Socket Programming

TS. Trần Quang Vinh
BM. Kỹ thuật Thông tin
Viện Điện tử - Viễn thông
Đại học Bách Khoa Hà Nội
m706501@shibaura-it.ac.jp

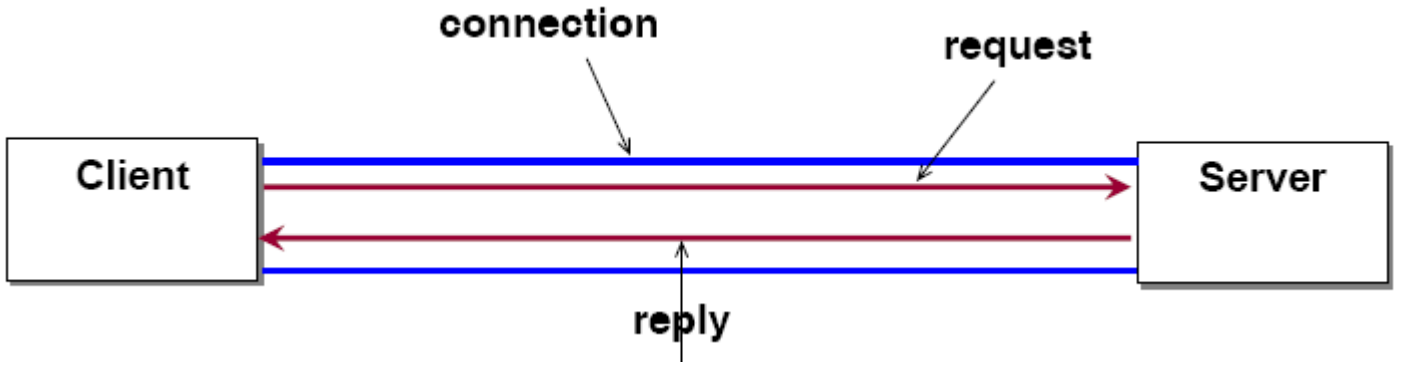
Socket là gì?

- Socket là một cổng logic mà một chương trình sử dụng để kết nối với một chương trình khác chạy trên một máy tính khác trên Internet
  - **Server thì sẽ cần một socket để lắng nghe các kết nối từ Client**
  - **Client thì phải cần có một socket để kết nối tới Sever**
- Chương trình mạng có thể sử dụng nhiều Socket cùng một lúc, nhờ đó nhiều ứng dụng có thể sử dụng Internet cùng một lúc.
- Socket Interface được định nghĩa trong UNIX BSD, dựa trên việc mở rộng tập các system calls (access files)
  - Hiện có nhiều platform khác nhau cung cấp API cho các ứng dụng mạng, có sự khác biệt nhỏ giữa các thư viện socket trên các platform khác nhau
- Việc lập trình ứng dụng sử dụng socket sẽ dựa vào 1 trong 2 giao thức TCP hoặc UDP của lớp giao vận

Mô hình hoạt động client/server dùng socket

- Hai ứng dụng client/server lúc còn độc lập nhau

- Hai ứng dụng client/server lúc giao tiếp nhau (dùng kết nối TCP và giao thức request/reply)



Cấu trúc địa chỉ Internet

- Định nghĩa dạng dữ liệu cấu trúc trong ngôn ngữ C. Cấu trúc này chỉ có 1 trường kiểu u_long chứa địa chỉ IP 32 bit

```
struct in_addr {
  in_addr_t    s_addr;              /* 32-bit IPv4 address */
                                    /* network byte ordered */
};
```

Cấu trúc địa chỉ Socket

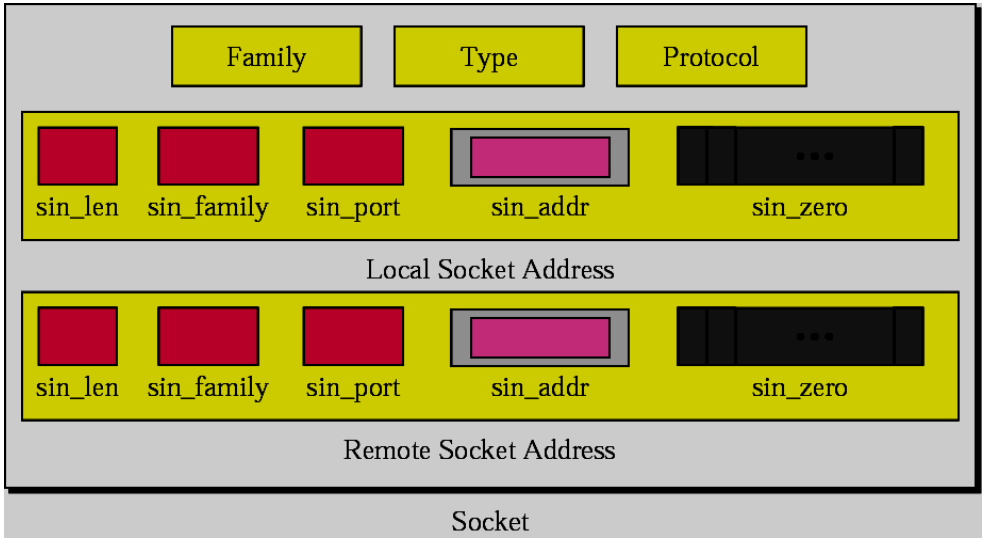
- Cấu trúc địa chỉ socket **sockaddr_in** lưu trữ địa chỉ IP, chỉ số port, và kiểu (family protocol)

```
struct sockaddr_in {
  uint8_t         sin_len;      /* length of structure (16) */
  sa_family_t     sin_family;   /* AF_INET */
  in_port_t       sin_port;     /* 16-bit TCP or UDP port number */
                                /* network byte ordered */
  struct in_addr  sin_addr;     /* 32-bit IPv4 address */
                                /* network byte ordered */
  char            sin_zero[8];  /* unused */
};
```

  - sin_len: lưu trữ chiều dài cấu trúc của sockaddr_in
  - sin_family: dạng protocol của socket
  - sin_port: chỉ số port
  - sin_addr: địa chỉ in Internet của socket
  - sin_zero[8]: không dùng, đặt giá trị = 0

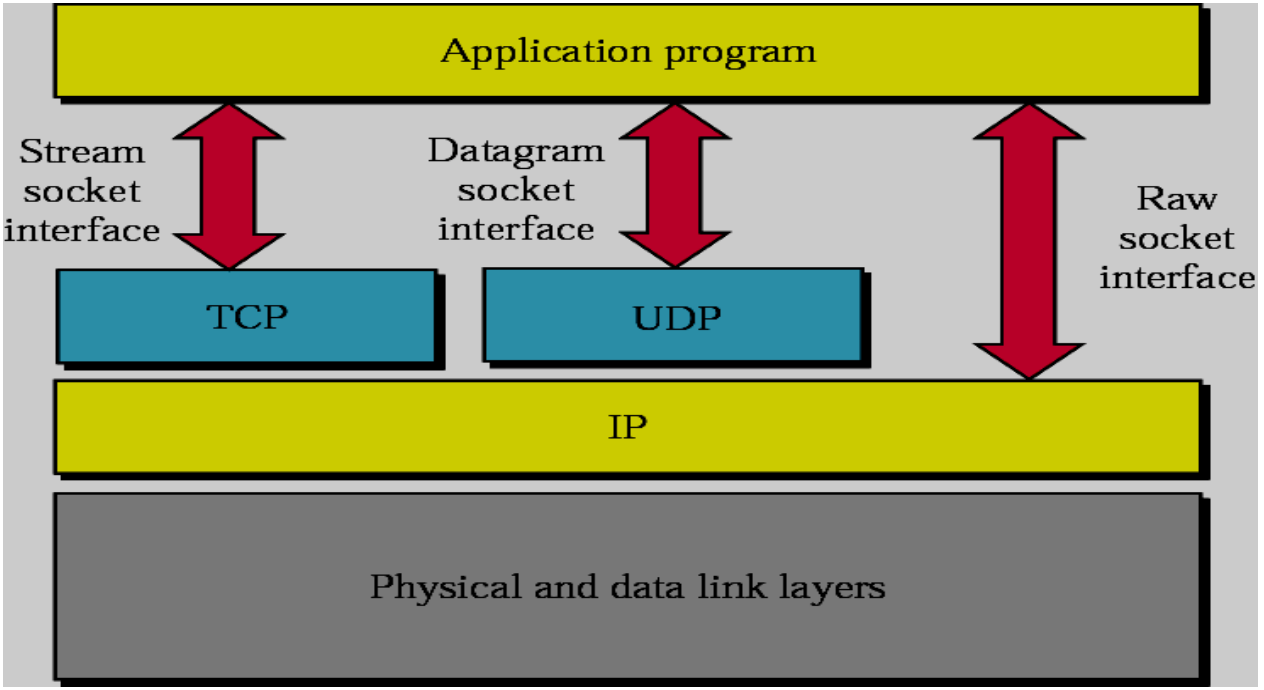
Cấu trúc socket

- Socket được định nghĩa trong hệ điều hành bằng một cấu trúc, được xem như điểm nối để hai procceses giao tiếp với nhau
- Cấu trúc socket gồm 5 field:
  - Family: xác định protocol group
  - Type: xác loại socket, stream, datagram hay raw socket.
  - Protocol: là field thường gán giá trị bằng 0
  - Local Socket Address và Remote Socket Address: là địa chỉ socket của process cục bộ và từ xa



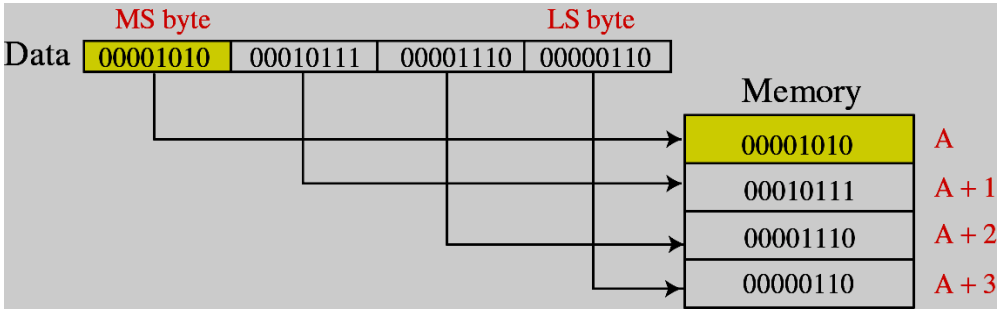
Các loại socket

- Các giao tiếp socket được định nghĩa trên môi trường TCP/IP
  - Stream Socket: dùng cho connection-oriented protocol (TCP)
  - Datagram Socket: dùng cho connectionless protocol (UDP)
  - Raw Socket: dùng cho một số protocol của một số ứng dụng đặc biệt, dùng các dịch vụ trực tiếp của lớp IP

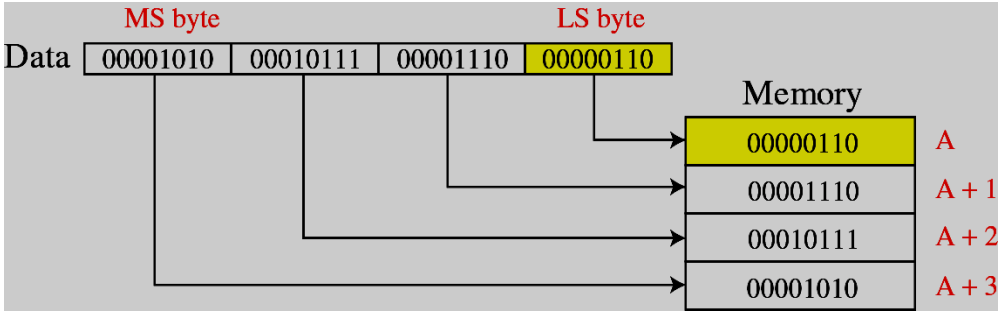


Byte Ordering

- Big-Endian Byte Order:
  - Byte có trọng số lớn lưu trước (Most Significant byte first)

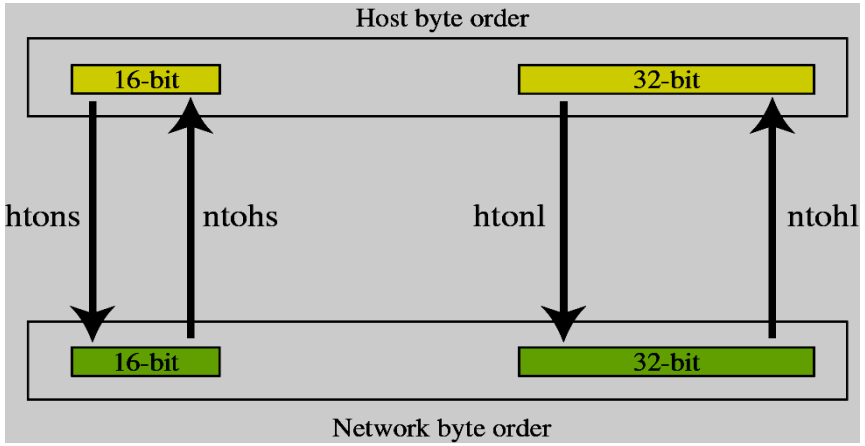


- Little-Endian Byte Order:
  - Byte có trọng số nhỏ lưu trước (Lest Significant byte first)



Byte Ordering

- Network Byte Order:
  - Thứ tự lưu trữ dùng cho giao tiếp mạng
  - Tuỳ cấu trúc của mỗi host, lưu trữ số theo một trong hai cách trên → không đồng nhất khi thực hiện giao tiếp trên network
  - Giao tiếp socket định nghĩa một số hàm để thực hiện các thao tác chuyển đổi:
    - **htons()** và **htonl()**: chuyển từ dạng lưu trữ của host sang network
    - **ntohs()** và **ntohl()**: chuyển từ dạng lưu trữ của network sang host

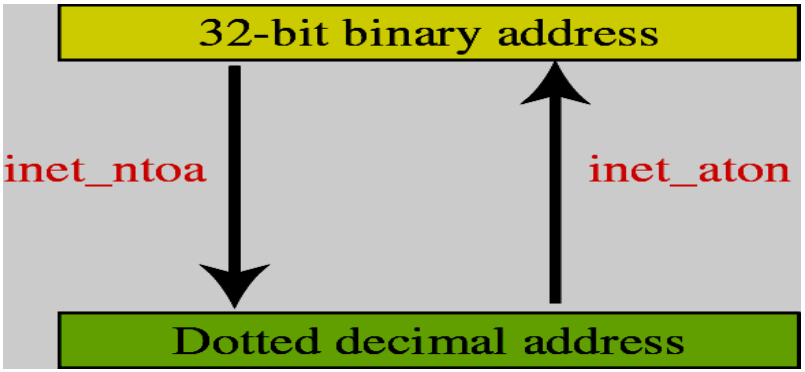



```
uint16_t  htons  ( uint16_t  host_short ) ;
uint16_t  ntohs  ( uint16_t  network_short ) ;
uint32_t  htonl  ( uint32_t  host_long ) ;
uint32_t  ntohl  ( uint32_t  network_long ) ;
```

inet_ntoa(), inet_aton()

```
char *inet_ntoa(struct in_addr in);
int inet_aton(const char *cp, struct in_addr *inp);
```

- Chuyển đổi địa chỉ IP từ dạng chuỗi "thập phân với dấu chấm" sang dạng cấu trúc **in_addr** và ngược lại
  - **inet_aton()** returns non-zero if the address is a valid one, and it returns zero if the address is invalid
  - **inet_ntoa()** returns the dots-and-numbers string in a static buffer that is overwritten with each call to the function
    - The "n" in "ntoa" stands for network, and the "a" stands for ASCII, so it's "Network To ASCII"



Các hàm cơ bản dùng cho lập trình socket

- socket()
- bind()
- connect()
- listen()
- accept()
- read() / write()
- send() / recv()
- sendto() / recvfrom()
- close() / shutdown()

Hàm socket()

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

- Hàm **socket()** tạo một socket, kết quả trả về là một số nguyên nhận dạng (socket descriptor), nếu có lỗi giá trị trả về là -1
- Các tham số:
  - **domain**: họ socket
  - **type**: kiểu socket
  - **protocol**: giao thức, thường đặt bằng 0

Hàm socket()

Figure 4.2. Protocol *family* constants for `socket` function

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols (Chapter 15)
AF_ROUTE	Routing sockets (Chapter 18)
AF_KEY	Key socket (Chapter 19)

Figure 4.3. *type* of socket for `socket` function.

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_SEQPACKET	sequenced packet socket
SOCK_RAW	raw socket

Figure 4.4. *protocol* of sockets for `AF_INET` or `AF_INET6`.

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

Hàm socket()

- Ví dụ

// Khởi tạo socket mới, nếu thất bại báo sai

```
#include <sys/types.h>
#include <sys/socket.h>
int main(void)
{
     int sockfd;
     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
     {
          perror("socket");
          exit(1);
     }
     printf("Sockfd : %d \n", sockfd);
}
```

Hàm bind()

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

- Chức năng:
  - Đăng ký socket đã khởi tạo với địa chỉ socket local
  - Trả về 0 nếu thành công, -1 nếu thất bại
- Các tham số:
  - **sockfd**: mô tả socket trả về bởi hàm **socket()**
  - **my_addr**: con trỏ chỉ đến struct sockaddr của địa chỉ socket bao gồm cổng và IP address
  - **addrlen**: chiều dài (byte) của địa chỉ socket

Hàm connect()

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

- Chức năng
  - Dùng cho chương trình client thiết lập kết nối đến server
  - Trả về 0 nếu thành công, -1 nếu thất bại
- Các tham số:
  - sockfd: mô tả socket được trả về bởi hàm socket()
  - serv_addr: con trỏ địa chỉ socket của server
  - addrlen: chiều dài của địa chỉ socket server

Hàm listen()

```
int listen(int sockfd, int backlog);
```

- Chức năng:
  - Hàm listen() để kết nối đến server, khai báo độ dài hàng chờ
  - Hàm này dùng cho chương trình server connection-oriented để đặt socket ở trạng thái chờ, lắng nghe kết nối từ phía client
  - Trả về 0 nếu thành công, -1 nếu thất bại
- Các tham số:
  - sockfd: mô tả socket đã tạo bởi hàm socket()
  - backlog: số request có thể queued (giới hạn bởi 20, thường từ 5-10)

Hàm accept()

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- Chức năng
  - Chấp nhận kết nối từ client, tạo socket mới
  - Giá trị là một socket descriptor của socket mới
  - Returns: non-negative descriptor if OK, -1 on error
- Các tham số:
  - sockfd: mô tả socket đã tạo bởi hàm socket()
  - addr: con trỏ địa chỉ socket của client kết nối đến
  - addrlen: chiều dài của addr

Các hàm trao đổi dữ liệu

- Gửi/Nhận dữ liệu trên giao thức TCP (stream sockets or connected datagram sockets)
  - **send()** / **recv()**
  - **write()** / **read()**
- Gửi/Nhận dữ liệu trên giao thức UDP (regular unconnected datagram sockets)
  - **sendto()** / **recvfrom()**

Hàm send()

```
int send(int sockfd, const void *msg, int len, int flags);
```

- Chức năng
  - Gửi dữ liệu tới một socket
  - Trả về số bytes được gửi thành công, trả về -1 nếu thất bại
    - Số bytes được gửi thành công có thể nhỏ hơn số bytes thực sự muốn gửi
- Các tham số:
  - sockfd: mô tả socket muốn gửi dữ liệu đến, đây có thể là giá trị trả về bởi hàm socket() hoặc accept()
  - msg: con trỏ đến dữ liệu muốn gửi
  - len: chiều dài của dữ liệu (in byte)
  - flags: tập các cờ điều khiển hoạt động của hàm này, thường set giá trị 0

Hàm recv()

```
int recv(int sockfd, void *buf, int len, int flags);
```

- Chức năng
  - Nhận dữ liệu từ một socket
  - Trả về số bytes nhận được, trả về -1 nếu thất bại
    - Số bytes nhận được có thể nhỏ hơn số bytes yêu cầu bởi tham số len
- Các tham số:
  - sockfd: mô tả socket muốn nhận dữ liệu
  - buf: con trỏ đến bộ đệm muốn lưu dữ liệu
  - len: chiều dài tối đa của bộ đệm (in byte)
  - flags: tập các cờ điều khiển hoạt động của hàm này, thường set giá trị 0

Hàm read()

```
int read(int sockfd, const void *buf, int len);
```

- Chức năng
  - Đọc dữ liệu từ connection vào bộ nhớ
  - Trả về số bytes đọc được nếu thành công, trả về 0 nếu không có dữ liệu, trả về -1 nếu thất bại
- Các tham số:
  - **sockfd**: mô tả socket đã tạo bởi hàm **socket()**
  - **buf**: con trỏ đến bộ đệm để lưu thông tin đọc được
  - **len**: chiều dài của bộ đệm

Hàm write()

```
int write(int sockfd, const void *buf, int len);
```

- Chức năng
  - Ghi dữ liệu từ bộ nhớ lên connection.
  - Trả về số bytes ghi được nếu thành công, trả về -1 nếu thất bại
- Các tham số:
  - **sockfd**: mô tả socket đã tạo bởi hàm **socket()**
  - **buf**: con trỏ đến bộ đệm để lưu thông tin đọc được
  - **len**: chiều dài của bộ đệm

Hàm sendto()

```
int sendto (int sockfd,
            const void *buf,
            int len,
            int flags,
            const struct sockaddr_in *toaddr,
            int toaddrlen);
```

- Chức năng:
  - Gửi dữ liệu đến một địa chỉ socket từ xa
  - Trả về số bytes gửi được nếu thành công, trả về -1 nếu thất bại
- Các tham số:
  - **sockfd**, **buf**, **len**: giống các hàm đã giới thiệu
  - **flags**: thường đặt bằng 0
  - **toaddr**, **toaddrlen**: địa chỉ socket đến và chiều dài của nó

Hàm recvfrom()

```
int recvfrom ( int sockfd,
               const void *buf,
               int len,
               int flags,
               const struct sockaddr_in *fromaddr,
               int fromaddrlen);
```

- Chức năng
  - Nhận dữ liệu từ một địa chỉ socket từ xa
  - Trả về số bytes gửi được nếu thành công, trả về -1 nếu thất bại
- Các tham số:
  - **sockfd**, **buf**, **len**: giống các hàm đã giới thiệu
  - **flags**: thường đặt bằng 0
  - **fromaddr**, **fromaddrlen**: địa chỉ socket gửi đến và chiều dài của nó

Hàm fork() – Khởi tạo process

```
#include <unistd.h>

pid_t  fork(void);
```

- Chức năng
  - Khởi tạo các process, sử dụng trong multi-threading
  - Không có tham số
  - Nếu khởi tạo thành công, trả về hai giá trị: (-1 nếu lỗi)
    - Trả về process ID của child's process cho parent's process
    - Trả về 0 cho child's process
  - type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer

Hàm close() / shutdown()

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

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

- Chức năng
  - Hàm **close()** đóng một socket cả phía client và server, đồng thời giải phóng socket
  - Hàm **shutdown()** đóng một phía client hoặc server, không giải phóng socket, vẫn phải gọi hàm **close()** khi muốn thực sự ngắt kết nối
    - Tham số **how**

0	Further receives are disallowed
1	Further sends are disallowed
2	Further sends and receives are disallowed (like **close()**)

  - Trả về 0 nếu thành công và -1 nếu thất bại
  - Trên windows platfrom gọi hàm **closesocket()**

Hàm thiết lập bộ nhớ - memset()

void* memset(void* buffer, int ch, size_t count);

- Chức năng
  - Khởi tạo vùng bộ nhớ chỉ ra bởi con trỏ **buffer**, điền đầy **count** bytes đầu tiên của vùng bộ nhớ này bởi **ch**
  - Trả về con trỏ chỉ tới vùng bộ nhớ **buffer**

```
#include <stdio.h>
#include <string.h>

#define LENGTH 10
int main(void)
{
   char x='X';
   char buffer[LENGTH];
   int cnt=0;

   memset(buffer,x,LENGTH);
   for(;cnt < LENGTH; cnt++) {
         printf("\nbuffer[%d] = %c",cnt,buffer[cnt]);
   }
```

Hàm so sánh memcmp()

```
int memcmp(const void *buffer1,
           const void *buffer2, size_t count);
```

- Chức năng
  - So sánh 'count' kí tự đầu tiên của 2 chuỗi buffer1 và buffer2
  - Giá trị trả về:
    - < 0: buffer1 < buffer2
    - = 0: buffer1=buffer2
    - > 0: buffer1 > buffer2

Hàm copy bộ nhớ memcpy()

void *memcpy(void *to, const void *from, size_t count);

- Chức năng
  - Copy **count** kí tự từ **from** sang **to**
  - Giá trị trả về là **to**
    - hàm này sẽ copy nguyên chính xác số bytes cần copy và không copy bytes cuối cùng của string '\0'. Vì vậy cần phải copy n+1 nếu muốn copy n byte

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   const char from[]="Xcross87 is a chick of programming";
   char* to;

   memcpy(to,from,strlen(from)+1);

            printf(" Gia tri cua \'to\' sau khi copy la: %s ",to);

   return 0;
}
```

getaddrinfo(), freeaddrinfo(), gai_strerror()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node,     // e.g. "www.example.com" or IP
                const char *service,  // e.g. "http" or port number
                const struct addrinfo *hints,
                struct addrinfo **res);
```

```
void freeaddrinfo(struct addrinfo *ai);

const char *gai_strerror(int ecode);
```

- Chức năng
  - **getaddrinfo()** lấy thông tin về host name, service, và khởi tạo sockaddr bằng kết quả trả về
  - Trả về 0 nếu thành công, khác 0 nếu có lỗi, bắt lỗi bằng hàm **gai_strerror()**
  - **freeaddrinfo()** giải phóng bộ nhớ

Giải thuật cho TCP client

- Xác định địa chỉ server
- Tạo socket
- Kết nối đến server
- Gởi/nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế
- Đóng kết nối

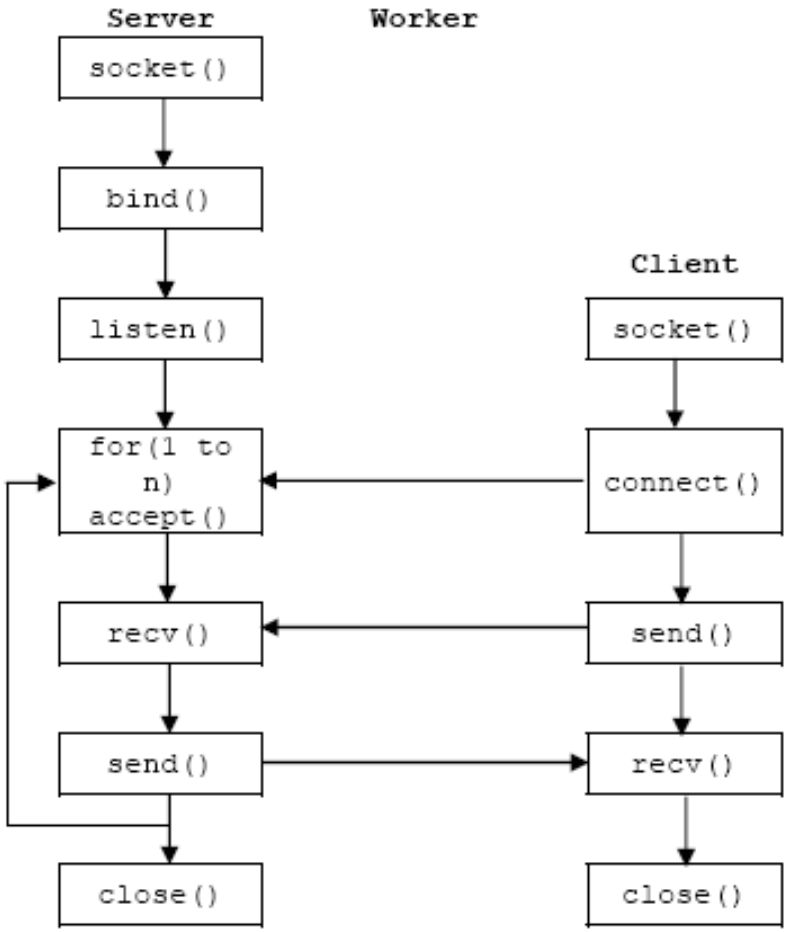
Giải thuật cho UDP client

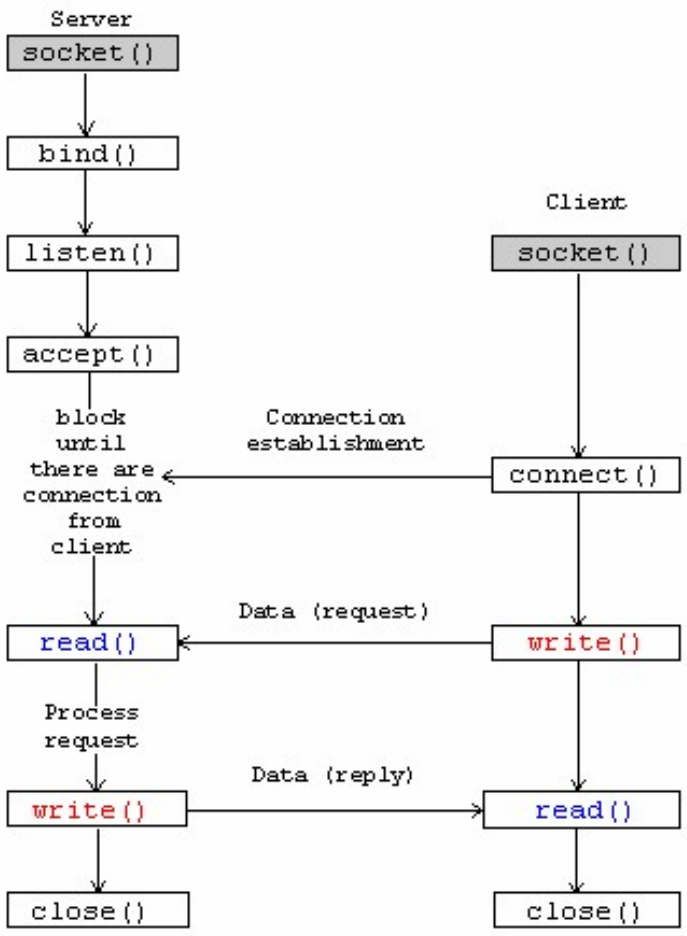
- Xác định địa chỉ server
- Tạo socket
- Đăng ký socket với hệ thống
- Gởi/nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế đến server theo địa chỉ đã xác định
- Đóng kết nối

Giải thuật cho iterative server

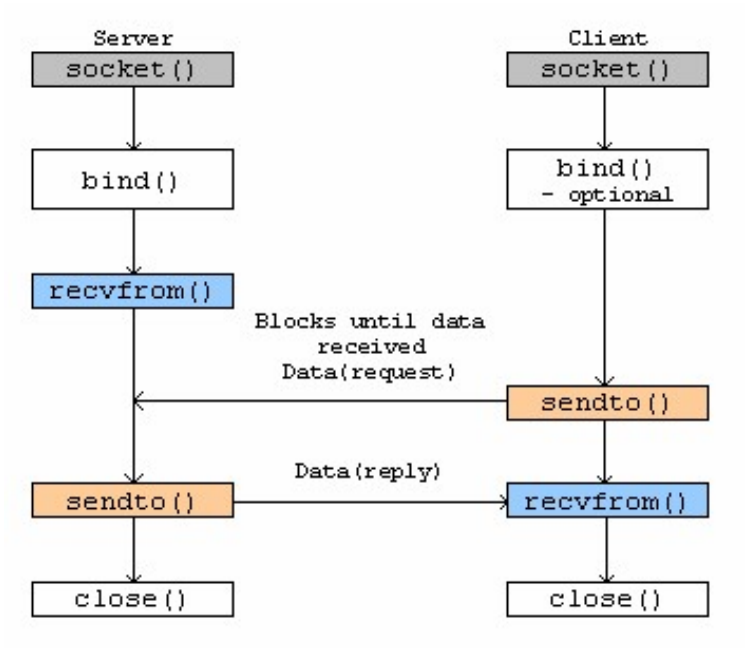
- Connection-oriented iterative server:
  - Tạo socket
  - Đăng ký địa chỉ socket với hệ thống
  - Đặt socket ở trạng thái lắng nghe, chờ và sẵn sàng cho việc kết nối từ client
  - Chấp nhận kết nối từ client, gửi/nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế
  - Đóng kết nối sau khi hoàn thành, trở lại trạng thái lắng nghe và chờ kết nối mới
- Connectionless iterative server:
  - Tạo socket
  - Đăng ký với hệ thống
  - Lặp công việc đọc dữ liệu từ client gửi đến, xử lý và gửi trả kết quả cho client theo đúng giao thức lớp ứng dụng đã thiết kế

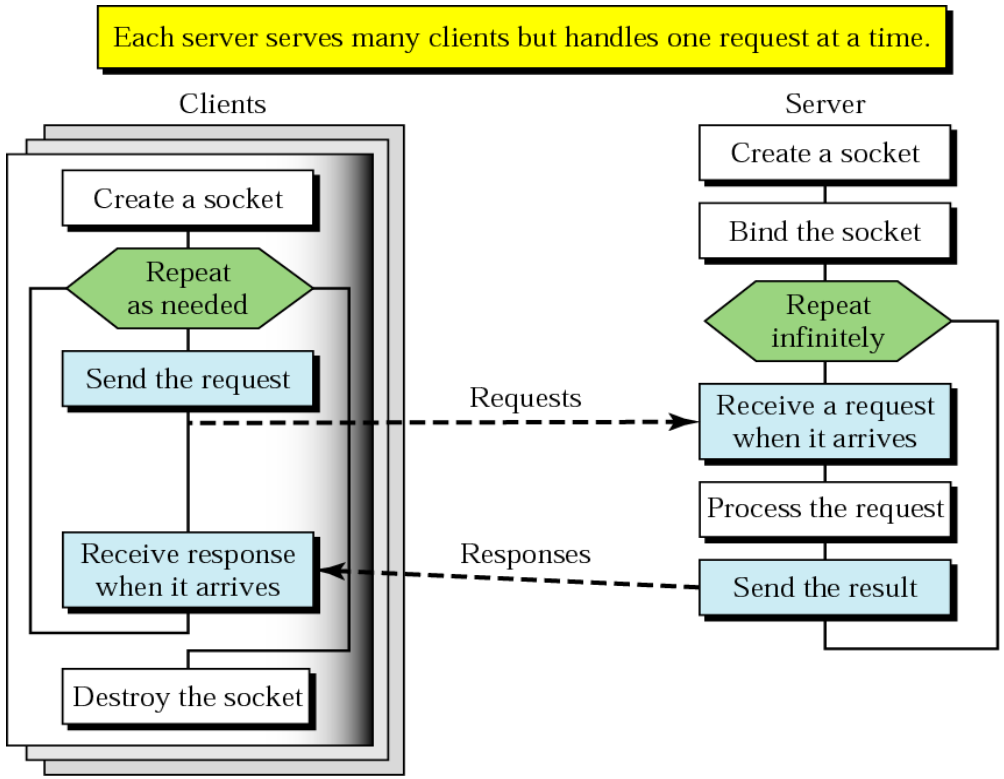
Connection-oriented iterative server



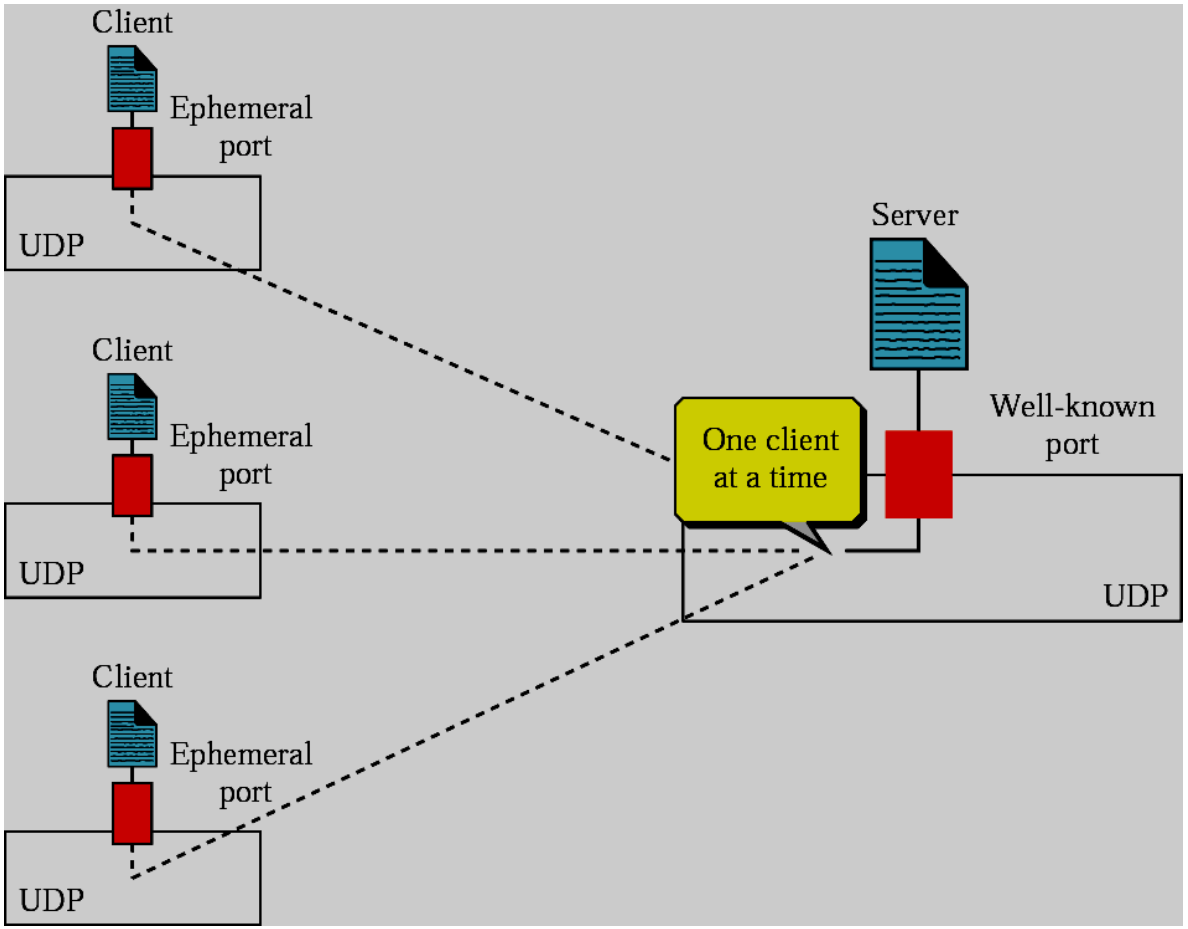


Connectionless iterative server





Connectionless iterative server

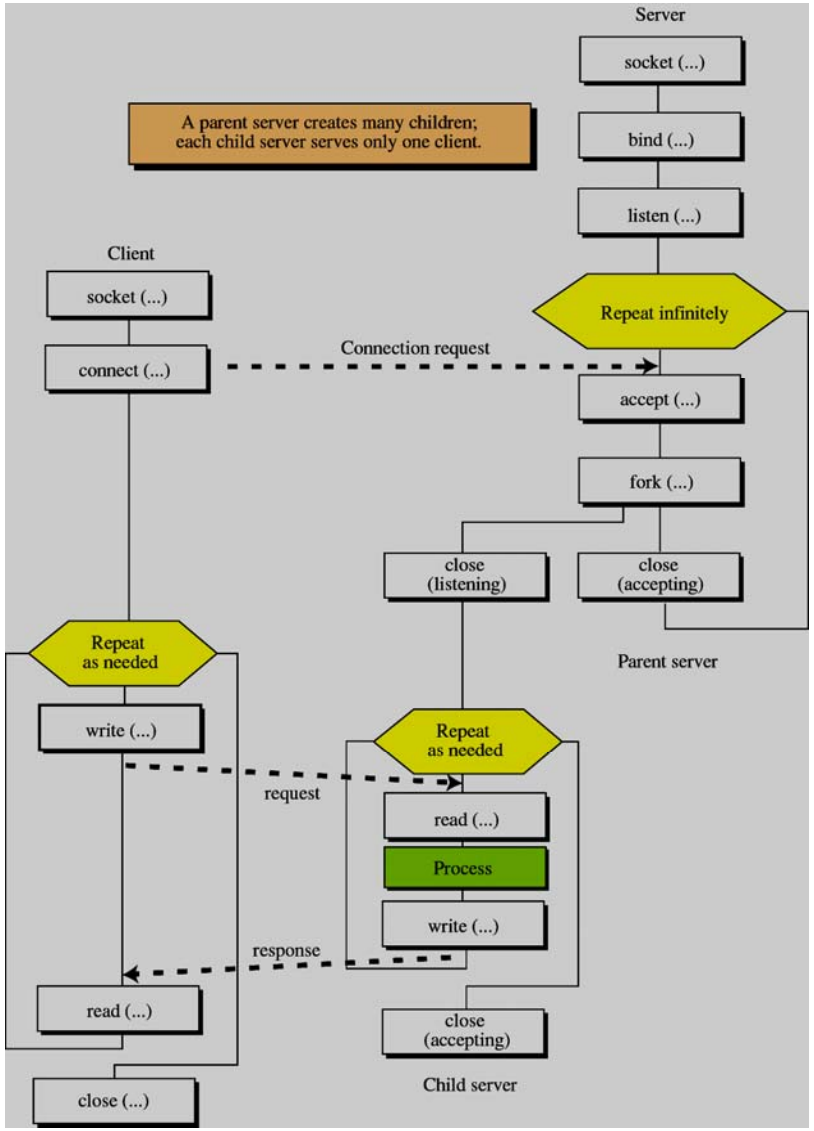


Concurrent Server

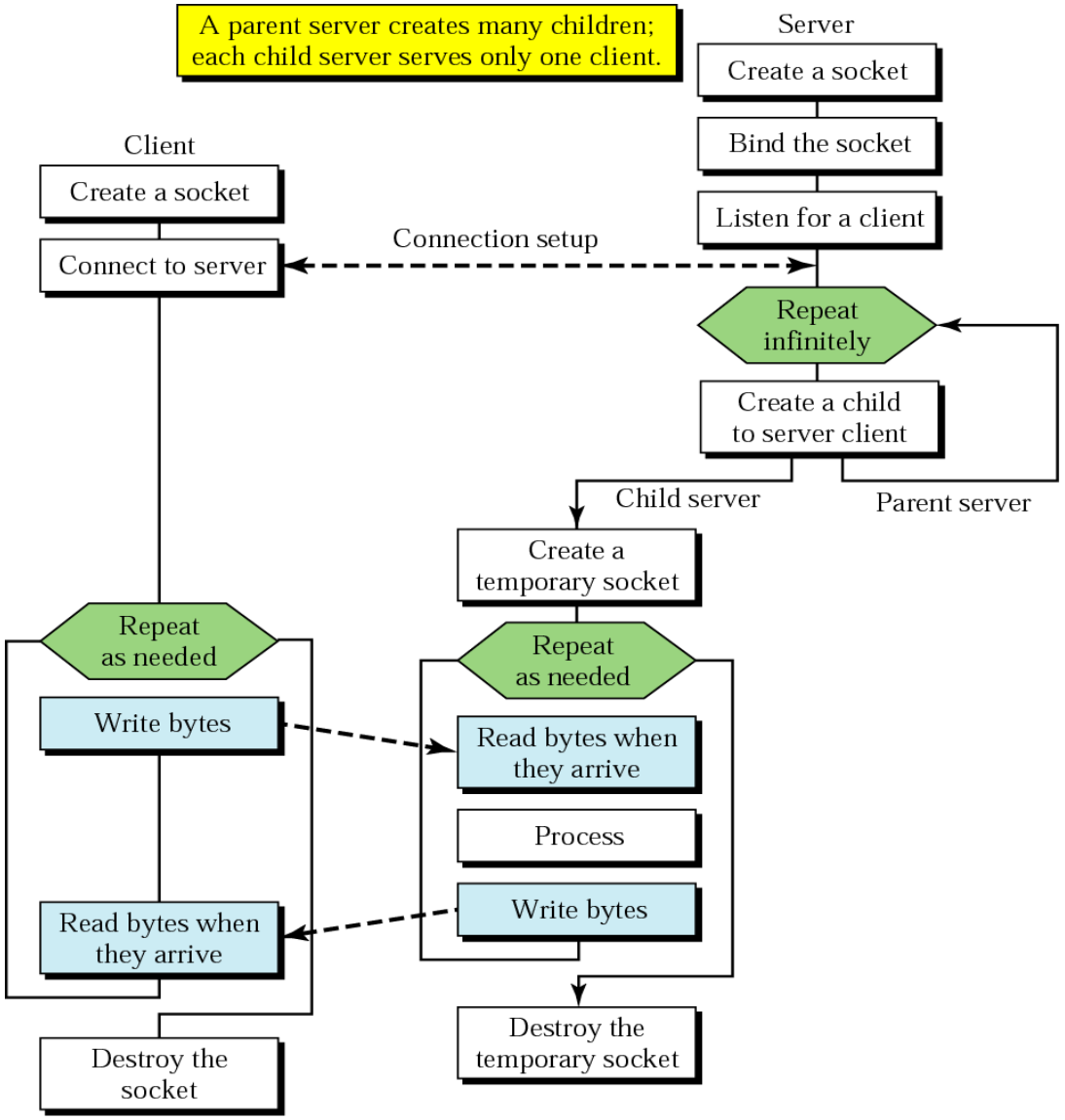
- Các yêu cầu cho concurrent Server:
  - Tại một thời điểm có thể xử lý nhiều yêu cầu từ client
  - Chương trình concurrent server có thể chạy trên máy chỉ có 1 CPU
  - Hệ thống phải hỗ trợ multi-tasking

Concurrent Server

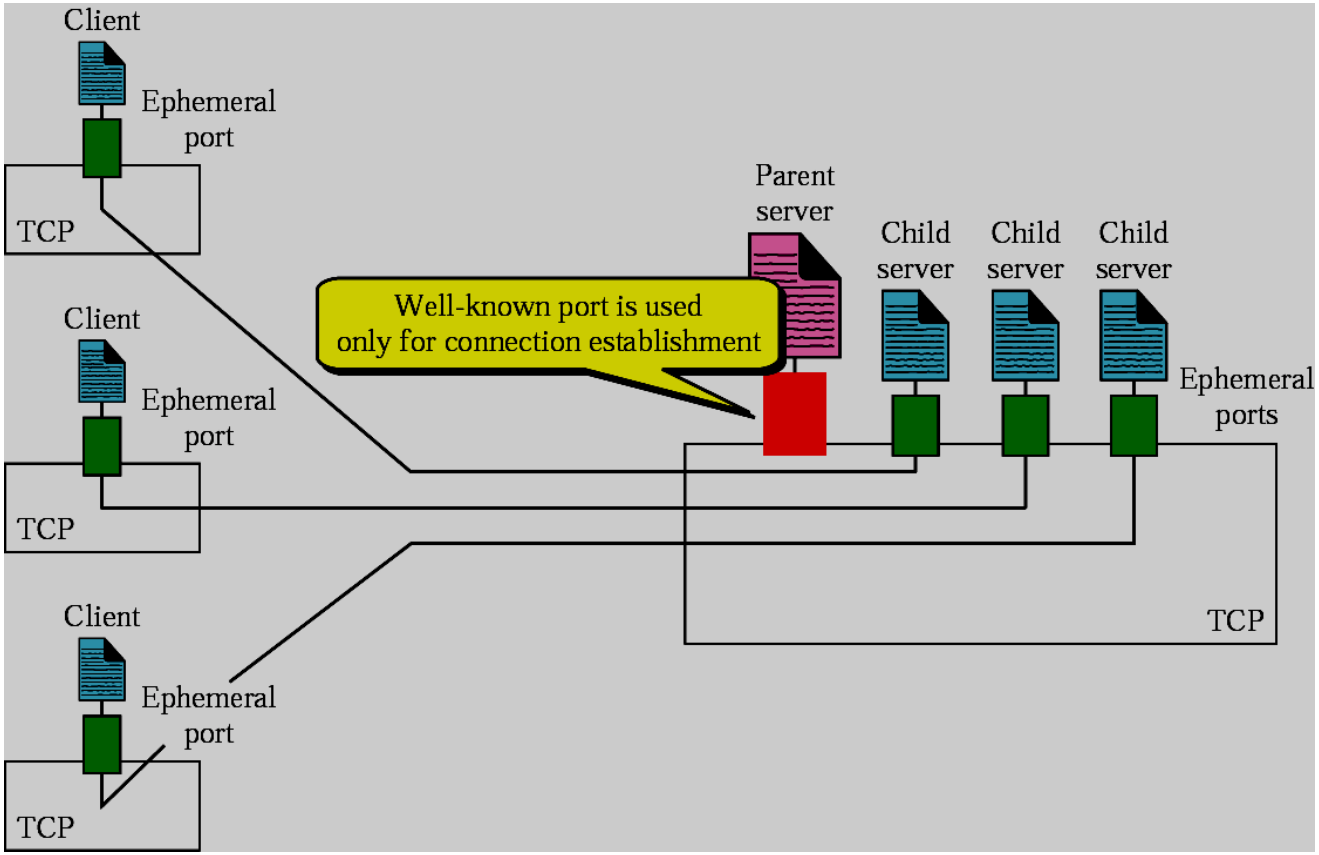
- Giải thuật cho chương trình connection-oriented concurrent server:
  - Tạo socket, đăng ký với hệ thống
  - Đặt socket ở chế độ chờ, lắng nghe kết nối
  - Khi có request từ client, chấp nhận kết nối, tạo một process con để xử lý. Quay lại trạng thái chờ, lắng nghe kết nối mới
  - Công việc của process mới gồm:
    - Nhận thông tin kết nối của client
    - Giao tiếp với client theo giao thức lớp ứng dụng đã thiết kế
    - Đóng kết nối và kết thúc process con



Connection-oriented concurrent server



Connection-oriented concurrent server



Concurrent Server

- Giải thuật cho chương trình concurrent, connectionless server:
  - Tạo socket, đăng ký với hệ thống
  - Lặp việc nhận dữ liệu từ client, đối với một dữ liệu nhận, tạo mới một process để xử lý. Tiếp tục nhận dữ liệu mới từ client
  - Công việc của process mới:
    - Nhận thông tin của process cha chuyển đến, lấy thông tin socket
    - Xử lý và gửi thông tin về cho client theo giao thức lớp ứng dụng đã thiết kế
    - Kết thúc

Multi-protocol Server (TCP,UDP)

- Dùng một chương trình để mở một master socket cho cả TCP và UDP
- Dùng hàm hệ thống (select) để chọn lựa TCP socket hay UDP socket sẵn sàng
- Tùy vào protocol (TCP, UDP) để xử lý gửi nhận thông điệp theo đúng giao thức của lớp ứng dụng
- Tham khảo thêm RFC 1060

Multi-service Server

- Tạo một điểm giao tiếp chung
- Với mỗi request, xem loại dịch vụ cần xử lý
- Với mỗi loại dịch vụ, xử lý riêng biệt
- Có thể kết hợp Multi-service và Multi- protocol để thiết kế cho chương trình server

SOME SUMMARY

- Let see, what we have covered till now.

Socket Library Functions

- System calls:

  1. Startup / close.
  2. Data transfer.
  3. Options control.
  4. Other.

- Network configuration lookup:

  1. Host address.
  2. Ports for services.
  3. Other.

- Utility functions:

  1. Data conversion.
  2. Address manipulation.
  3. Error handling.

SOME SUMMARY

Primary Socket Calls

`socket()`	Create a new socket and return its descriptor.
`bind()`	Associate a socket with a port and address.
`listen()`	Establish queue for connection requests.
`accept()`	Accept a connection request.
`connect()`	Initiate a connection to a remote host.
`recv()`	Receive data from a socket descriptor.
`send()`	Send data to a socket descriptor.
`read()`	Reads from files, devices, sockets etc.
`write()`	Writes to files, devices, sockets etc.
`close()`	"One-way" close of a socket descriptor.
`shutdown()`	Allows you to cut off communication in a certain direction, or both ways just like close() does.

SOME SUMMARY

Network Database Administration functions

- gethostbyname() - given a hostname, returns a structure which specifies its DNS name(s) and IP address (es).
- getservbyname() - given service name and protocol, returns a structure which specifies its name(s) and its port address.
- gethostname() - returns hostname of local host.
- getservbyname(), getservbyport(), getservent().
- getprotobyname(), getprotobynumber(), getprotobyent(), getnetbyname(), getnetbyaddr(), getnetent().

Socket Utility Functions

`ntohs()/ntohl()`	Convert short/long from network byte order (big endian) to host byte order.
`htons()/htonl()`	Convert short/long from host byte order to network byte order.
`inet_ntoa()/inet_addr()`	Convert 32-bit IP address (network byte order to/from a dotted decimal string).
`perror()`	Print error message (based on "errno") to stderr.
`herror()`	Print error message for gethostbyname() to stderr (used with DNS).

SOME SUMMARY

Primary Header Files

- Include file sequence may affect processing (order is important!). Other header files that define macro, data type, structure and functions are given in the summary Table at the end of this Tutorial.

`<sys/types.h>`	Prerequisite typedefs.
`<errno.h>`	Names for "errno" values (error numbers).
`<sys/socket.h>`	struct sockaddr; system prototypes and constants.
`<netdb.h.h>`	Network info lookup prototypes and structures.
`<netinet/in.h>`	struct sockaddr_in; byte ordering macros.
`<arpa/inet.h>`	Utility function prototypes.