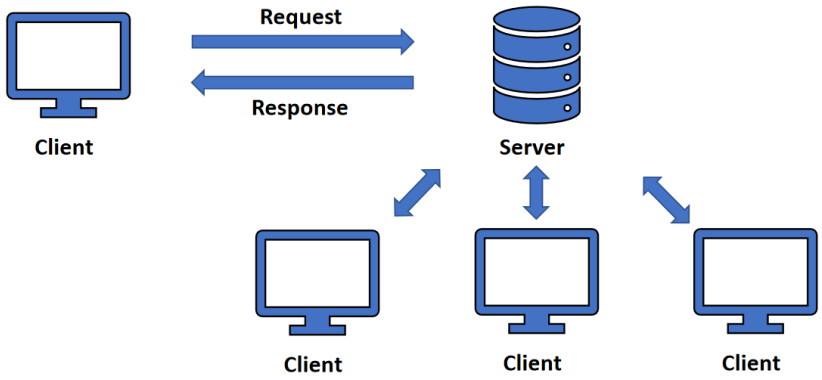
ÔN TẬP
LẬP TRÌNH MẠNG (IT4060)

Nội dung

Chương 1. Giới thiệu về Lập trình mạng



Thư viện được sử dụng: **Socket API**

- Thư viện đa nền tảng, được hỗ trợ bởi các chương trình dịch trong các ngôn ngữ và hệ điều hành khác nhau.
- Thường sử dụng cùng với C/C++
- Cho hiệu năng cao nhất

Công cụ hỗ trợ: **Netcat**

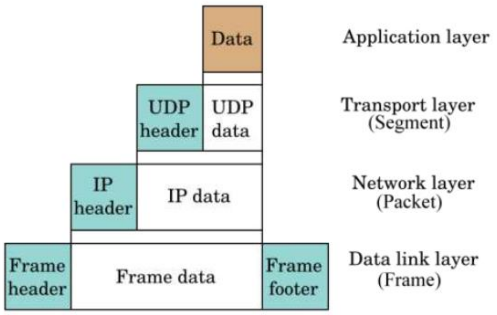
- Sử dụng Netcat để gửi nhận dữ liệu đơn giản
- Netcat là một tiện ích mạng rất đa năng.
- Có thể sử dụng như TCP server:
  - nc -v –l –p <cổng đợi kết nối>
- Có thể sử dụng như TCP client:
  - nc –v <ip/tên miền> <cổng>
- Sử dụng như UDP receiver:
  - nc –v –l –u –p <cổng đợi kết nối>
- Sử dụng như UDP sender:
  - nc –v –u <ip/tên miền> <cổng>

1.1. Nhắc lại một số kiến thức Mạng máy tính

a. Giao thức TCP/IP

- TCP/IP: Transmission Control Protocol / Internet Protocol.
- Là bộ giao thức truyền thông được sử dụng trên Internet và hầu hết các mạng thương mại.
- Được chia thành các tầng gồm nhiều giao thức, thuận tiện cho việc quản lý và phát triển.
- Là thể hiện đơn giản hóa của mô hình lý thuyết OSI (7 tầng).

- Gồm bốn tầng
    - Tầng ứng dụng – Application Layer.
    - Tầng giao vận – Transport Layer.
    - Tầng Internet – Internet Layer.
    - Tầng truy nhập mạng – Network Access Layer.
- Tầng ứng dụng
    - Đóng gói dữ liệu người dùng theo giao thức riêng và chuyển xuống tầng dưới.
    - Các giao thức thông dụng: HTTP, FTP, SMTP, POP3, DNS, SSH, IMAP...
    - Việc lập trình mạng sẽ xây dựng ứng dụng tuân theo một trong các giao thức ở tầng này hoặc giao thức do người phát triển tự định nghĩa
- Tầng giao vận
    - Cung cấp dịch vụ truyền dữ liệu giữa ứng dụng - ứng dụng.
    - Đơn vị dữ liệu là các đoạn (segment, datagram)
    - Các giao thức ở tầng này: TCP, UDP.
    - Việc lập trình mạng sẽ sử dụng dịch vụ do các giao thức ở tầng này cung cấp để truyền dữ liệu
- Tầng Internet
    - Định tuyến và truyền các gói tin liên mạng.
    - Cung cấp dịch vụ truyền dữ liệu giữa máy tính – máy tính trong cùng nhánh mạng hoặc giữa các nhánh mạng.
    - Đơn vị dữ liệu là các gói tin (packet).
    - Các giao thức ở tầng này: IPv4, IPv6
    - Việc lập trình ứng dụng mạng sẽ rất ít khi can thiệp vào tầng này, trừ khi phát triển một giao thức liên mạng mới.
- Tầng truy nhập mạng
    - Cung cấp dịch vụ truyền dữ liệu giữa các nút mạng trên cùng một nhánh mạng vật lý.
    - Đơn vị dữ liệu là các khung (frame).
    - Phụ thuộc rất nhiều vào phương tiện kết nối vật lý.
    - Các giao thức ở tầng này đa dạng: MAC, LLC, ADSL, 802.11...
    - Việc lập trình mạng ở tầng này là xây dựng các trình điều khiển phần cứng tương ứng, thường do nhà sản xuất thực hiện
- Dữ liệu gửi đi qua mỗi tầng sẽ được thêm phần thông tin điều khiển (header).
- Dữ liệu nhận được qua mỗi tầng sẽ được bóc tách thông tin điều khiển.



- Internat Protocol: Giao thức mạng thông dụng nhất trên thế giới
- Chức năng
    - Định địa chỉ các máy chủ
    - Định tuyến các gói dữ liệu trên mạng
- Bao gồm 2 phiên bản: IPv4 và IPv6
- Thành công của Internet là nhờ IPv4
- Được hỗ trợ trên tất cả các hệ điều hành
- Là công cụ sử dụng để lập trình ứng dụng mạng

b. Giao thức IPv4

- Được IETF công bố dưới dạng RFC 791 vào 9/1981.
- Phiên bản thứ 4 của họ giao thức IP và là phiên bản đầu tiên phát hành rộng rãi.
- Sử dụng trong hệ thống chuyển mạch gói.
- Truyền dữ liệu theo kiểu Best-Effort: không đảm bảo tính trật tự, trùng lặp, tin cậy của gói tin.
- Kiểm tra tính toàn vẹn của dữ liệu qua checksum

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

Một số trường cần quan tâm:
Version (4 bit): có giá trị là 4 với IPv4
IHL – Internet Header Length (4 bit): chiều dài của header, tính bằng số từ nhớ 32 bit
Total Length (16 bit): kích thước của gói tin (theo bytes) bao gồm cả header và data
Protocol: giao thức được sử dụng ở tầng trên (nằm trong phần data)
Source IP Address: địa chỉ IP nguồn
Destination IP Address: địa chỉ IP đích

Địa chỉ IPv4

- Sử dụng 32 bit để đánh địa chỉ các máy tính trong mạng.
- Bao gồm: phần mạng và phần host.
- Số địa chỉ tối đa: 232 ~ 4,294,967,296.
- Dành riêng một vài dải đặc biệt không sử dụng.
- Chia thành bốn nhóm 8 bít (octet).

Các lớp địa chỉ IPv4

- Có năm lớp địa chỉ: A, B, C, D, E.
- Lớp A, B, C: trao đổi thông tin thông thường.
- Lớp D: multicast
- Lớp E: để dành

Lớp	MSB	Địa chỉ đầu	Địa chỉ cuối
A	0xxx	0.0.0.0	127.255.255.255
B	10xx	128.0.0.0	191.255.255.255
C	110x	192.0.0.0	223.255.255.255
D	1110	224.0.0.0	239.255.255.255
E	1111	240.0.0.0	255.255.255.255

Mặt nạ mạng (Network Mask)

- Phân tách phần mạng và phần host trong địa chỉ IPv4.
- Sử dụng trong bộ định tuyến để tìm đường đi cho gói tin.
- Với mạng có dạng

Network	Host
192.168.0.	1
11000000.10101000.00000000.	00000001

- Biểu diễn theo dạng /n
  - n là số bit dành cho phần mạng.
  - Ví dụ: 192.168.0.1/24
- Biểu diễn dưới dạng nhị phân
  - Dùng 32 bit đánh dấu, bít dành cho phần mạng là 1, cho phần host là 0.
  - Ví dụ: 11111111.11111111.11111111.00000000 hay 255.255.255.0
- Biểu diễn dưới dạng Hexa

- Dùng số Hexa: 0xFFFFFF00
- Ít dùng

Số lượng địa chỉ trong mỗi mạng:

- Mỗi mạng sẽ có n bit dành cho phần mạng, 32-n bit dành cho phần host.
- Phân phối địa chỉ trong mỗi mạng:
  - 01 địa chỉ mạng (các bit phần host bằng 0).
  - 01 địa chỉ quảng bá (các bit phần host bằng 1).
  - 2^{32-n-2} địa chỉ gán cho các máy trạm (host).
- Với mạng 192.168.0.1/24
  - Địa chỉ mạng: 192.168.0.0
  - Địa chỉ quảng bá: 192.168.0.255
  - Địa chỉ host: 192.168.0.1 - 192.168.0.254

Các dải địa chỉ đặc biệt:

Địa chỉ	Diễn giải
10.0.0.0/8	Mạng riêng
127.0.0.0/8	Địa chỉ loopback
172.16.0.0/12	Mạng riêng
192.168.0.0/16	Mạng riêng
224.0.0.0/4	Multicast
240.0.0.0/4	Dự trữ

Dải địa chỉ cục bộ:

Tên	Dải địa chỉ	Số lượng	Mô tả mạng	Viết gọn
Khối 24-bit	10.0.0.0 – 10.255.255.255	16,777,216	Một dải trọn vẹn thuộc lớp A	10.0.0.0/8
Khối 20-bit	172.16.0.0 – 172.31.255.255	1,048,576	Tổ hợp từ mạng lớp B	172.16.0.0/12
Khối 16-bit	192.168.0.0 – 192.168.255.255	65,536	Tổ hợp từ mạng lớp C	192.168.0.0/16

c. Giao thức IPv6

- IETF đề xuất năm 1998.
- Khắc phục vấn đề thiếu địa chỉ của IPv4.
- Đang dần phổ biến và chưa thể thay thế hoàn toàn IPv4.
- Sử dụng 128 bit để đánh địa chỉ các thiết bị, dưới dạng các cụm số hexa phân cách bởi dấu :
  - Ví dụ: FEDC:BA98:768A:0C98:FEBA:CB87:7678:1111

Quy tắc rút gọn địa chỉ IPv6

- Cho phép bỏ các số 0 nằm trước mỗi nhóm (octet).

- Thay bằng số 0 cho nhóm có toàn số 0.
- Thay bằng dấu "::" cho các nhóm liên tiếp nhau có toàn số 0.
- Chú ý: Dấu "::" chỉ sử dụng được 1 lần trong toàn bộ địa chỉ IPv6
    - Ví dụ: 1080:0000:0000:0070:0000:0989:CB45:345F Có thể viết tắt thành 1080::70:0:989:CB45:345F hoặc 1080:0:0:70::989:CB45:345F

Cổng (Port)

- Một số nguyên duy nhất trong khoảng 0 – 65535 tương ứng với một kết nối của ứng dụng.
- TCP sử dụng cổng để chuyển dữ liệu tới đúng ứng dụng hoặc dịch vụ.
- Một ứng dụng có thể mở nhiều kết nối => có thể sử dụng nhiều cổng.
- Một số cổng thông dụng: HTTP(80), FTP(21, 20), SMTP(25), POP3(110), HTTPS(443)...

Giao thức TCP

- Giao thức lõi chạy ở tầng giao vận.
- Chạy bên dưới tầng ứng dụng và trên nền IP
- Cung cấp dịch vụ truyền dữ liệu theo dòng tin cậy giữa các ứng dụng.
- Được sử dụng bởi hầu hết các ứng dụng mạng.
- Chia dữ liệu thành các gói nhỏ, thêm thông tin kiểm soát và gửi đi trên đường truyền.
- Lập trình mạng sẽ sử dụng giao thức này để trao đổi thông tin.

Đặc tính của TCP

- Hướng kết nối (connection oriented)
    - Hai bên phải thiết lập kênh truyền trước khi truyền dữ liệu.
    - Được thực hiện bởi quá trình gọi là bắt tay ba bước (three ways handshake).
- Truyền dữ liệu theo dòng (stream oriented): tự động phân chia dòng dữ liệu thành các đoạn nhỏ để truyền đi, tự động ghép các đoạn nhỏ thành dòng dữ liệu và gửi trả ứng dụng.
- Đúng trật tự (ordering guarantee): dữ liệu gửi trước sẽ được nhận trước
- Tin cậy, chính xác: thông tin gửi đi sẽ được đảm bảo đến đích, không dư thừa, sai sót...
- Độ trễ lớn, khó đáp ứng được tính thời gian thực.

Header của TCP

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)														
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

Một số trường cần quan tâm:
Source port: cổng gửi dữ liệu
Destination port: cổng nhận dữ liệu
Data offset: độ dài TCP header tính bằng số từ 32-bit

Các dịch vụ trên nền TCP

- Rất nhiều dịch vụ chạy trên nền TCP: FTP (21), HTTP (80), SMTP (25), SSH (22), POP3 (110), VNC (4899)...

Giao thức UDP:

- Cũng là giao thức lõi trong TCP/IP.
- Cung cấp dịch vụ truyền dữ liệu giữa các ứng dụng.
- UDP chia nhỏ dữ liệu ra thành các datagram

Đặc tính của UDP:

- Không cần thiết lập kết nối trước khi truyền (connectionless).
- Nhanh, chiếm ít tài nguyên dễ xử lý.
- Sử dụng trong các ứng dụng khắt khe về mặt thời gian, chấp nhận sai sót: audio, video, game ...
- Hạn chế:
  - Không có cơ chế báo gửi (report).
  - Không đảm báo trật tự các datagram (ordering).
  - Không phát hiện được mất mát hoặc trùng lặp thông tin (loss, duplication).

Header của UDP:

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Một số trường cần quan tâm:
Source port: cổng gửi dữ liệu
Destination port: cổng nhận dữ liệu
Length: độ dài của gói tin UDP (header luôn có kích thước cố định là 8 bytes)

Các dịch vụ trên nền UDP

- Phân giải tên miền: DNS (53)
- Streamming: MMS, RTSP...
- Game

1.2. Nhắc lại một số kiến thức lập trình C

Cấp phát bộ nhớ động:

- Hàm **malloc()** cấp phát bộ nhớ.
- Hàm **realloc()** cấp phát lại bộ nhớ đã cấp phát trướcđó.
- Hàm **free()** giải phóng bộ nhớ đã cấp phát.
  - ⇨ Cần khai báo tệp tiêu đề **stdlib.h**
- Hàm **memset()** thiết lập nội dung cho vùng nhớ
- Hàm **memcpy()** sao chép nội dung giữa 2 vùng nhớ
  - ⇨ Cần khai báo tệp tiêu đề **string.h**

Xử lý xâu ký tự:

- Hàm **strlen()** xác định độ dài xâu
- Hàm **strcpy()/strncpy()** sao chép xâu
- Hàm **strcmp()/strncmp()** so sánh xâu
- Hàm **strcat()/strncat()** ghép xâu
- Hàm **strchr()** tìm ký tự trong xâu
- Hàm **strstr()** tìm xâu con trong xâu
- Hàm **strtok()** tách xâu
- Hàm **sscanf()** đọc dữ liệu có định dạng trong xâu
- Hàm **sprintf()** ghi dữ liệu có định dạng vào xâu
  - ⇨ Cần khai báo tệp tiêu đề **string.h**

Đọc, ghi file:

- Khai báo con trỏ file: **FILE *f**;
- Mở file: hàm **fopen()** mở file với các chế độ khác nhau
- Đọc dữ liệu từ file:
  - Hàm **fread()**: đọc theo từng mảng
  - Hàm **fscanf()**: đọc dữ liệu định dạng
  - Hàm **fgets()**: đọc một dòng
  - Hàm **fgetc()**: đọc một ký tự
- Ghi dữ liệu vào file:
  - Hàm **fwrite():** ghi vào một mảng
  - Hàm **fprintf():** ghi dữ liệu định dạng
- Xác định vị trí con trỏ file: hàm **ftell()**
- Di chuyển con trỏ file: hàm **fseek()**
- Kiểm tra kết thúc file: hàm **feof()**
- Đóng file: hàm **fclose()**

Truyền tham số dòng lệnh:

```
int main(int argc, char *argv[])
{
    // argc là số tham số đã truyền
    // argv là mảng các tham số kiểu chuỗi
    // tham số đầu tiên là tên file thực thi
}
```

Chương 2. Lập trình socket cơ bản

2.1. Khái niệm socket

- Socket là điểm cuối (end-point) trong liên kết truyền thông hai chiều (two-way communication) biểu diễn kết nối giữa client – server.
- Các lớp socket được ràng buộc với một cổng port (thể hiện là một con số cụ thể) để các tầng TCP (TCP Layer) có thể định danh ứng dụng mà dữ liệu sẽ được gửi tới.
- Socket là giao diện lập trình mạng được hỗ trợ bởi nhiều ngôn ngữ, hệ điều hành khác nhau.
- Socket có thể được sử dụng để chờ các kết nối trong ứng dụng server hoặc để thiết lập kết nối trong ứng dụng client.

2.2. Cấu trúc địa chỉ socket

- Socket cần được gắn địa chỉ để thực hiện chức năng truyền nhận dữ liệu trên mạng.
- Cấu trúc địa chỉ lưu trữ địa chỉ IP và cổng.
- Các cấu trúc địa chỉ:

```
struct sockaddr => Mô tả địa chỉ nói chung
struct sockaddr_in => Mô tả địa chỉ IPv4
struct sockaddr_in6 => Mô tả địa chỉ IPv6
```

Các hàm chuyển đổi địa chỉ:

- Cần khai báo tệp **<arpa/inet.h>**
- Chuyển đổi địa chỉ IP dạng xâu sang số nguyên 32 bit (IPv4)
  - **in_addr_t inet_addr (const char *cp)**
  - ⇨ Hàm trả về địa chỉ dạng số nguyên, -1 nếu gặp lỗi
- Chuyển đổi địa chỉ IP dạng xâu sang cấu trúc in_addr
  - **int inet_aton (const char *cp, struct in_addr *inp)**
  - ⇨ Hàm trả về 1 nếu thành công, 0 nếu gặp lỗi
- Chuyển đổi địa chỉ từ dạng in_addr sang dạng xâu (IPv4)
  - **char *inet_ntoa (struct in_addr in)**
  - ⇨ Hàm trả về chuỗi ký tự chứa địa chỉ
- Chuyển đổi từ dạng số sang dạng xâu (cho IPv4 và IPv6)
  - **const char *inet_ntop (int af, const void *cp, char *buf, socklen_t len)**
  - ⇨ hàm trả về xâu ký tự chứa địa chỉ, trả về NULL nếu gặp lỗi

- Chuyển đổi từ dạng xâu sang dạng số (cho IPv4 và IPv6)
    - **int inet_pton (int af, const char *cp, void *buf)**
    - ⇨ Hàm trả về 1 nếu thành công, 0 nếu xâu ký tự không hợp lệ, -1 nếu gặp lỗi khác
- Chuyển đổi little-endian => big-endian (network order)
    - // Chuyển đổi 4 byte từ little-endian=>big-endian
        - **uint32_t htonl (uint32_t hostlong)**
    - // Chuyển đổi 2 byte từ little-endian=>big-endian
        - **uint16_t htons (uint16_t hostshort)**
- Chuyển đổi big-endian => little-endian (host order)
    - // Chuyển 4 byte từ big-endian=>little-endian
        - **uint32_t ntohl (uint32_t netlong)**
    - // Chuyển 2 byte từ big-endian=>little-endian
        - **uint16_t ntohs (uint16_t netshort)**

Phân giải tên miền:

- Địa chỉ của máy đích được cho dưới dạng tên miền
- Ứng dụng cần thực hiện phân giải tên miền để có địa chỉ IP thích hợp
- Hàm **getaddrinfo()** sử dụng để phân giải tên miền ra các địa chỉ IP
- Cần thêm tệp tiêu đề **netdb.h**
- Giá trị trả về
    - Thành công: 0
    - Thất bại: mã lỗi, sử dụng hàm gai_strerror() để in ra thông báo lỗi
- Giải phóng: hàm **freeaddrinfo()**

VD:

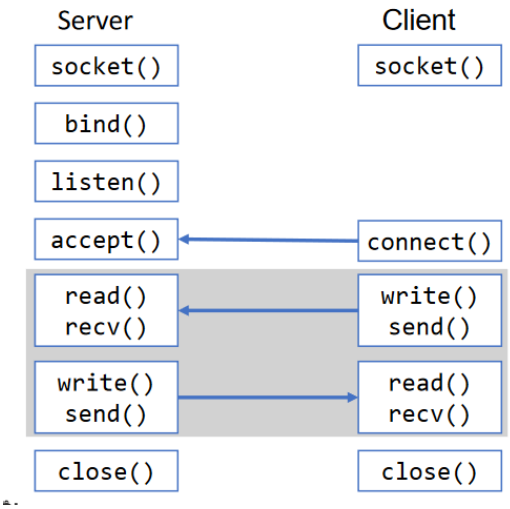
```
struct addrinfo *res, *p;
int ret = getaddrinfo(argv[1], "http", NULL, &res);
if (ret != 0) {
    printf("Failed to get IP\n");
    return 1;
}

p = res;
while (p != NULL) {
    if (p->ai_family == AF_INET) {
        printf("IPv4\n");
        struct sockaddr_in addr;
        memcpy(&addr, p->ai_addr, p->ai_addrlen);
        printf("IP: %s\n", inet_ntoa(addr.sin_addr));
    } else if (p->ai_family == AF_INET6) {
        printf("IPv6\n");
        char buf[64];
        struct sockaddr_in6 addr6;
        memcpy(&addr6, p->ai_addr, p->ai_addrlen);
        printf("IP: %s\n", inet_ntop(p->ai_family, &addr6.sin6_addr,
buf, sizeof(addr6)));
    }
    p = p->ai_next;
```

```
}
freeaddrinfo(res);
```

2.3. Ứng dụng TCP server/client

- Các hàm được sử dụng để tạo ứng dụng server/client hoạt động theo giao thức TCP.
- Cần khai báo thư viện **<sys/socket.h>**



- Cú pháp:

```
#include <sys/socket.h>
int socket (
      int domain, // Giao thức AF_INET hoặc AF_INET6
      int type, // Kiểu socket SOCK_STREAM hoặc SOCK_DGRAM
      int protocol // Giao thức IPPROTO_TCP hoặc IPPROTO_UDP
)
```

=> Hàm trả về giá trị mô tả của socket (kiểu int) nếu thành công, trả về -1 nếu gặp lỗi (biến errno chứa mã lỗi, cần khai báo errno.h để truy nhập errno)

- VD:

```
// Tạo socket TCP
int s1 = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s1 == -1) {
      printf("Không tạo được socket\n");
      return 1;
}

// Tạo socket UDP
int s2 = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (s2 == -1) {
      printf("Không tạo được socket\n");
      return 1;
}
```

- Đa số các hàm trả về -1 nếu gặp lỗi.
- Biến errno trả về mã lỗi xảy ra gần nhất (cần khai báo thư viện **errno.h**)
- Hàm **strerror(int errnum)** trả về chuỗi ký tự mô tả mã lỗi (thư viện **string.h**)

- Hàm **perror(const char *s)** in ra chuỗi ký tự mô tả mã lỗi gần nhất, s là chuỗi ký tự tiền tố, có thể bằng NULL (thư viện **stdio.h**).

```c
int listener = socket(AF_INET, SOCK_STREAM, -1);
if (listener != -1)
  printf("Socket created: %d\n", listener);
else {
  printf("Failed to create socket: %d - %s\n", errno, strerror(errno));
  perror("socket() failed");
  exit(1);
}
```

Truyền dữ liệu sử dụng TCP - Ứng dụng server

- Tạo socket qua hàm **socket()**
- Gắn socket vào một giao diện mạng thông qua hàm **bind()**
- Chuyển socket sang trạng thái đợi kết nối qua hàm **listen()**
- Chấp nhận kết nối từ client thông qua hàm **accept()**
- Gửi dữ liệu tới client thông qua hàm **send()/write()**
- Nhận dữ liệu từ client thông qua hàm **recv()/read()**
- Đóng socket khi việc truyền nhận kết thúc bằng hàm **close()**

TCP_Server:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>

#define MAX_CLIENT 5
#define MAX_BUF_SIZE 1024

int main(int argc, char *argv[])
{
    // Kiểm tra đầu vào
    ...

    // Tạo socket
    int server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    ...
```

```
    // Thiết lập thông tin địa chỉ cho socket
    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //
inet_addr("127.0.0.1");
    server_addr.sin_port = htons(atoi(argv[1]));

    // Gán địa chỉ cho socket
    int ret = bind(server, (struct sockaddr *)&server_addr,
sizeof(server_addr));
    ...

    // Lắng nghe kết nối từ client
    ret = listen(server, MAX_CLIENT);
    ...

    while (1)
    {
        // Chấp nhận kết nối từ client
        struct sockaddr_in client_addr;
        memset(&client_addr, 0, sizeof(client_addr));
        socklen_t client_addr_len = sizeof(client_addr);
        int client = accept(server, (struct sockaddr *)&client_addr,
&client_addr_len);
        ...

        // Nhận dữ liệu từ client
        char buf[MAX_BUF_SIZE];
        memset(buf, 0, MAX_BUF_SIZE);
        int bytes_received = recv(client, buf, MAX_BUF_SIZE, 0);
        if (bytes_received < 0){
            ...
        } else if (bytes_received == 0){
            printf("Client from %s:%d disconnected\n\n",
inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
        } else {
            if (bytes_received < sizeof(buf)){
                buf[bytes_received] = '\0'; // buf[strcspn(buf, "\n")] =
0;
            }
            puts(buf);
            send(client, buf, strlen(buf), 0);
            ...
        }

        // Đóng kết nối với client
        close(client);
    }

    // Đóng socket
    close(server);
    return 0;
}
```

Truyền dữ liệu sử dụng TCP - Ứng dụng client

- Tạo socket qua hàm **socket()**
- Điền thông tin về server vào cấu trúc **sockaddr_in**

- Kết nối tới server qua hàm **connect()**
- Gửi dữ liệu tới server thông qua hàm **send()**
- Nhận dữ liệu từ server thông qua hàm **recv()**
- Đóng socket khi việc truyền nhận kết thúc bằng hàm **close()**

TCP_Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>

#define MAX_BUF_SIZE 1024

int main(int argc, char *argv[])
{
    // Kiểm tra đầu vào
    ...

    // Thiết lập thông tin địa chỉ cho socket
    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(argv[1]); //
inet_addr("127.0.0.1");
    server_addr.sin_port = htons(atoi(argv[2]));

    // Tạo socket
    int client = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    ...

    // Kết nối đến server
    int ret = connect(client, (struct sockaddr *)&server_addr,
sizeof(server_addr));
    ...

    // Nhận dữ liệu từ bàn phím và gửi đến server
    char buf[MAX_BUF_SIZE];
    memset(buf, 0, MAX_BUF_SIZE);

    // Nhận dữ liệu từ bàn phím
    printf("Enter a message: ");
    fgets(buf, MAX_BUF_SIZE, stdin);

    // Gửi tin nhắn đến server
    int bytes_sent = (send(client, buf, strlen(buf), 0);
    ...

    // Đóng socket
    close(client);

    return 0;
}
```

2.4. Ứng dụng UDP Sender/Receiver

- Giao thức UDP là giao thức không kết nối (connectionless)
- Ứng dụng không cần phải thiết lập kết nối trước khi gửi tin.
- Ứng dụng có thể nhận được tin từ bất kỳ máy tính nào trong mạng.
- Trình tự gửi thông tin ở bên gửi như sau:

- Trình tự nhận thông tin ở bên nhận như sau:

UDP_Sender:

```
// Tạo socket theo giao thức UDP
int sender = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
char *msg = "Hello. I am sending a message.\n";

// Khai báo địa chỉ bên nhận
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_port = htons(9090);

// Gửi tin nhắn
char buf[256];
while (1) {
    printf("Enter message: ");
    fgets(buf, sizeof(buf), stdin);
    sendto(sender, buf, strlen(buf), 0, (struct sockaddr *)&addr,
sizeof(addr));
}
```

UDP_Receiver:

```
// Tạo socket theo giao thức UDP
int receiver = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// Khai báo địa chỉ bên nhận
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(9090);
bind(receiver, (struct sockaddr *)&addr, sizeof(addr));

// Nhận tin nhắn
char buf[16];
while (1) {
    int ret = recvfrom(receiver, buf, sizeof(buf), 0, NULL, NULL);
    ...
```

```
        buf[ret] = 0;
        printf("%d - %s\n", ret, buf);
    }
```

Chú ý:

- Lệnh nhận dữ liệu:
  - **recv(client, buf, sizeof(buf), 0);**
  - **recvfrom(client, buf, sizeof(buf), 0, NULL, NULL);**
  - ⇨ Hai lệnh này tương đương nhau, có thể sử dụng thay thế
- Lệnh truyền dữ liệu:
  - **send(client, buf, strlen(buf), 0);**
  - **sendto(client, buf, strlen(buf), 0, NULL, 0);**
  - ⇨ Hai lệnh này tương đương nhau, có thể sử dụng thay thế

Chương 3. Các kiến trúc client-server

3.1 Các chế độ hoạt động của socket

Chế độ đồng bộ (blocking)

- Là chế độ mà các hàm vào ra sẽ chặn thread đến khi thao tác vào ra hoàn tất (các hàm vào ra sẽ không trở về cho đến khi thao tác hoàn tất).
- Là chế độ mặc định của socket
- Các hàm ảnh hưởng:
  - accept()
  - connect()
  - send()
  - recv()
  - …
- Thích hợp với các ứng dụng xử lý tuần tự. Không nên gọi các hàm blocking khi ở thread xử lý giao diện (GUI thread).
- Ví dụ: thread bị chặn bởi hàm recv() thì không thể thực hiện các công việc khác.

Chế độ bất đồng bộ (non-blocking)

- Là chế độ mà các thao tác vào ra sẽ trở về nơi gọi ngay lập tức và tiếp tục thực thi thread. Kết quả của thao tác vào ra sẽ được thông báo cho chương trình dưới một cơ chế đồng bộ nào đó.
- Các hàm vào ra bất đồng bộ sẽ trả về mã lỗi EWOULDBLOCK (EAGAIN - 11) nếu thao tác đó không thể hoàn tất ngay và mất thời gian đáng kể (chấp nhận kết nối, nhận dữ liệu, gửi dữ liệu...). Đây là điều hoàn toàn bình thường.
- Có thể sử dụng trong thread xử lý giao diện của ứng dụng.
- Thích hợp với các ứng dụng hướng sự kiện

Chuyển socket sang chế độ bất đồng bộ

- Áp dụng cho socket chờ kết nối và socket truyền nhận dữ liệu.

- Hàm **ioctl()** được sử dụng để chuyển socket sang trạng thái bất đồng bộ
- Hàm **fcntl()** cũng được sử dụng để thay đổi chế độ hoạt động của socket.

```
// Sử dụng ioctl()
#include <sys/ioctl.h>
unsigned long ul = 1;
ioctl(socket_fd, FIONBIO, &ul);

// Sử dụng fcntl()
#include <fcntl.h>
fcntl(socket_fd, F_SETFL, O_NONBLOCK);
```

- ***VD: Socket chờ kết nối – Lệnh accept()***

```
// Chap nhan ket noi
int client = accept(listener, NULL, NULL);
if (client == -1) {
    if (errno != EWOULDBLOCK) {
        printf("accept() failed.\n");
        exit(1);
    } else {
        // Lỗi do thao tác vào ra chưa hoàn tất. Không cần xử lý gì
thêm.
    }
} else {
    printf("New client connected: %d\n", client);
    clients[numClients++] = client;
    ul = 1;
    ioctl(client, FIONBIO, &ul);
}
```

- ***VD: Socket truyền nhận dữ liệu – Lệnh recv()***

```
int ret = recv(clients[i], buf, sizeof(buf), 0);
if (ret == -1) {
    if (errno != EWOULDBLOCK) {
        printf("recv() failed.\n");
        continue;
    } else {
        // Lỗi do thao tác vào ra chưa hoàn tất. Không cần xử lý gì
thêm.
    }
} else if (ret == 0) {
    printf("client disconnected.\n");
    close(clients[i]);
    continue;
} else {
    // Xử lý dữ liệu nhận được
}
```

3.2. Iteractive server

- Server xử lý tuần tự yêu cầu từ các client.
- Không phù hợp trong việc xử lý đồng thời nhiều kết nối.
- Mỗi client gửi 1 yêu cầu, server xử lý xong trả lại kết quả cho client và chuyển sang client khác.

3.3. Multiplexing

- Là cơ chế quản lý nhiều kết nối client chỉ trong một tiến trình của server.

- Ứng dụng sử dụng mảng để quản lý các socket, hệ thống sẽ cho biết socket nào cần được phục vụ hoặc có yêu cầu kết nối mới.
- Chỉ có duy nhất 1 kết nối được phục vụ tại 1 thời điểm.
- Sử dụng hàm **select()** để thăm dò các kết nối.

Hàm select():

- Hoạt động ở chế độ đồng bộ.
- Được sử dụng để chờ các sự kiện và trả về kết quả khi một hoặc nhiều sự kiện xảy ra hoặc đã qua một khoảng thời gian (timeout).
- Được sử dụng để thăm dò các sự kiện của socket (gửi dữ liệu, nhận dữ liệu, kết nối thành công, yêu cầu kết nối, …)
- Hỗ trợ nhiều kết nối.
- Có thể xử lý tập trung tất cả các socket trong cùng một luồng.
- Xử lý các sự kiện tuần tự, tại một thời điểm chỉ xử lý một sự kiện của 1 socket.

Cú pháp hàm select():

```
#include <sys/select.h>
int select (int nfds, fd_set *readfds,
            fd_set *writefds, fd_set *exceptfds,
            struct timeval *timeout)
```

- nfds: giá trị socket lớn nhất cộng 1 được gán vào 3 tập hợp (không vượt quá FD_SETSIZE)
- readfds: tập các socket chờ sự kiện đọc
- writefds: tập các socket chờ sự kiện ghi
- exceptfds: tập các socket chờ sự kiện ngoại lệ hoặc lỗi
- timeout: thời gian chờ các sự kiện
  - NULL – chờ với thời gian vô hạn
  - 0 – không chờ sự kiện nào
  - > 0 – chờ với thời gian xác định

Kết quả trả về của hàm select():

- Giá trị trả về:
  - Thành công: số lượng socket xảy ra sự kiện
  - Hết thời gian chờ: 0
  - Bị lỗi: -1
- Điều kiện thành công của hàm **select()**
  - Một trong các socket của tập **readfds** nhận dữ liệu hoặc kết nối bị đóng, bị hủy hoặc có yêu cầu kết nối.
  - Một trong các socket của tập **writefds** có thể gửi dữ liệu, hoặc hàm connect thành công trên socket non-blocking.
  - Một trong các socket của tập **exceptfds** nhận dữ liệu OOB, hoặc connect thất bại.
- Các tập **readfds, writefds, exceptfds** có thể NULL, nhưng không thể cả 3 cùng NULL.

Cấu trúc fd_set:

- fd_set là kiểu dữ liệu cấu trúc chứa mảng các bit mô tả các socket được gắn vào để thăm dò sự kiện.
- Các macro được sử dụng để thực hiện các thao tác với tập fd_set:

```
void FD_CLR(int fd, fd_set *set); => Xóa fd ra khỏi tập set
int FD_ISSET(int fd, fd_set *set); => Kiểm tra sự kiện của fd xảy ra
với tập set
void FD_SET(int fd, fd_set *set); => Gắn fd vào tập set
void FD_ZERO(fd_set *set); => Xóa tất cả các socket khỏi tập set
```

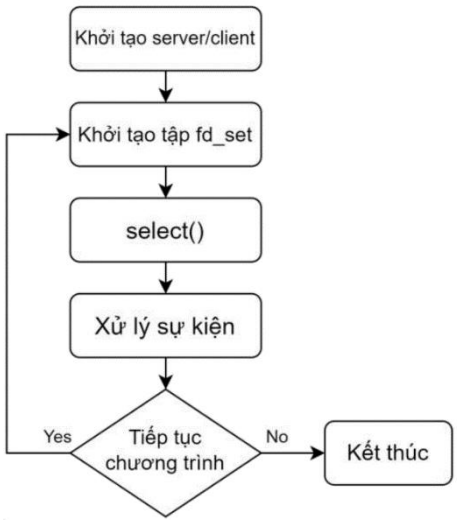
- Hàm select() thay đổi giá trị của tập fd_set sau khi trả về kết quả để chỉ ra socket nào có sự kiện => Cần khởi tạo lại tập fd_set nếu gọi select() nhiều lần.

Cấu trúc timeval

```
struct timeval
{
    time_t tv_sec; /* Seconds. */
    suseconds_t tv_usec; /* Microseconds. */
};
```

- Xác định thời gian chờ của hàm select()
- Bị thay đổi giá trị khi hàm select() trả về kết quả
  ⇨ Cần khởi tạo mỗi khi gọi hàm select().

Cấu trúc chương trình sử dụng hàm **select():**



Select_Server(V1):

```
fd_set fdread;
int clients[64];
int numClients = 0;

struct timeval tv;
char buf[2048];

while (1) {
    // Khởi tạo và gắn các socket vào tập fdread
```

```
        FD_ZERO(&fdread);
        FD_SET(listener, &fdread);
        int maxdp = listener + 1;

        for (int i = 0; i < numClients; i++) {
            FD_SET(clients[i], &fdread);
            if (clients[i] + 1 > maxdp) maxdp = clients[i] + 1;
        }

        // Thiết lập thời gian chờ
        tv.tv_sec = 5;
        tv.tv_usec = 0;

        // Chờ đến khi sự kiện xảy ra
        int ret = select(maxdp, &fdread, NULL, NULL, &tv);
        if (ret < 0) {
            printf("select() failed.\n"); return 1;
        } else if (ret == 0) {
            printf("Timed out.\n"); continue;
        }

        // Kiểm tra nếu là sự kiện có yêu cầu kết nối
        if (FD_ISSET(listener, &fdread)) {
            int client = accept(listener, NULL, NULL);
            clients[numClients++] = client;
        }

        // Kiểm tra sự kiện nhận dữ liệu của các socket client
        for (int i = 0; i < numClients; i++){
            if (FD_ISSET(clients[i], &fdread)) {
                ret = recv(clients[i], buf, sizeof(buf), 0);
                if (ret <= 0) {
                    printf("Client %d disconnected\n", clients[i]);
                    removeClient(clients, &numClients, i);
                    i--;
                    continue;
                }
                buf[ret] = 0;
                printf("Data from client %d: %s\n", clients[i], buf);
            }
        }
    }
```

Select_Server(V2):

```
    // Khai báo tập fdread chứa các socket và tập fdtest để thăm dò sự
    kiện
    fd_set fdread, fdtest;
    struct timeval tv;
    char buf[2048];

    FD_ZERO(&fdread);
    FD_SET(listener, &fdread);

    while (1) {
        fdtest = fdread; // Giữ nguyên các socket trong tập fdread
        tv.tv_sec = 5; // Khởi tạo lại giá trị cấu trúc thời gian
        tv.tv_usec = 0;

        // Chờ đến khi sự kiện xảy ra hoặc hết giờ
```

```
    int ret = select(FD_SETSIZE, &fdtest, NULL, NULL, &tv);
    if (ret < 0) {
        printf("select() failed.\n");
        return 1;
    } else if (ret == 0) {
        printf("Timed out.\n"); continue;
    }

    for (int i = 0; i < FD_SETSIZE; i++){
        if (FD_ISSET(i, &fdtest)) {
            if (i == listener) { // Socket listener có sự kiện yêu
cầu kết nối
                int client = accept(listener, NULL, NULL);
                if (client < FD_SETSIZE) { // Chưa vượt quá số kết
nối tối đa
                    printf("New client connected %d\n", client);
                    FD_SET(client, &fdread); // Thêm socket vào tập
sự kiện
                } else { // Đã vượt quá số kết nối tối đa
                    close(client);
                }
            } else { // Socket client có sự kiện nhận dữ liệu
                ret = recv(i, buf, sizeof(buf), 0);
                if (ret <= 0) {
                    printf("Client %d disconnected\n", i);
                    FD_CLR(i, &fdread); // Xóa socket ra khỏi tập sự
kiện
                } else {
                    buf[ret] = 0;
                    printf("Received data from client %d: %s\n", i,
buf);
                }
            }
        }
    }
} // End while
```

Hàm poll():

- Hàm poll() thực hiện chức năng tương tự hàm select(): đợi trên một tập mô tả cho đến khi các thao tác vào ra sẵn sàng.
- Cú pháp:

```
#include <poll.h>
int poll (
      struct pollfd *fds, // Tập hợp các mô tả cần đợi sự kiện
      nfds_t nfds, // Số lượng các mô tả, không vượt quá
RLIMIT_NOFILE
      int timeout // Thời gian chờ theo ms. Nếu bằng -1 thì hàm chỉ
trả về kết quả khi có sự kiện xảy ra.
) => Hàm trả về số lượng cấu trúc có sự kiện xảy ra nếu thành công,
trả về -1 nếu bị lỗi. Trả về 0 nếu hết giờ.
```

- Cấu trúc pollfd

```
struct pollfd {
      int fd; // Mô tả (socket) cần thăm dò
      short int events; // Mặt nạ sự kiện cần kiểm tra
      short int revents; // Mặt nạ sự kiện đã xảy ra
}
```

- Chương trình cần thiết lập mặt nạ sự kiện trong trường events trước khi thăm dò và kiểm tra mặt nạ sự kiện trong trường revents sau khi thăm dò.
- Một số mặt nạ sự kiện hay dùng:
  - POLLIN/POLLRDNORM – Có kết nối / có dữ liệu để đọc
  - POLLOUT – Sẵn sàng ghi dữ liệu
  - POLLERR – Lỗi đọc / ghi dữ liệu

Poll Client:

```
// Khởi tạo client
struct pollfd fds[2];
fds[0].fd = STDIN_FILENO; // Mô tả của thiết bị nhập dữ liệu
fds[0].events = POLLIN;
fds[1].fd = client; // Mô tả của socket client
fds[1].events = POLLIN;

while (1) {
    int ret = poll(fds, 2, -1);
    if (fds[0].revents & POLLIN) { // Nếu có dữ liệu từ bàn phím
        fgets(buf, sizeof(buf), stdin);
        send(client, buf, strlen(buf), 0);
    }

    if (fds[1].revents & POLLIN) { // Nếu có dữ liệu từ socket
        ret = recv(client, buf, sizeof(buf), 0);
        if (ret <= 0) {
            break;
        }
        buf[ret] = 0;
        printf("Received: %s\n", buf);
    }
}
```

Poll server:

```
// Khởi tạo server
struct pollfd fds[64];
int nfds = 1;

fds[0].fd = listener;
fds[0].events = POLLIN;

char buf[2048];

while (1) {
    int ret = poll(fds, nfds, -1);
    if (fds[0].revents & POLLIN) { // Sự kiện có kết nối mới
        int client = accept(listener, NULL, NULL);
        printf("New client connected %d\n", client);
        fds[nfds].fd = client;
        fds[nfds].events = POLLIN;
        nfds++;
    }

    for (int i = 1; i < nfds; i++) {
        if (fds[i].revents & (POLLIN | POLLERR)) {// Sự kiện client
```

```
            ret = recv(fds[i].fd, buf, sizeof(buf), 0);
            if (ret <= 0) {
                // Xử lý khi kết nối client bị ngắt
            }

            // Xử lý dữ liệu từ client
            buf[ret] = 0;
            printf("Received data from client %d: %s\n", fds[i].fd,
buf);
        }
    }
} // end while
```