

BÀI 2. BẮT ĐẦU VỚI LẬP TRÌNH WINSOCK

1

1

Nội dung

- Giới thiệu một số hàm lập trình WinSock cơ bản
- Xây dựng một ứng dụng TCP cơ bản
- Xây dựng một ứng dụng UDP cơ bản
- Thiết kế giao thức ứng dụng

2

2

1

1. MỘT SỐ HÀM CƠ BẢN

3

3

Khởi tạo WinSock

- WinSock cần được khởi tạo ở đầu mỗi ứng dụng trước khi có thể sử dụng
- Hàm WSASStartup sẽ làm nhiệm vụ khởi tạo

```
int WSASStartup(  
    WORD wVersionRequested,  
    LPWSADATA lpWSADATA  
);
```

- *wVersionRequested*: [IN] phiên bản WinSock cần dùng.
- *lpWSADATA*: [OUT] con trỏ chứa thông tin về WinSock cài đặt trong hệ thống.
- Giá trị trả về:
 - Thành công: 0
 - Thất bại: SOCKET_ERROR

4

4

2

Giải phóng WinSock

- Ứng dụng khi kết thúc sử dụng WinSock có thể gọi hàm sau để giải phóng tài nguyên về cho hệ thống

```
int WSACleanup(void);
```

- Giá trị trả về:
 - Thành công: 0
 - Thất bại: SOCKET_ERROR

```
// Initiates Winsock v2.2
WSADATA wsaData;
WORD wVersion = MAKEWORD(2,2);
WSAStartup(wVersion, &wsaData);

//do something with WinSock
//...

//Terminates use of the WinSock
WSACleanup();
```

5

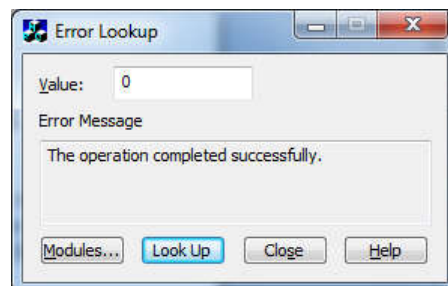
5

Xác định lỗi

- Phần lớn các hàm của WinSock nếu thành công đều trả về 0.
- Nếu thất bại, giá trị trả về của hàm là SOCKET_ERROR.
- Ứng dụng có thể lấy mã lỗi gần nhất bằng hàm

```
int WSAGetLastError(void);
```

- Tra cứu lỗi với công cụ Error Lookup trong Visual Studio



6

6

3

Địa chỉ socket

- Xác định địa chỉ
 - WinSock sử dụng cấu trúc **sockaddr_in** để lưu địa chỉ của socket
 - Địa chỉ IPv6: **sockaddr_in6**
 - Ứng dụng cần khởi tạo thông tin trong cấu trúc này

```
struct sockaddr_in{  
    short sin_family; // Loại địa chỉ cho socket  
    u_short sin_port; // Số hiệu cổng(big-endian)  
    struct in_addr sin_addr; // Địa chỉ IPv4  
    char sin_zero[8]; // Không sử dụng  
};
```

```
struct in_addr {  
    unsigned long s_addr;  
};
```

7

7

Các hàm hỗ trợ xử lý địa chỉ socket

- Chuyển đổi địa chỉ IP dạng xâu sang nhị phân

```
int inet_pton(  
    int family, // [IN] AF_INET hoặc AF_INET6  
    char ipstr, // [IN] Xâu biểu diễn  
    void* addr // [OUT] Biểu diễn dạng nhị phân  
);
```

- Trả về
 - 1 nếu thành công
 - 0 nếu xâu biểu diễn không hợp lệ
 - -1 nếu có lỗi
- Ví dụ:

```
in_addr address;  
inet_pton(AF_INET, "127.0.0.1", &address);
```

8

8

Các hàm hỗ trợ xử lý địa chỉ socket

- Chuyển đổi địa chỉ IP dạng nhị phân sang chuỗi

```
const char* inet_ntop(  
    int family,          //[IN] AF_INET hoặc AF_INET6  
    const void *addr,    //[OUT] Biểu diễn dạng nhị phân  
    char *ipstr,         //[IN] Chuỗi biểu diễn  
    size_t size          //[IN] Kích thước chuỗi biểu diễn  
);
```

- Trả về
 - Thành công: Con trỏ tới chuỗi biểu diễn
 - Thất bại: NULL

9

9

Các hàm hỗ trợ xử lý địa chỉ socket

- Biểu diễn số nguyên trong máy tính: little-endian hoặc big-endian
- Biểu diễn số nguyên trong mạng: big-endian
→ phải chuyển đổi biểu diễn số nguyên về đúng dạng
- Chuyển đổi host order => network order
`u_long htonl(u_long hostlong); //4 byte-value`
`u_short htons(u_short hostshort); //2 byte-value`
- Chuyển đổi network order => host order
`u_long ntohl(u_long netlong); //4 byte-value`
`u_short ntohs(u_short netshort); //2 byte-value`

10

10

Các hàm hỗ trợ xử lý địa chỉ socket

- Phân giải tên miền: **getaddrinfo()**
- Cần thêm tệp tiêu đề ws2tcpip.h

```
int getaddrinfo(  
    const char *nodename, //[IN] Tên miền hoặc địa chỉ IP  
    const char *servname, //[IN] Tên dịch vụ hoặc cổng  
    const struct addrinfo *hints, //[IN] cấu trúc gợi ý  
    struct addrinfo **res        //[OUT] danh sách liên kết  
                                   //chứa thông tin về địa chỉ  
);
```

- Giải phóng thông tin chứa trong kết quả:
`void freeaddrinfo(struct addrinfo *ai)`
- Các hàm tương tự: *gethostbyname()*, *gethostbyaddr()*, *gethostname()*

11

11

Cấu trúc *addrinfo*

```
typedef struct addrinfo {  
    int          ai_flags;        //[tùy chọn của hàm  
                                   //getaddrinfo()  
    int          ai_family;      //[họ giao thức  
    int          ai_socktype;    //[kiểu socket  
    int          ai_protocol;    //[giao thức tầng giao vận  
    size_t       ai_addrlen;     //[kích thước cấu trúc  
    char         *ai_canonname;  //[tên miền phụ  
    struct sockaddr *ai_addr;    //[địa chỉ socket  
    struct addrinfo *ai_next;    //[phần tử tiếp theo  
} ADDRINFOA, *PADDRINFOA;
```

12

12

Ví dụ

```
addrinfo *result;    //pointer to the linked-list
                     //containing information about the host
int rc;
sockaddr_in *address;
addrinfo hints;      //pointer to the linked-list
hints.ai_family = AF_INET; //only focus on IPv4 address
rc = getaddrinfo("soict.hust.edu.vn", NULL, &hints, &result);
// Get the address info
char ipStr[INET_ADDRSTRLEN];
if (rc == 0) {
    address = (struct sockaddr_in *) result->ai_addr;
    inet_ntop(AF_INET, &address->sin_addr, ipStr, sizeof(ipStr));
    printf("IPv4 address %s\n", ipStr);
}
else
    printf("getaddrinfo() error: %d", WSAGetLastError());
// free linked-list
freeaddrinfo(result);
```

13

13

Bài tập trên lớp

- Đọc hiểu mã nguồn chương trình
- Biên dịch và chạy mã nguồn minh họa cho các tên miền theo ý thích của sinh viên.
- Nâng cấp mã nguồn:
 - Phân giải tên miền nhập vào từ bàn phím
 - Hiển thị đầy đủ tất cả các địa chỉ IP trong kết quả phân giải. *Gợi ý: Duyệt toàn bộ danh sách liên kết trong tham số chứa thông tin đã phân giải được*

14

14

Khởi tạo socket

- SOCKET là một số nguyên để tham chiếu tới socket.
- Ứng dụng phải tạo SOCKET trước khi có thể gửi nhận dữ liệu.
- Trả về:
 - Thành công: Giá trị nguyên >0
 - Thất bại: *INVALID_SOCKET*
- Giải phóng socket sau khi sử dụng: *closesocket(SOCKET s)*

```
SOCKET socket (  
    int af,          // [IN] họ giao thức sẽ sử dụng  
                    // thường dùng AF_INET, AF_INET6  
    int type,        // [IN] Kiểu socket, SOCK_STREAM cho  
                    // TCP hoặc SOCK_DGRAM cho UDP  
    int protocol     // [IN] Giao thức tầng giao vận  
                    // IPPROTO_TCP hoặc IPPROTO_UDP  
);
```

15

15

Hàm *bind()*

- Gán địa chỉ cho socket

```
int bind(  
    SOCKET s, // [IN] socket chưa được gán địa chỉ  
    const struct sockaddr *name, // [IN] địa chỉ  
    int namelen // [IN] kích thước của giá trị được  
                // trả bởi tham số name  
);
```

- Ví dụ

```
sockaddr_in addr;  
short port = 8888;  
addr.sin_family = AF_INET;  
addr.sin_port = htons(port);  
addr.sin_addr.s_addr = htonl(INADDR_ANY);  
bind(s, (sockaddr *)&addr, sizeof(addr));
```

16

16

Tùy chọn trên socket

- WinSock cung cấp cơ chế cấu hình các thông số tùy chọn trên socket
- Thiết lập tùy chọn

```
int setsockopt (  
    SOCKET s,          //[IN]socket được thiết lập  
    int level,         //[IN]giao thức của tùy chọn  
    int optname,       //[IN]tên tùy chọn  
    const char FAR * optval, //[IN]giá trị thiết lập  
    int optlen         //[IN]kích thước của tham số optval  
);
```

- Lấy thông tin

```
int getsockopt (  
    SOCKET s,          //[IN]socket được thiết lập  
    int level,         //[IN]giao thức của tùy chọn  
    int optname,       //[IN]tên tùy chọn  
    const char FAR * optval, //[OUT]giá trị thiết lập  
    int FAR * optlen   //[IN/OUT]kích thước của optval  
);
```

17

17

Một số tùy chọn mức socket

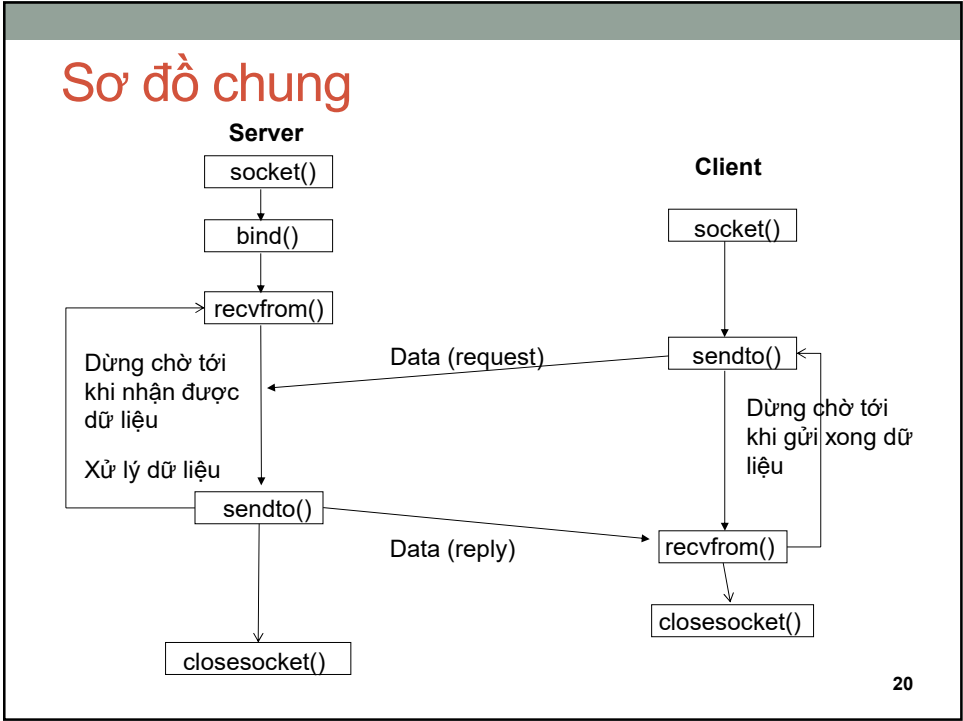
- *level* = *SOL_SOCKET*: Mức socket

Tùy chọn	Kiểu dữ liệu optval	Ý nghĩa
<i>SO_BROADCAST</i>	bool	Sử dụng socket để gửi thông tin quảng bá (chỉ sử dụng trên UDP socket và các giao thức hỗ trợ quảng bá)
<i>SO_KEEPALIVE</i>	DWORD	Socket gửi định kỳ các thông điệp keep-alive để duy trì kết nối
<i>SO_MAX_MSG_SIZE</i>	DWORD	Kích thước tối đa của gói tin
<i>SO_REUSEADDR</i>	bool	Cho phép socket sử dụng số hiệu cổng đang sử dụng bởi tiến trình khác
<i>SO_RCVTIMEO</i>	DWORD	Thiết lập thời gian time-out khi nhận dữ liệu ở chế độ chặn dừng (blocking)
<i>SO_SNDTIMEO</i>	DWORD	Thiết lập thời gian time-out khi gửi dữ liệu ở chế độ chặn dừng (blocking)

18

18

2. XÂY DỰNG ỨNG DỤNG VỚI UDP SOCKET



Hàm *sendto()*

- Gửi dữ liệu tới một tiến trình đích xác định
- Trả về:
 - Thành công: kích thước dữ liệu đã gửi đi (byte). Không hàm ý rằng ứng dụng phía bên kia đã nhận được.
 - Thất bại: `SOCKET_ERROR`
- Đọc thêm: *WSASendto()*

```
int sendto(  
    SOCKET s,           //[IN]socket sử dụng  
    const char *buf,     //[IN]bộ đệm chứa dữ liệu gửi  
    int len,            //[IN]kích thước dữ liệu gửi  
    int flags,          //[IN]cờ điều khiển. Thường là 0  
    const struct sockaddr *to, //[IN]địa chỉ đích  
    int tolen           //[IN]kích thước cấu trúc địa chỉ  
);
```

21

21

Hàm *recvfrom()*

- Nhận dữ liệu từ một nguồn xác định
- Trả về:
 - Thành công: kích thước dữ liệu ứng dụng đã nhận (byte).
 - Thất bại: `SOCKET_ERROR`
- Đọc thêm: *WSARecvFrom()*

```
int recvfrom(  
    SOCKET s,           //[IN]socket sử dụng  
    const char *buf,     //[OUT]bộ đệm chứa dữ liệu nhận  
    int len,            //[IN]kích thước bộ đệm nhận  
    int flags,          //[IN]cờ điều khiển. Thường là 0  
    const struct sockaddr *from, //[OUT]địa chỉ nút gửi  
    int *fromlen        //[OUT]kích thước cấu trúc địa chỉ  
);
```

22

22

Các cờ điều khiển

- Hàm *recvfrom()*

Giá trị cờ	Ý nghĩa
MSG_PEEK	Không xóa dữ liệu trong bộ đệm của socket sau khi nhận
MSG_OOB	Nhận dữ liệu out-of-band

- Hàm *sendto()*

Giá trị cờ	Ý nghĩa
MSG_DONTROUTE	Không chuyển dữ liệu tới default-gateway. Sử dụng khi gửi dữ liệu giữa các nút cùng mạng
MSG_OOB	Gửi dữ liệu out-of-band

- Sử dụng toán tử OR nhị phân (|) để kết hợp các cờ

23

23

Ví dụ: UDP Echo Server

- Server:

- Chờ dữ liệu trên cổng 5500
- Nhận thông điệp từ client gửi tới và hiển thị
- Trả lại thông điệp nhận được

- Client:

- Nhận thông điệp từ bàn phím
- Gửi dữ liệu tới cổng 5500 trên server
- Nhận thông điệp từ server và hiển thị

24

24

UDP server

```
//Step 1: Initiate WinSock
WSADATA wsaData;
WORD wVersion = MAKEWORD(2,2);
if(WSAStartup(wVersion, &wsaData))
    printf("Version is not supported\n");

//Step 2: Construct socket
SOCKET server;
server = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

//Step 3: Bind address to socket
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(5500);
inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
if(bind(server, (sockaddr *)&serverAddr, sizeof(serverAddr)))
{
    printf("Error! Cannot bind this address.");
    _getch();
    return 0;
}
```

25

25

UDP server (tiếp)

```
printf("Server started!");

//Step 4: Communicate with client
sockaddr_in clientAddr;
char buff[BUFF_SIZE], clientIP[INET_ADDRSTRLEN];
int ret, clientAddrLen = sizeof(clientAddr), clientPort;

while(1){
    //Receive message
    ret = recvfrom(server, buff, BUFF_SIZE, 0,
                  (sockaddr *)&clientAddr, &clientAddrLen);
    if(ret == SOCKET_ERROR)
        printf("Error : %", WSAGetLastError());
    else {
        buff[ret] = 0;
        inet_ntop(AF_INET, &clientAddr.sin_addr, clientIP,
                  sizeof(clientIP));
        clientPort = ntohs(clientAddr.sin_port);
        printf("Receive from client %s:%d %s\n",
              clientIP, clientPort, buff);
    }
}
```

26

26

UDP server (tiếp)

```
//Echo to client
ret = sendto(server, buff, ret, 0,
             (SOCKADDR *) &clientAddr, sizeof(clientAddr));
if(ret == SOCKET_ERROR)
    printf("Error: %", WSAGetLastError());

}
} //end while

//Step 5: Close socket
closesocket(server);

//Step 6: Terminate Winsock
WSACleanup();
```

27

27

UDP client

```
//Step 1: Initiate WinSock
WSADATA wsaData;
WORD wVersion = MAKEWORD(2,2);
if(WSAStartup(wVersion, &wsaData))
    printf("Version is not supported.\n");
printf("Client started!\n");

//Step 2: Construct socket
SOCKET client;
client = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
//optional Set time-out for receiving
int tv = 10000; //Time-out interval: 10000ms
setsockopt(client, SOL_SOCKET, SO_RCVTIMEO,
           (const char*)(&tv), sizeof(int));

//Step 3: Specify server address
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(5500);
inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
```

28

28

14

UDP client(tiếp)

```
//Step 4: Communicate with server
char buff[BUFF_SIZE];
int ret, serverAddrLen = sizeof(serverAddr);
do {
    //Send message
    printf("Send to server: ");
    gets_s(buff, BUFF_SIZE);
    ret = sendto(client, buff, strlen(buff), 0,
                (sockaddr *) &serverAddr, serverAddrLen);
    if(ret == SOCKET_ERROR)
        printf("Error! Cannot send message.");

    //Receive echo message
    ret = recvfrom(client, buff, BUFF_SIZE, 0,
                  (sockaddr *) &serverAddr, &serverAddrLen);
```

29

29

UDP client(tiếp)

```
if(ret == SOCKET_ERROR){
    if (WSAGetLastError() == WSAETIMEDOUT)
        printf("Time-out!");
    else printf("Error! Cannot receive message.");
}
else {
    buff[ret] = '\0';
    printf("Receive from server: %s\n", buff);
}
_strupr_s(buff, BUFF_SIZE);
}while(strcmp(buff, "BYE") != 0); //end while

//Step 5: Close socket
closesocket(client);

//Step 6: Terminate Winsock
WSACleanup();
```

30

30

Bài tập trên lớp

- Sinh viên chia thành từng cặp để thực hiện
- Yêu cầu bổ sung:
 - Chương trình client cho phép người dùng nhập thông điệp nhiều lần tới khi gặp xâu “bye”
 - Chương trình client hiển thị tổng số byte đã gửi
 - Chạy server ở địa chỉ IP và số hiệu cổng bất kỳ theo tham số dòng lệnh
- Dịch và chạy thử ứng dụng Echo trên 2 máy khác nhau
- Lưu ý: sửa lại các thông tin địa chỉ cho phù hợp

31

31

Kích thước bộ đệm

- Kích thước bộ đệm của UDP socket trên Windows 8.1 là 64KB
- Kích thước bộ đệm của ứng dụng:
 - Hàm *sendto()*: cần đủ lớn để chứa được thông điệp gửi đi
 - Dùng vòng lặp nếu dữ liệu gửi đi lớn hơn kích thước bộ đệm
 - Hàm *recvfrom()*: khi kích thước bộ đệm nhận nhỏ hơn kích thước thông điệp gửi tới:
 - Chỉ nhận phần dữ liệu vừa đủ với kích thước bộ đệm còn trống. Phần còn lại bị bỏ qua
 - Trả về SOCKET_ERROR

32

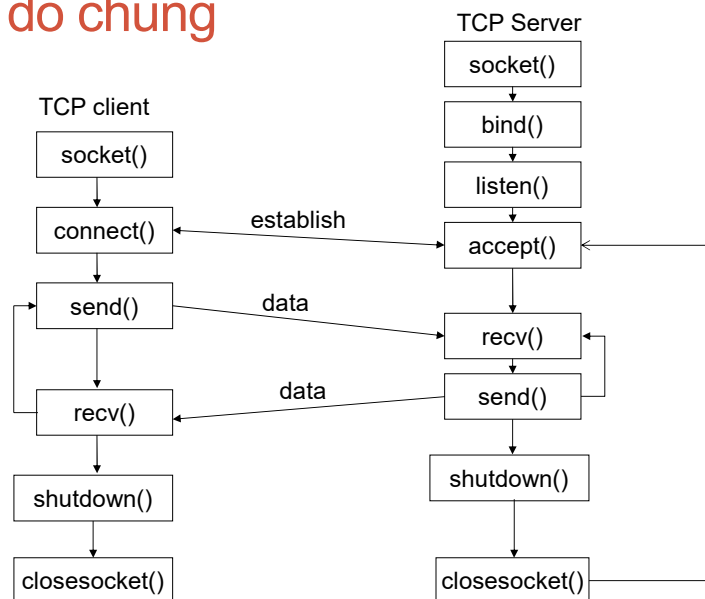
32

3. XÂY DỰNG ỨNG DỤNG VỚI TCP SOCKET

33

33

Sơ đồ chung



34

34

17

Hàm *listen()*

- Đặt SOCKET sang trạng thái lắng nghe kết nối (LISTENING)

```
int listen(SOCKET s, int backlog);
```

- Trong đó
 - **s**: [IN] SOCKET đã được tạo trước đó bằng hàm *socket()*
 - **backlog**: [IN] chiều dài hàng đợi chờ xử lý cho các kết nối đã được thiết lập
- Trả về:
 - Thành công: 0
 - Thất bại: SOCKET_ERROR

35

35

Hàm *accept()*

- Khởi tạo một SOCKET gắn với kết nối TCP nằm trong hàng đợi

```
SOCKET accept(  
    SOCKET s,    //[IN] socket đang ở trạng thái LISTENING  
    struct sockaddr *addr, //[OUT] Địa chỉ socket  
                //phía xin kết nối  
    int *addrlen    //[IN/OUT] Kích thước tham số addr  
);
```

- Trả về
 - Thành công: Một giá trị SOCKET gắn với kết nối TCP để trao đổi dữ liệu với client.
 - Thất bại: SOCKET_ERROR
- Đọc thêm về hàm *WSAAccept()*, *AcceptEx()*

36

36

Hàm *connect()*

- Gửi yêu cầu thiết lập kết nối tới server

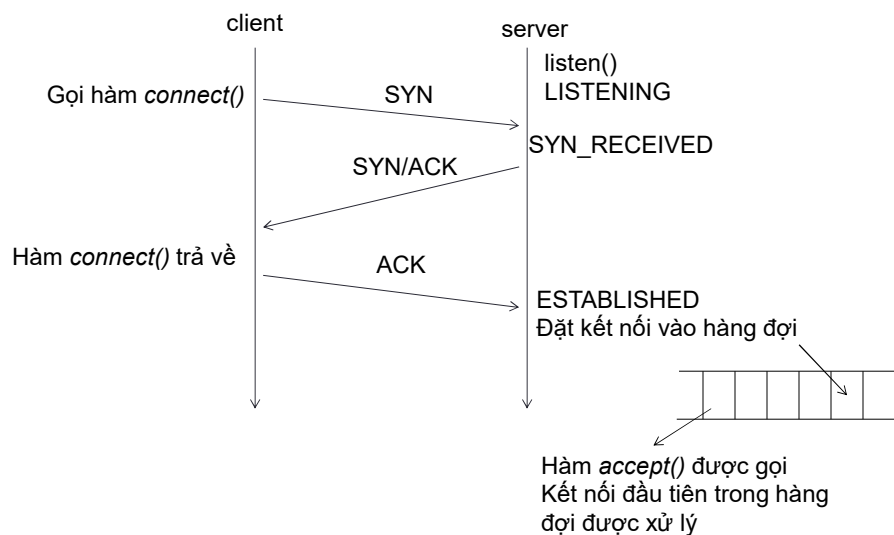
```
int connect(  
    SOCKET s,          //[IN] SOCKET của client  
    const struct sockaddr *name, //[IN] Địa chỉ server  
    int namelen        //[IN] Kích thước tham số name  
);
```

- Giá trị trả về
 - Thành công: 0 và một kết nối TCP đã được thiết lập.
 - Thất bại: SOCKET_ERROR
- Lưu ý: trên UDP socket có thể sử dụng hàm *connect()* để thiết lập địa chỉ của phía bên kia khi truyền tin
- Đọc thêm về hàm *WSAConnect()*, *ConnectEx()*

37

37

Thiết lập và xử lý kết nối



38

38

Hàm *send()*

- Gửi dữ liệu bằng SOCKET
- Trả về:
 - Thành công: kích thước dữ liệu đã gửi đi (byte). Không hàm ý rằng ứng dụng phía bên kia đã nhận được.
 - Thất bại: `SOCKET_ERROR`
- Lưu ý: nếu UDP socket đã dùng hàm *connect()* để kiểm tra, có thể sử dụng *send()* thay cho *sendto()*
- Đọc thêm: *WSASend()*

```
int send(  
    SOCKET s,           //[IN]socket sử dụng  
    const char *buf,    //[IN]bộ đệm chứa dữ liệu gửi  
    int len,            //[IN]kích thước dữ liệu gửi  
    int flags,          //[IN]cờ điều khiển. Thường là 0  
);
```

39

39

Hàm *recv()*

- Nhận dữ liệu bằng SOCKET
- Trả về:
 - Thành công: kích thước dữ liệu ứng dụng đã nhận (byte)
 - Thất bại: `SOCKET_ERROR`
- Lưu ý: nếu UDP socket đã dùng hàm *connect()* để kiểm tra, có thể sử dụng *recv()* thay cho *recvfrom()*
- Đọc thêm: *WSARecv()*

```
int recv(  
    SOCKET s,           //[IN]socket sử dụng  
    const char *buf,    //[OUT]bộ đệm chứa dữ liệu nhận  
    int len,            //[IN]kích thước bộ đệm  
    int flags,          //[IN]cờ điều khiển. Thường là 0  
);
```

40

40

Các cờ điều khiển

- Hàm *recv()*

Giá trị cờ	Ý nghĩa
MSG_PEEK	Không xóa dữ liệu trong bộ đệm sau khi nhận
MSG_OOB	Gửi dữ liệu out-of-band
MSG_WAITALL	Hàm <i>recv()</i> chỉ trả về khi: <ul style="list-style-type: none">- Nhận đủ số byte theo yêu cầu (tham số kích thước bộ đệm đã truyền khi gọi hàm)- Kết nối bị đóng- Có lỗi xảy ra

- Hàm *send()*

Giá trị cờ	Ý nghĩa
MSG_DONTROUTE	Không chuyển dữ liệu tới default-gateway. Sử dụng khi gửi dữ liệu giữa các nút cùng mạng
MSG_OOB	Gửi dữ liệu out-of-band

41

41

Hàm *shutdown()*

```
int shutdown(  
    SOCKET s,      //[IN] socket sử dụng  
    int how,       //[IN] cờ điều khiển  
);
```

- Đóng kết nối trên socket
- Cờ điều khiển
 - SD_RECEIVE: Đóng chiều nhận
 - SD_SEND: Đóng chiều gửi
 - SD_BOTH: Đóng đồng thời hai chiều
- Trả về:
 - Thành công: 0
 - Thất bại: SOCKET_ERROR

42

42

TCP Echo server

```
//Step 1: Initiate WinSock
//...
//Step 2: Construct socket
SOCKET listenSock;
listenSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

//Step 3: Bind address to socket
//...
//Step 4: Listen request from client
if(listen(listenSock, 10)){
    printf("Error: ");
    return 0;
}

printf("Server started!");
```

43

43

TCP Echo server (tiếp)

```
//Step 5: Communicate with client
sockaddr_in clientAddr;
char buff[1024], clientIP[INET_ADDRSTRLEN];
int ret, clientAddrLen = sizeof(clientAddr), clientPort;
SOCKET connSock;
//accept request
connSock = accept(listenSock, (sockaddr *) & clientAddr,
                  &clientAddrLen);
inet_ntop(AF_INET, &clientAddr.sin_addr, clientIP,
          sizeof(clientIP));
clientPort = ntohs(clientAddr.sin_port);
while(1){
    //receive message from client
    ret = recv(connSock, buff, 1024, 0);
    if(ret == SOCKET_ERROR){
        printf("Error %d", WSAGetLastError());
        break;
    }
}
```

44

44

TCP Echo server (tiếp)

```
else{
    printf("Receive from client[%s:%d] %s\n",
           clientIP, clientPort, buff);

    //Echo to client
    ret = send(connSock, buff, ret, 0);
    if(ret == SOCKET_ERROR){
        printf("Error %d", WSAGetLastError());
        break;
    }
} //end communicating

//Step 6: Close socket
closesocket(connSock);
closesocket(listenSock);

//Step 7: Terminate Winsock
WSACleanup();
```

45

45

TCP Echo client

```
//Step 1: Initiate WinSock
//...
//Step 2: Construct socket
SOCKET client;
client = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// (optional) Set time-out for receiving
int tv = 10000; //Time-out interval: 10000ms
setsockopt(client, SOL_SOCKET, SO_RCVTIMEO,
           (const char*)(&tv), sizeof(int));

//Step 3: Specify server address
//...
//Step 4: Request to connect server
if(connect(client, (sockaddr *) &serverAddr,
           sizeof(serverAddr))){
    printf("Error! Cannot connect server.");
    return 0;
}
```

46

46

TCP Echo client (tiếp)

```
//Step 5: Communicate with server
char buff[1024];
int ret, messageLen;
//Send message
while (1) {
    //Send message
    printf("Send to server: ");
    gets_s(buff, BUFF_SIZE);
    messageLen = strlen(buff);
    if (messageLen == 0) break;

    ret = send(client, buff, messageLen, 0);
    if (ret == SOCKET_ERROR)
        printf("Error %d", WSAGetLastError());
}
```

47

47

TCP Echo client(tiếp)

```
//Receive echo message
ret = recv(client, buff, BUFF_SIZE, 0);
if (ret == SOCKET_ERROR) {
    if (WSAGetLastError() == WSAETIMEDOUT)
        printf("Time-out!");
    else printf("Error %d", WSAGetLastError());
}
else if (strlen(buff) > 0) {
    buff[ret] = 0;
    printf("Receive from server: %s\n", buff);
}
}

//Step 6: Close socket
closesocket(client);

//Step 7: Terminate Winsock
WSACleanup();
```

48

48

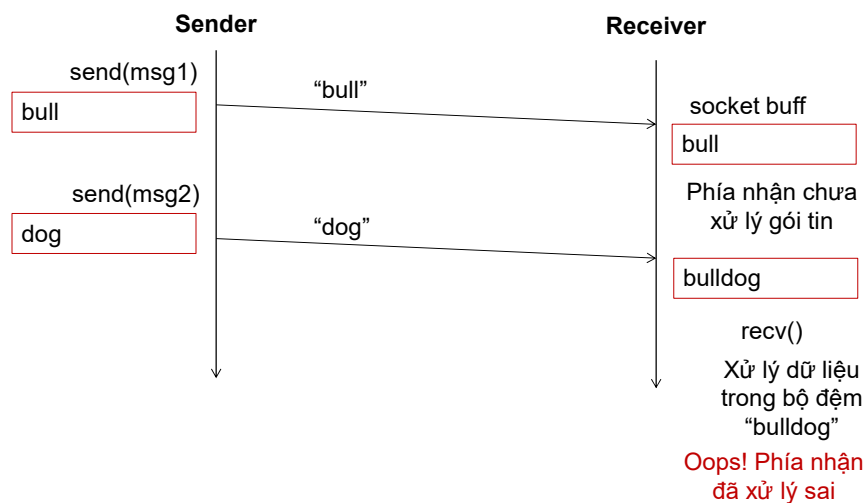
Kích thước bộ đệm

- Kích thước bộ đệm của TCP socket trên Windows 8.1 là 64KB
- Kích thước bộ đệm của ứng dụng:
 - Hàm *send()*: Dừng vòng lặp nếu dữ liệu gửi đi lớn hơn kích thước bộ đệm của ứng dụng
 - Sử dụng bộ đệm có kích thước lớn hiệu quả hơn khi kích thước dữ liệu gửi đi lớn
 - Hàm *recv()*: khi kích thước bộ đệm nhận nhỏ hơn kích thước thông điệp gửi tới cần sử dụng vòng lặp để đọc được hết dữ liệu
 - Làm thế nào để xác định đã nhận đủ dữ liệu?

49

49

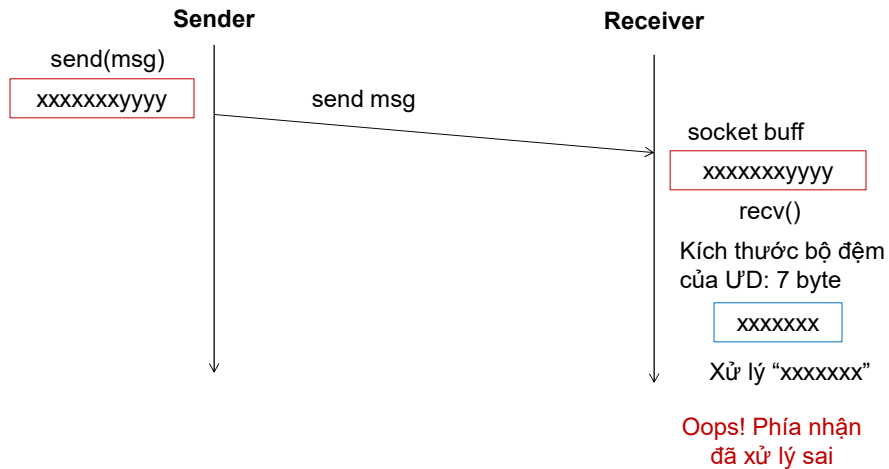
Truyền theo dòng byte trong TCP



50

50

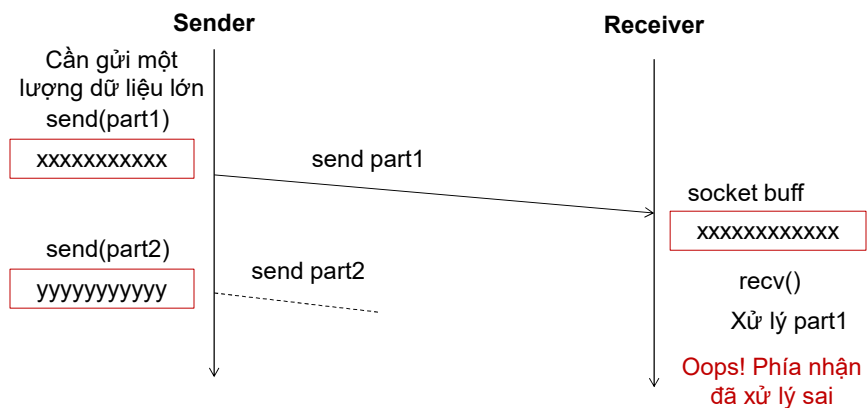
Truyền theo dòng byte trong TCP(tiếp)



51

51

Truyền theo dòng byte trong TCP(tiếp)



52

52

Truyền theo dòng byte trong TCP(tiếp)

- Phía nhận không biết kích thước dữ liệu mà phía gửi sẽ gửi đi
- Giải pháp 1: Sử dụng thông điệp có kích thước cố định
 - Vấn đề cần xử lý?
- Giải pháp 2: Sử dụng mẫu ký tự phân tách (delimiter)
 - Vấn đề cần xử lý



- Giải pháp 3: Gửi kèm kích thước thông điệp



Quy ước trước có bao nhiêu byte?(Ví dụ: 4 byte) n bytes

- Phía nhận: `recv(..., 4, MSG_WAITALL)` trả về kích thước của dữ liệu cần nhận

53

53

Bài tập trên lớp

- Client: Gửi 1 xâu bất kỳ chỉ chứa chữ cái và chữ số cho server
- Server: Trả lại 2 xâu, một xâu chỉ chứa các ký tự chữ số, một xâu chỉ chứa các ký tự chữ cái của xâu nhận được. Nếu xâu nhận được có ký tự đặc biệt, báo lỗi.

54

54

4. XÂY DỰNG GIAO THỨC CHO ỨNG DỤNG

55

55

Nhắc lại

- Giao thức là quy tắc:
 - Khuôn dạng, ý nghĩa bản tin
 - Thứ tự truyền các bản tin
 - Cách thức xử lý bản tin của mỗi bên
- Giao thức tầng ứng dụng: điều khiển hoạt động của các tiến trình của ứng dụng mạng
- Yêu cầu của giao thức:
 - Rõ ràng
 - Đầy đủ: bao quát mọi trường hợp có thể
 - Cam kết: các bên phải thực hiện đầy đủ và đúng thứ tự các bước xử lý giao thức đã chỉ ra

56

56

Ví dụ 1: Một phiên làm việc của POP3

```
C: <client connects to service port 110>
S: +OK POP3 server ready
   <1896.6971@mailgate.dobbs.org>
C: USER bob
S: +OK bob
C: PASS redqueen
S: +OK bob's maildrop has 2 messages (320 octets)
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: QUIT
S: +OK dewey POP3 server signing off (maildrop
   empty)
C: <client hangs up>
```

57

57

Ví dụ 2: Đăng nhập trên giao thức FTP

```
> ftp 202.191.56.65
C: Connected to 202.91.56.65
S: 220 Servers identifying string
User: tungbt (C: USER tungbt)
S: 331 Password required for tungbt
Password: (C: PASS)
S: 530 Login incorrect
C: ls
S: 530 Please login with USER and PASS
C: USER tungbt
S: 331 Password required for tungbt
Password: (C: PASS)
S: 230 User tungbt logged in
```

58

58

Một số vấn đề

- Có bao nhiêu bên tham gia giao thức? Mỗi bên có giao tiếp với tất cả các bên còn lại không?
- Giao thức là “stateful” hay “stateless”?
 - Stateless: các yêu cầu của client được xử lý độc lập.
 - Không yêu cầu server lưu trữ trạng thái của phiên làm việc
 - Ưu điểm: Đơn giản
 - Hạn chế: cần thêm thông tin đính kèm trong yêu cầu
- Sử dụng UDP hay TCP?
- Giao thức unicast, multicast hay broadcast?
 - Multicast và broadcast: phải sử dụng UDP

59

59

Một số vấn đề (tiếp)

- Có cần thông điệp trả lời?
 - Phát hiện và xử lý mất thông điệp trả lời thế nào?
- Giao thức đơn kết nối hay đa kết nối?
 - Đa kết nối: phải đồng bộ
- Quản lý phiên
- Các vấn đề về an toàn bảo mật: bí mật, xác thực các bên, toàn vẹn thông điệp...
- Xử lý ngoại lệ

60

60

Ví dụ: Ứng dụng sử dụng đa kết nối

- FTP: File Transfer Protocol
- 2 kết nối:
 - Kết nối để gửi lệnh điều khiển
 - Kết nối để truyền file
- Xử lý như thế nào nếu?
 - Trên kết nối điều khiển:
 - STORE a.txt
 - DEL a.txt
 - Trên kết nối truyền file: đang upload file a.txt

61

61

Các bước thiết kế

1. Xác định các dịch vụ cần cung cấp trên ứng dụng
2. Lựa chọn mô hình (client/server, P2P...)
3. Xác định các mục tiêu của giao thức
4. Thiết kế khuôn dạng thông điệp
5. Thứ tự truyền thông điệp và cách thức xử lý thông điệp
6. Tương tác với các giao thức khác

62

62

Thiết kế thông điệp


- Header: bao gồm các trường thông tin mô tả về thông điệp
 - Loại thông điệp
 - Thao tác, lệnh
 - Kích thước phần thân(body)
 - Thông tin của phía tiếp nhận
 - Thông tin về thứ tự của thông điệp
 - Số lần thử lại...
- Body: chứa dữ liệu của ứng dụng(tham số của lệnh, dữ liệu cần truyền)
- Khuôn dạng đơn giản:
 - Type-Value/Data
 - Type-Length-Value/Data



63

63

Thông điệp điều khiển

- Xác định giai đoạn của giao thức
- Thể hiện thông tin điều khiển của giao thức
- Xác định các thông tin của quá trình truyền thông giữa các bên:
 - Khởi tạo, kết thúc phiên
 - Các giai đoạn thực hiện(VD: xác thực, trạng thái xử lý của yêu cầu, trạng thái của quá trình truyền dữ liệu)
 - Phối hợp các bên(báo nhận, yêu cầu phát lại...)
 - Thay đổi của liên kết(khởi tạo liên kết mới, thiết lập lại liên kết)
- Khuôn dạng thông thường: 
 - Command: kích thước cố định hoặc có dấu phân cách với phần tham số

64

64

Thông điệp truyền dữ liệu

- Thông điệp mang theo dữ liệu cần truyền
- Thông thường là đáp ứng cho các yêu cầu
- Dữ liệu cần truyền có thể bị phân mảnh
- Phần tiêu đề thường mô tả:
 - Định dạng của dữ liệu
 - Kích thước của dữ liệu
 - Vị trí của mảnh dữ liệu
 - ...

65

65

Định dạng thông điệp

- Định dạng theo chuỗi byte (byte format)
 - Phần đầu thường là 1 byte quy định kiểu thông điệp
 - Phần dữ liệu: tổ chức thành các trường có kích thước xác định
 - Ưu điểm: hiệu quả truyền cao do phần đầu có kích thước nhỏ
 - Hạn chế: xử lý thông điệp phức tạp
 - Ví dụ: DNS, DHCP
- Định dạng theo chuỗi ký tự
 - Phần đầu thường là các ký tự quy định kiểu thông điệp
 - Phần dữ liệu: thông tin nối thành chuỗi, có thể sử dụng ký tự ngăn cách (delimiter)
 - Ưu điểm: dễ hiểu, linh hoạt, dễ kiểm thử, gỡ lỗi
 - Hạn chế: làm tăng kích thước thông điệp, có thể sẽ phức tạp
- Giải pháp khác: Ép kiểu, Serialisation, JSON, XML...

66

66

Ví dụ: Giao thức đăng nhập/đăng xuất

- Client: Gửi yêu cầu:
 - Đăng nhập: cần gửi thông tin id và password
 - Đăng xuất: Không cần gửi thông tin kèm theo
- Server: Kiểm tra thông tin tài khoản và trạng thái. Gửi thông điệp trả lời tương ứng
- Yêu cầu:
 - Client đăng nhập sai quá 5 lần sẽ bị khóa tài khoản
 - Client đang ở trạng thái đã đăng nhập thì không được đăng nhập tiếp ở một phiên khác.
- Thiết kế khuôn dạng thông điệp giữa client và server

67

67

Client	Server
Yêu cầu đăng nhập: id, password Khuôn dạng: LOGIN id password\n	Kết quả đăng nhập: - Thành công +OK [Thông điệp] - Thất bại + ID không tồn tại + Password không đúng + Đã được đăng nhập + Tài khoản bị khóa + Yêu cầu sai khuôn dạng Khuôn dạng -ERR [Thông báo lỗi]
Yêu cầu đăng xuất: LOGOUT\n	Kết quả đăng xuất: - Thành công +OK [Thông điệp] - Thất bại + Tài khoản chưa đăng nhập + Yêu cầu sai khuôn dạng Khuôn dạng: -ERR [Thông báo lỗi]

68

68

Một thiết kế khác

- Sử dụng định dạng theo byte?

69

69

Mô tả trạng thái

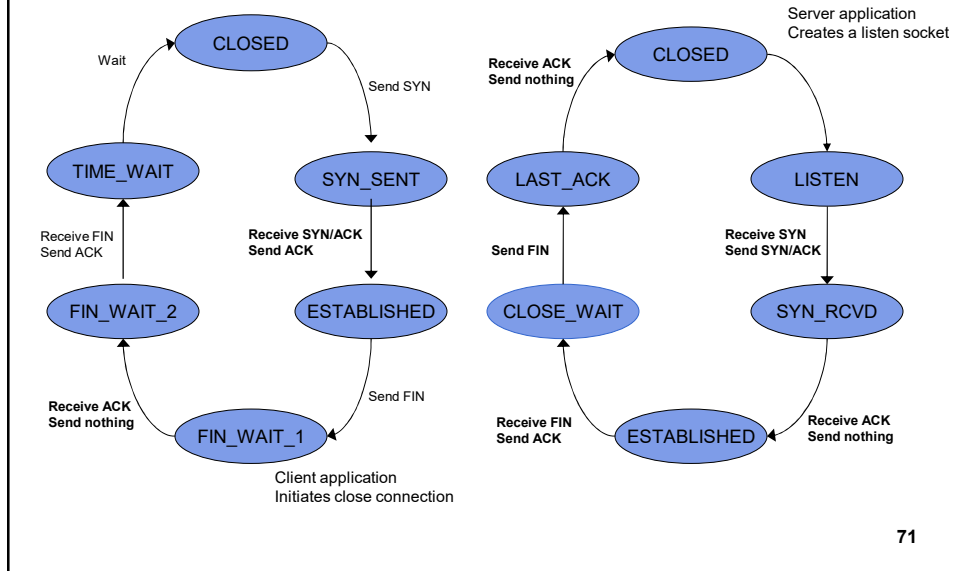
- Sử dụng biểu đồ trạng thái (State Machine Diagram)
- Trạng thái: Tên trạng thái
- Chuyển trạng thái: Trigger[Guard]/[Effect] →
 - Trigger: Nguyên nhân gây chuyển trạng thái (sự kiện, tín hiệu...)
 - Guard: Điều kiện canh giữ
 - Effect: hành động cần thực thi do có chuyển trạng thái
- Lựa chọn/Rẽ nhánh: ◊
- Cách thức khác: Sử dụng bảng mô tả

Trạng thái hiện tại	Chuyển trạng thái		Trạng thái kế tiếp
	Thông điệp nhận	Thông điệp gửi	

70

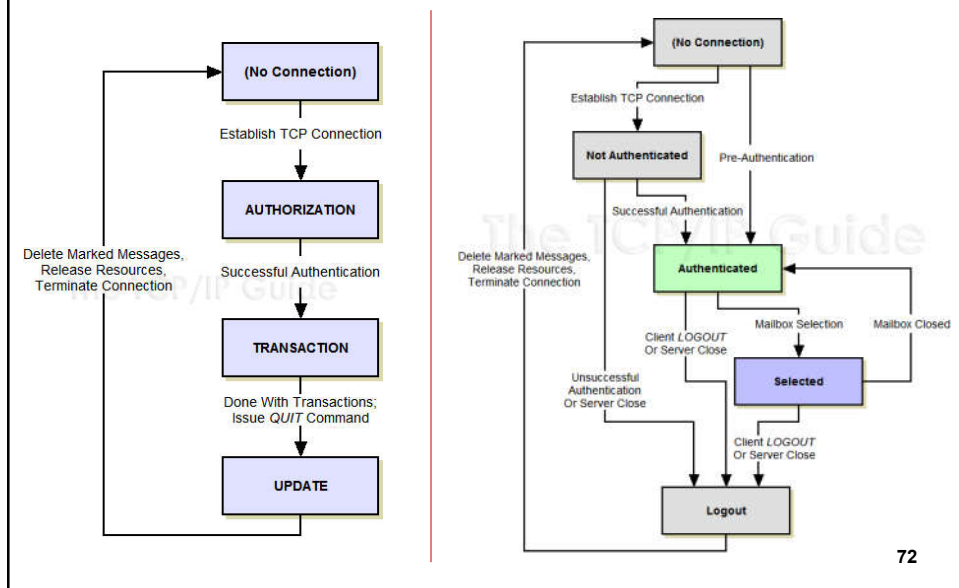
70

Ví dụ: Giao thức TCP



71

Ví dụ: POP3 và IMAP4



72

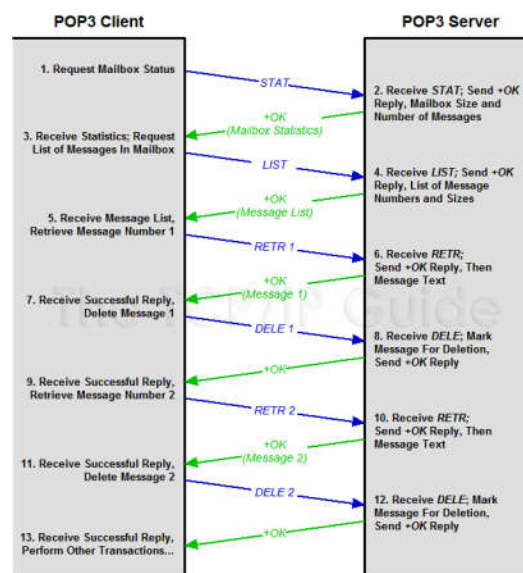
Ví dụ: Giao thức đăng nhập

73

73

Mô tả trình tự của giao thức

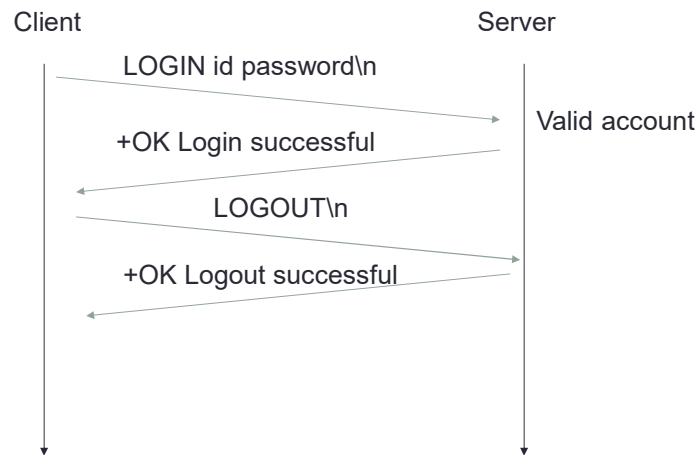
- Mô tả thứ tự các thông điệp được truyền đi trong giao thức
- Sử dụng đường thời gian



74

37

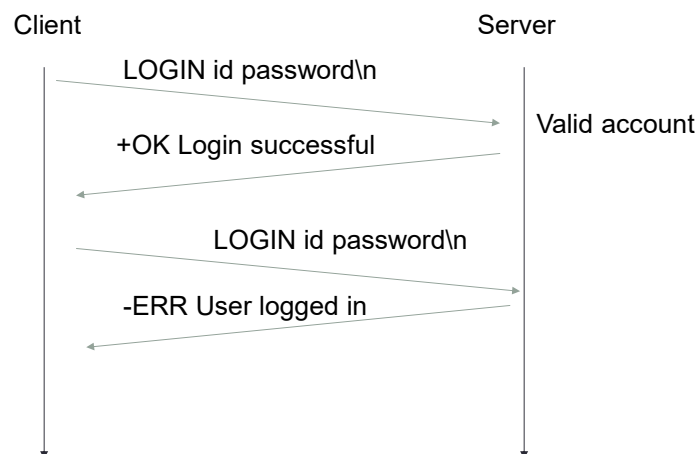
Ví dụ: Giao thức đăng nhập, đăng xuất



75

75

Ví dụ: Giao thức đăng nhập, đăng xuất(2)



76

76

Cài đặt giao thức với ngôn ngữ lập trình

- Khai báo dạng thông điệp, trạng thái

- Dùng số nguyên

```
typedef enum messType {...}
```

hoặc khai báo hằng

- Dùng mẫu ký tự: USER, PASS
 - Kết hợp
- Khuôn dạng thông điệp
 - Dùng cấu trúc: Cần ép kiểu khi gửi và khi nhận
 - Dùng chuỗi ký tự: cần có ký hiệu phân cách giữa các trường
 - Khác: Serialisation, XML, JSON

77

77

Cài đặt cấu trúc thông điệp

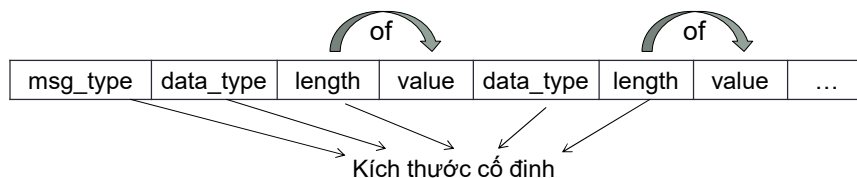
- Dùng kiểu struct

```
struct message{  
    char msg_type[4];  
    int length;  
    char data[8];  
}
```

```
struct message{  
    msg_type type;  
    struct msg_payload payload;  
};
```

```
struct msg_payload{  
    int id;  
    char fullname[30];  
    int age;  
    //...  
}
```

- Sử dụng mảng byte:



78

78

Ví dụ: Thông điệp đăng nhập LOGIN

- Định dạng ký tự: LOGIN username password\n
- Định dạng kiểu byte (dùng mảng)

```
| message_type | message_length | data_type |  
username_length | username | data_type |  
password_length | password
```

message_type(1 byte): = 0 nếu là LOGIN

message_length (2 byte)

data_type (1 byte): = 0 nếu là username, = 1 nếu là password

username_length(1 byte)

password_length(1 byte)

79

79

Ví dụ: Thông điệp đăng nhập LOGIN

- Định dạng kiểu byte (dùng cấu trúc)

```
typedef message{  
    int type;  
    int mess_len;  
    int data_type;  
    ....  
}
```

80

80

Cài đặt giao thức với ngôn ngữ lập trình

- Xử lý thông điệp

```
//receive message
switch (messType){
    case MSG_TYPE1:
    {
        //...
    }
    case MSG_TYPE2:
    {
        //...
        if(data_type == DATA_TYPE1)
        //...
    }
    //...
}
```

Lưu ý: Module hóa chương trình

81

81

Đọc thêm

- 1) <http://theamiableapi.com/2012/03/04/rest-and-the-art-of-protocol-design/>
- 2) <http://xmpp.org/extensions/xep-0134.html#guidelines>
- 3) <http://tools.ietf.org/html/rfc4101>

82

82