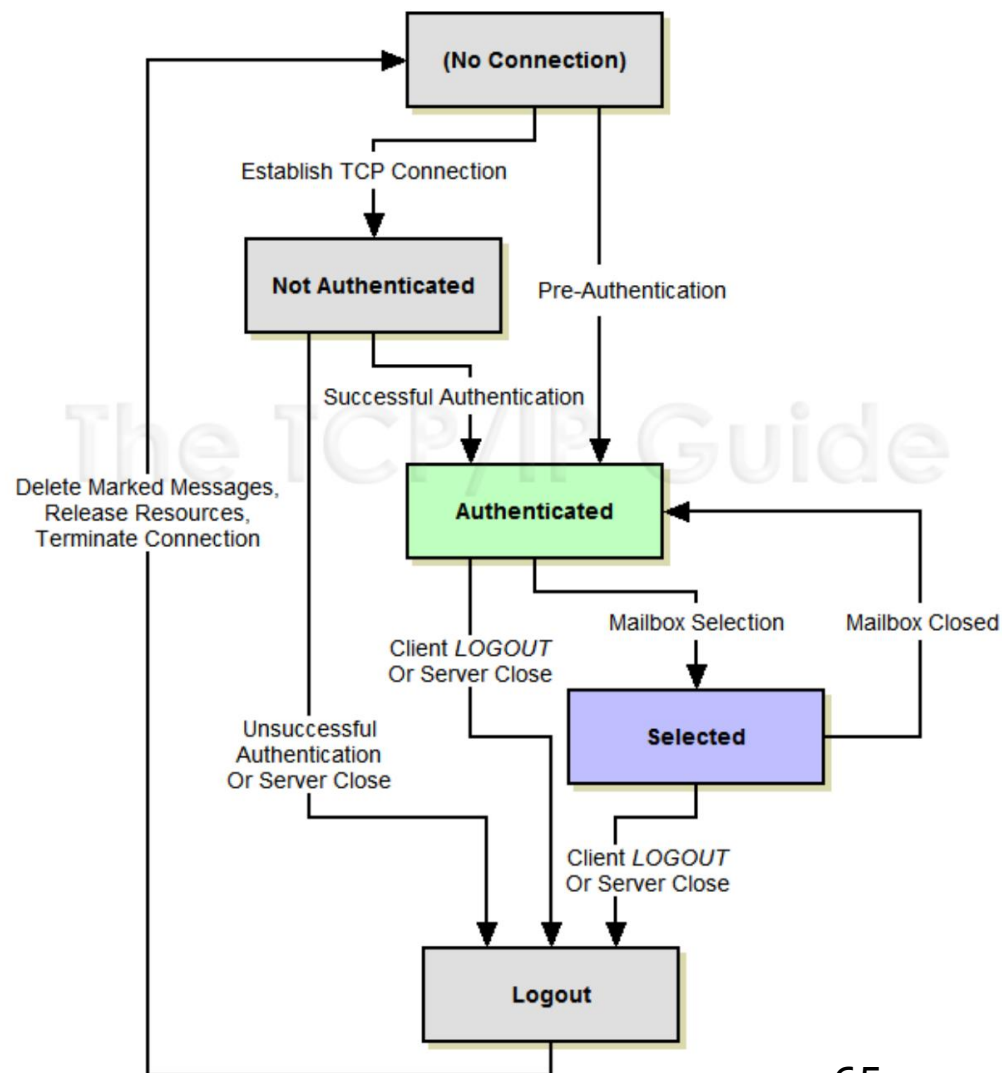
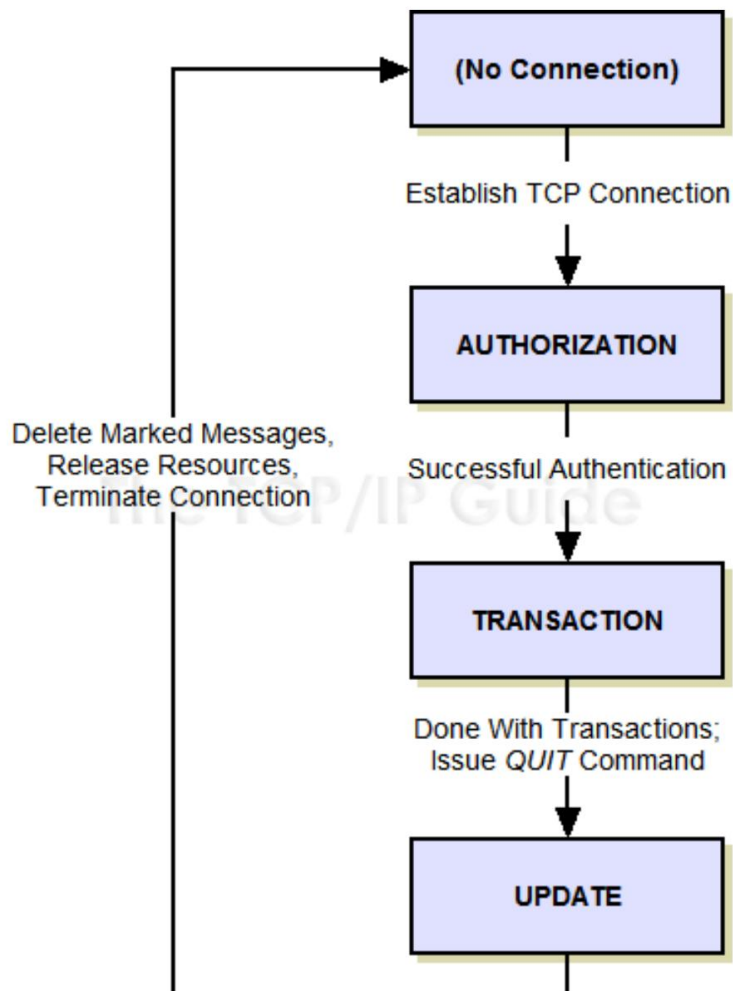
Xử lý giao thức

- Mô tả trình tự các thông báo, ở mỗi và tất cả các giai đoạn trong mỗi kịch bản giao tiếp, cho tất cả các bên trong hệ thống
- Máy trạng thái hữu hạn là bắt buộc:

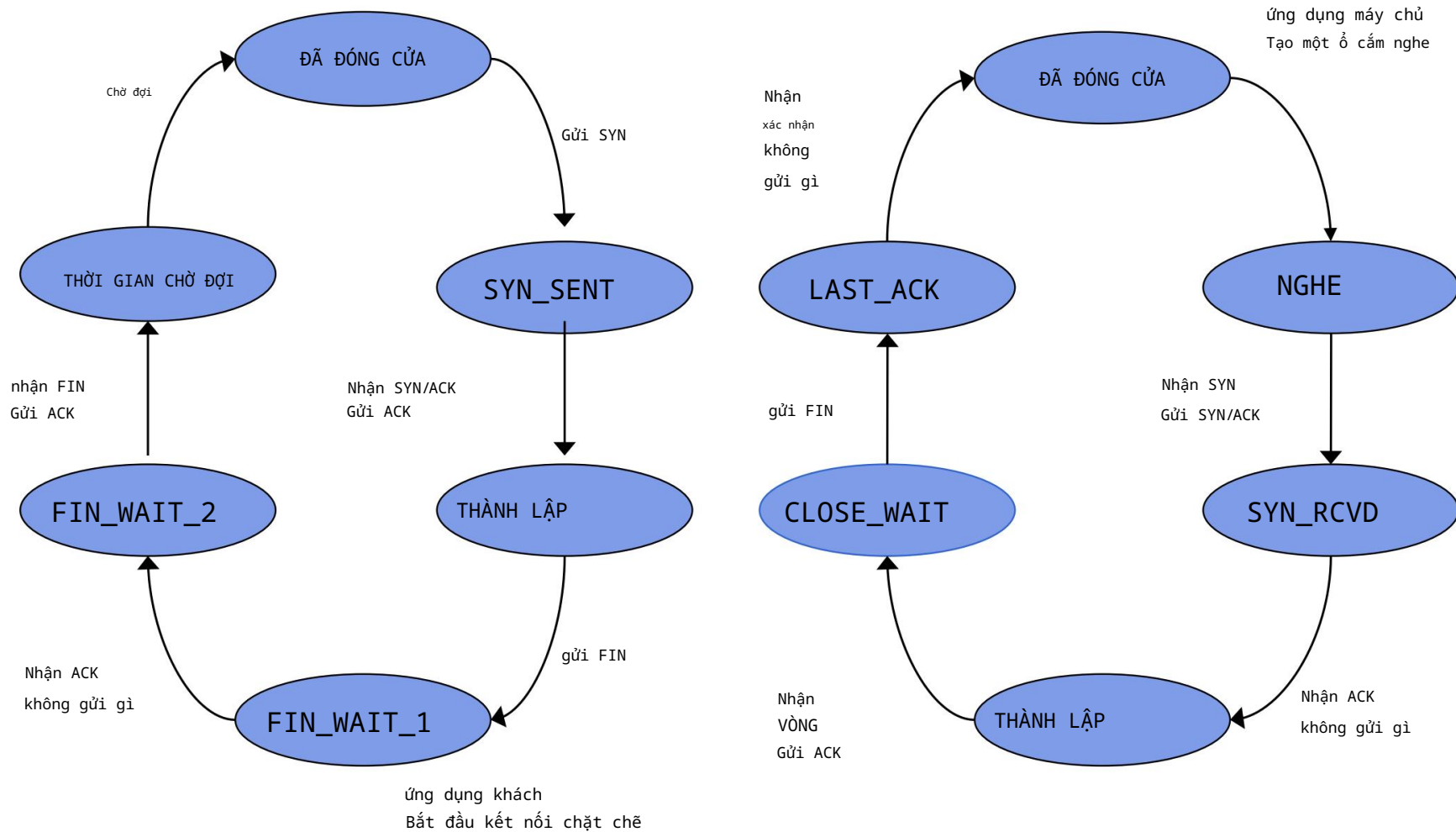
- Nhà nước: 
- Giao dịch:  Kích hoạt[Bảo vệ]/[Hiệu ứng]
- Chọn: 
- Và/Hoặc sử dụng Bảng trạng thái

Tình trạng hiện tại	chuyển đổi		trạng thái tiếp theo
	Nhận	Gửi	

Ví dụ: phiên POP3 và IMAP4

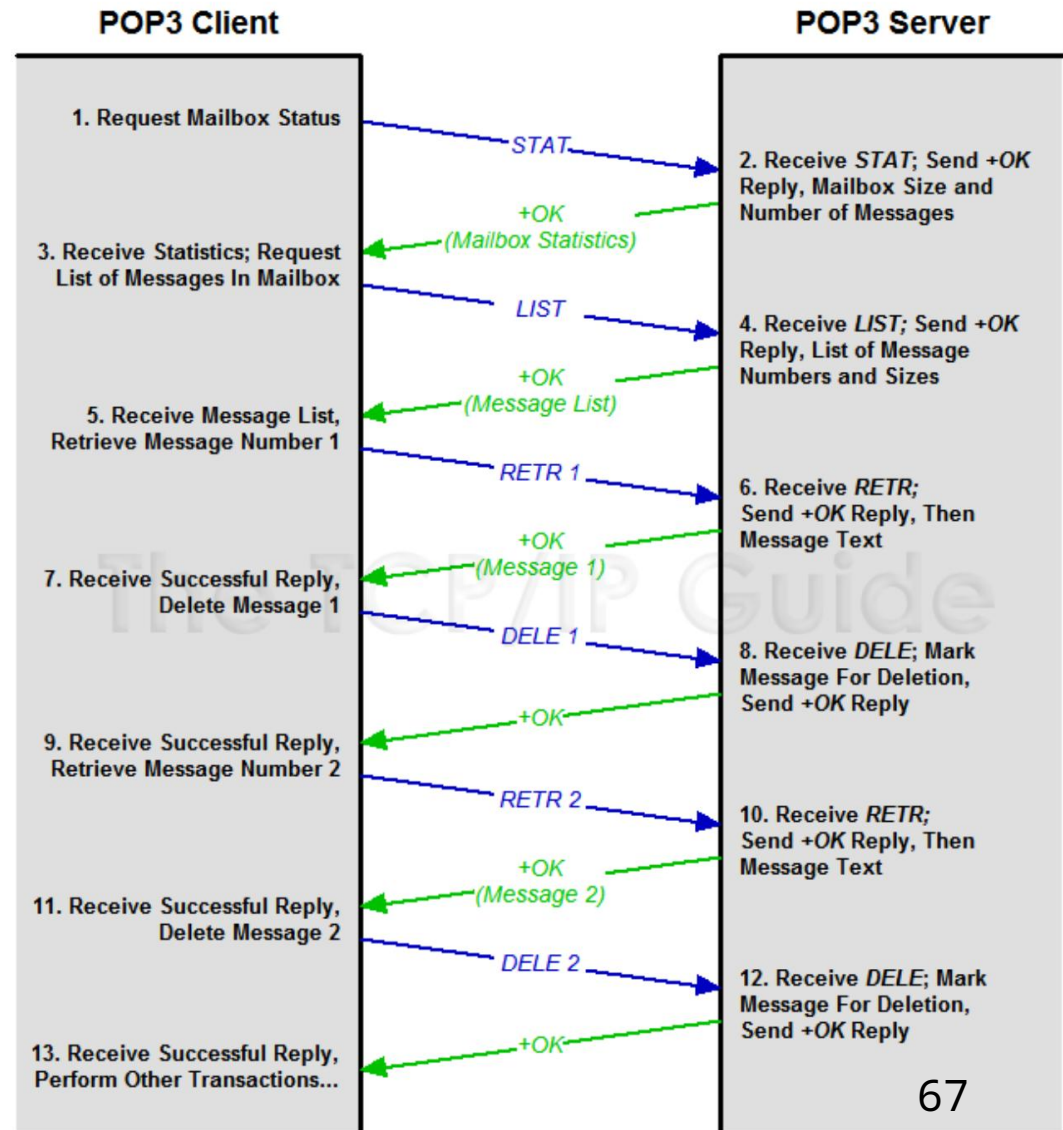


Ví dụ: kết nối TCP



Sơ đồ giao dịch tin nhắn

- Biểu diễn chuỗi giao dịch bản tin • Ví dụ: POP3



Thực hiện một giao thức ứng dụng

- Loại thông báo • Sử dụng số nguyên: `enum msg_type {.` • Sử dụng chuỗi • Cấu trúc dữ liệu

- Sử dụng cấu trúc. Thí dụ:

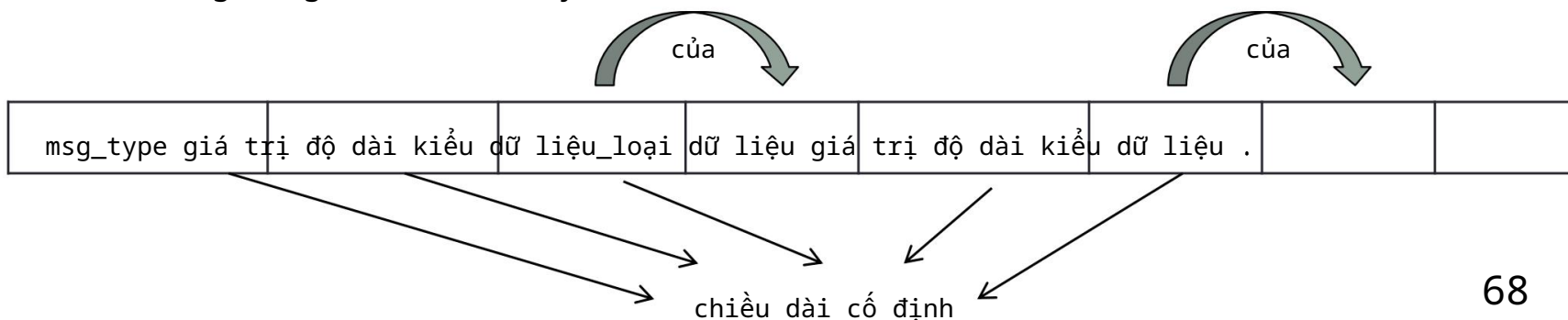
```
thông báo cấu trúc
{ char msg_type[4];
  kiểu_dữ_liệu [8]; giá trị
  int ;
}
```

hoặc

```
thông báo cấu trúc
{ loại msg_type; tải
  trọng struct msg_payload;
};
```

```
cấu trúc msg_payload{ int
  id; tên đầy đủ
  char [30]; int tuổi; //...
}
```

- Sử dụng mảng chuỗi hoặc byte



Thực hiện một giao thức ứng dụng

- Trình xử lý tin nhắn (mã giả)

```
// xử lý chuyển đổi  
tin nhắn (msg_type){ case  
    MSG_TYPE1: {  
  
        //...  
  
    } trường hợp MSG_TYPE2:  
    {  
        //...  
        if(data_type == DATA_TYPE1) //...  
  
    } //...  
}
```