Lập trình socket nâng cao: Tùy biến socket

TS. Nguyễn Hoài Sơn
Bộ môn mạng và Truyền thông máy tính,
Đại học Công nghệ, Đại học QG Hà Nội

Nội dung

- Tùy biến socket
- Xử lý tín hiệu POSIX
- Xuất nhập dữ liệu với ngắt tín hiệu
- Các hàm xuất nhập dữ liệu nâng cao

It‘s important to know about some of these topics, although it might not be apparent how and when to use them.

Chúng ta có thể thiết lập các tùy chọn nào?

- Tùy chọn socket
  - Liên quan đến cách thức hoạt động của socket và các giao thức mạng (TCP/IP)
    - Tùy chọn chung, tùy chọn IPv4, IPv6, TCP
- Tùy chọn file
  - Liên quan đến cách thức xử lý xuất nhập dữ liệu
    - Xuất nhập dữ liệu không đồng bộ, xuất nhập dữ liệu dựa tín hiệu, thiết lập sở hữu socket

Tùy biến socket

- Cách thức hoạt động của socket được quy định bởi nhiều thuộc tính
  - Làm trễ gói tin, kích thước bộ đệm, quy trình đóng socket,…
- Thiết lập tùy biến socket để thông báo cho OS/Protocol Stack cách thức hoạt động của socket mà chúng ta muốn
- Có hai loại
  - Tùy biến chung (áp dụng cho tất cả (nhiều) các loại socket)
  - Tùy biến áp dụng cho một giao thức nhất định

Kiểu tùy biến

- Tùy biến nhị phân:
  - biểu thị bằng cờ nhị phân
    - 1: cho phép, 0: không cho phép

- Tùy biến theo giá trị
  - có giá trị kiểu **`int`**, **`timeval`**, **`in_addr`**, **`sockaddr`**, etc.

getsockopt(): Đọc giá trị tùy biến

#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname,
void *opval, socklen_t *optlen);

- ***sockfd:*** mô tả socket
- ***level***: loại tùy biến: tùy biến chung (SOL_SOCKET) hay tùy biến với một loại giao thức nhất định (IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP)
- ***optname***: số nguyên dương đặc tả tùy biến
- ***optval:*** con trỏ đến biến lưu giá trị của tùy biến
- ***optlen:*** con trỏ đến biến lưu kích thước của tùy biến

Chương trình: *sockopt/checkopts.c*

setsockopt(): Thiết lập giá trị tùy biến

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

- ***optval:*** con trỏ đến biến lưu giá trị của tùy biến được gán
- ***optlen:*** Kích thước của tùy biến.

Tùy biến chung

- Các tùy biến áp dụng với nhiều loại giao thức khác nhau
  - Một số tùy biến chung chỉ áp dụng với một số kiểu socket nhất định (SOCK_DGRAM, SOCK_STREAM)
- Một số tùy biến chung

SO_BROADCAST

SO_DONTROUTE

SO_ERROR

SO_KEEPALIVE

SO_LINGER

SO_RCVBUF,SO_SNDBUF

SO_REUSEADDR

Tùy biến SO_BROADCAST

- Tùy biến nhị phân: cho phép/không cho phép gửi các gói tin phát tràn
  - Đ/k: Tầng liên kết dữ liệu phải hỗ trợ gửi phát tràn
  - Không áp dụng với SOCK_STREAM sockets.
- Ngăn ứng dụng gửi gói tin phát tràn một cách vô ý thức
  - OS sẽ kiểm tra cờ nhị phân này trước khi gửi một gói tin có địa chỉ broadcast
- Ví dụ: bcast/dgclibcast1.c

Tùy biến SO_DONTROUTE

- Tùy biến nhị phân: cho phép/không cho phép bỏ qua cơ chế định tuyến thông thường
  - E.g. Gói tin được gửi thẳng đến card mạng thích hợp dựa trên địa chỉ đích
- Được sử dụng bởi các chương trình định tuyến (e.g. routed and gated).

Tùy biến SO_ERROR

- Tùy biến giá trị số nguyên: chỉ ra lỗi xuất hiện tại socket
  - Trong modun giao thức, biến *so_error* ghi lại lỗi xảy ra tại socket
  - Tiến trình được báo lỗi socket theo hai cách: dựa trên giá trị trả về của hàm *select* hoặc phát sinh tín hiệu SIGIO
- Tiến trình sử dụng tùy biến SO_ERROR để nhận giá trị của biến *so_error*
  - Tùy biến này chỉ được đọc

Tùy biến SO_KEEPALIVE

- Tùy biến nhị phân
  - Nếu cho phép tùy biến này, TCP sockets sẽ gửi gói tin "thăm dò" nếu không có dữ liệu trao đổi trong một khoảng thời gian "dài"
    - Cho phép tiến trình xác định máy kết nối có bị lỗi hay không
  - Thường dùng với máy chủ
    - Ví dụ: telnet
    - Phân biệt trạng thái rỗi và trạng thái lỗi

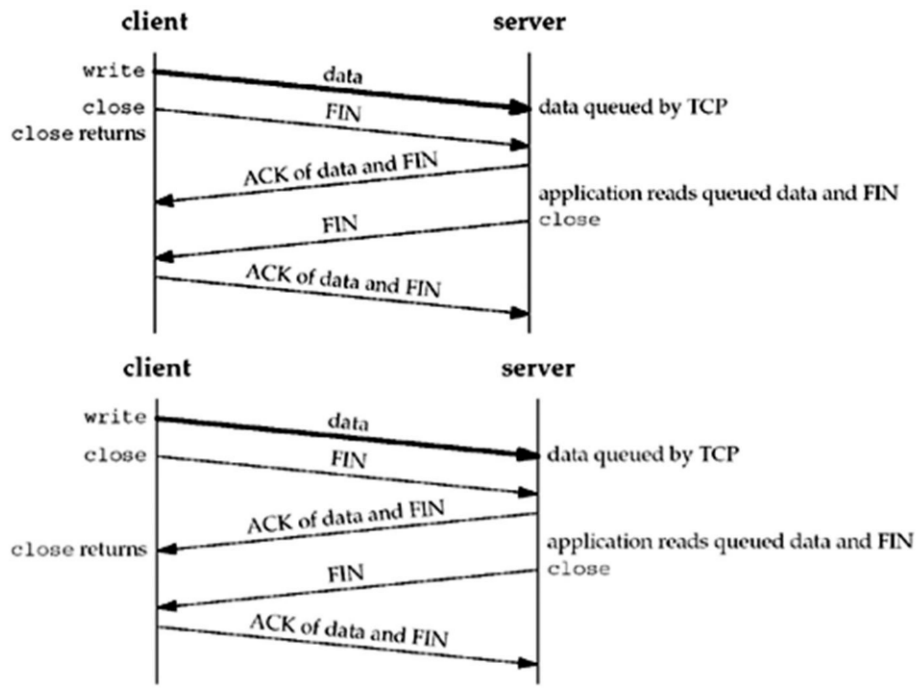
SO_LINGER

- Tùy biến giá trị với kiểu:

```
struct linger {
  int l_onoff;      /* 0 = off */
  int l_linger;  /* time in seconds */
};
```

  - Điều khiển cách thức đóng socket bởi lệnh close()
    - **l_onoff = 1, l_linger = 0: Loại bỏ các gói tin ở bộ đệm gửi, gửi gói tin RST và trả về giá trị**
    - **l_onoff = 1, l_linger > 0: Chờ FIN của tất cả các gói tin còn trong bộ đệm hoặc thời gian chờ (l_linger) hết hạn trước khi trả về giá trị**
  - Chỉ dùng với TCP sockets

So sánh close() mặc định vs. có sử dụng tùy biến SO_LINGER



SO_RCVBUF
SO_SNDBUF

- Tùy biến giá trị nguyên
  - Thay đổi kích thước bộ đệm gửi và nhận
    - Giá trị mặc định phụ thuộc vào các OS khác nhau
  - Dùng với STREAM và DGRAM sockets
- Với TCP socket, tùy biến này phải được thiết lập trước khi kết nối
  - tùy biến này ảnh hưởng đến kích thước window trong điều khiển luồng
- Kích thước bộ đệm phụ thuộc vào:
  - Kích thước MSS
    - Kích thước bộ đệm ít nhất là 4 MSS
    - Một số nguyên lần MSS
  - Băng thông cuối-cuối

SO_REUSEADDR

- Tùy biến nhị phân
  - cho phép gán địa chỉ socket đã được sử dụng bởi một kết nối khác
- Được sử dụng trong các trường hợp
  - Tiến trình của máy chủ sử dụng lại cổng đang được sử dụng bởi một tiến trình con của nó
  - Thiết lập nhiều máy chủ trên cùng một cổng nhưng với các giao diện mạng khác nhau (hay với các địa chỉ IP khác nhau trên cùng một giao diện mạng)
  - Gán địa chỉ socket với địa chỉ IP và cổng trùng nhau cho các socket khác nhau
    - Chỉ hỗ trợ với UDP sockets khi sử dụng unicast và multicast socket trên cùng một cổng

Các tùy biến IP (IPv4)

- IP_HDRINCL: sử dụng với IP socket thô (raw IP socket) khi muốn chương trình tự tạo tiêu đề IP
- IP_TOS: Thiết lập trường "Type-of-service" trong tiêu đề IP
  - Ví dụ: IPTOS_LOWDELAY, IPTOS_THROUGHPUT
- IP_TTL: Thiết lập trường "Time-to-live" trong tiêu đề IP
- IP_RECVDSTADDR/IP_RECVIF: Trả về địa chỉ đích/tên giao diện mạng của gói tin UDP với lệnh *recvmsg()*

Các tùy biến TCP socket

- TCP_KEEPALIVE:
  - Thiết lập khoảng thời gian rỗi đối với tùy biến SO_KEEPALIVE
- TCP_MAXSEG:
  - Thiết lập/Lấy kích thước khung tin tối đa gửi bởi TCP socket
    - Việc thiết lập MSS phụ thuộc vào OS: Chỉ cho phép thiết lập giá trị MSS nhỏ hơn giá trị đã quy ước giữa bên gửi và nhận
- TCP_NODELAY
  - Bỏ thiết lập sử dụng thuật toán TCP's Nagle
    - Làm trễ các gói tin nhỏ nếu có các gói tin chờ ACK
  - Tùy biến này cũng bỏ thiết lập sử dụng thuật toán làm trễ ACKS
  - Sử dụng với các ứng dụng tương tác người – máy như rlogin hay telnet

Thiết lập tùy biến file với hàm fcntl

- Viết tắt của "file control“, dùng để thiết lập các tùy biến liên quan đến truy cập file
- Thiết lập các đặc tính xuất nhập dữ liệu qua socket như sau:
  - Xuất nhập dữ liệu socket không bịchặn
    - Thiết lập cờ trạng thái O_NONBLOCK bằng lệnh F_SETFL
  - Xuất nhập dữ liệu socket dựa tín hiệu
    - Thiết lập cờ trạng thái O_ASYNC bằng lệnh F_SETFL
    - Sinh ra tín hiệu SIGIO khi trạng thái của socket thay đổi
  - Thiết lập quyền sở hữu (owner) của socket
    - Sử dụng lệnh F_SETOWN
    - Tiến trình hoặc nhóm tiến trình được gán quyền sẽ nhận các tín hiệu liên quan đến socket như SIGIO hoặc SIGURG

Thiết lập tùy biến file với hàm fcntl (2)

```
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /* int arg */ );
                    Returns: depends on cmd if OK, -1 on error
```

- Mỗi mô tả file (bao gồm cả socket) có một tập các cờ hiệu có thể lấy ra bằng lệnh F_GETFL và thiết lập giá trị bằng lệnh F_SETFL

Ví dụ về cách sử dụng fcntl

- Cho phép xuất nhập dữ liệu không đồng bộ, sử dụng fcntl:

```
int flags;
 /* Set a socket as nonblocking */
if ( (flags = fcntl (fd, F_GETFL, 0)) < 0)
     err_sys("F_GETFL error");
flags |= O_NONBLOCK;
if (fcntl(fd, F_SETFL, flags) < 0)
     err_sys("F_SETFL error");
```

không phải là:

```
if (fcntl(fd, F_SETFL, O_NONBLOCK) < 0)
          err_sys("F_SETFL error");
```

Thiết lập tùy biến với hàm ioctl

- Hàm ioctl
  - Thường sử dụng để làm việc với giao diện hệ thống
  - Phụ thuộc vào cách thức làm việc của hệ điều hành (implementation-dependent)
- Trong lập trình mạng, hàm ioctl thường được dùng để lấy thông tin về giao diện mạng:
  - địa chỉ IP của giao diện mạng
  - có hay không việc hỗ trợ quảng bá (broadcasting)
  - có hay không việc hỗ trợ quảng phát (multicasting)
  - …

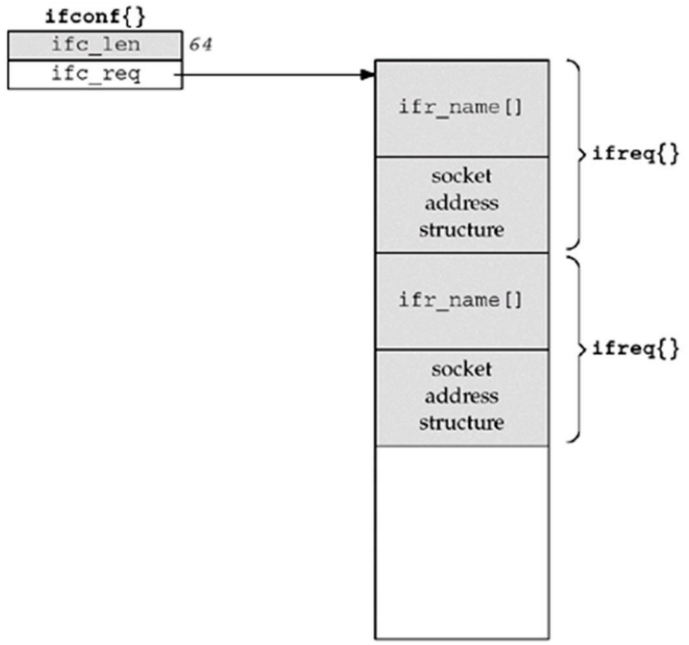
Thiết lập tùy biến với hàm ioctl

```
#include <unistd.h>
int ioctl(int fd, int request, ... /* void *arg */ );
                                        Returns:0 if OK, -1 on error
```

- Các giá trị của biến *requests* liên quan đến mạng có thể chia ra làm 6 loại:
  - Hoạt động của socket
  - Hoạt động của file
  - Hoạt động của giao diện
  - Hoạt động của ARP cache
  - Hoạt động của bảng định tuyến
  - Hệ thống STREAMS

Ví dụ về lấy cấu hình giao diện mạng

- Dữ liệu được lưu dưới dạng cấu trúc *ifconf*



Ví dụ về lấy cấu hình giao diện mạng (2)

- *lib/get_ifi_info.c*
- Sử dụng yêu cầu SIOCGIFCONF để lấy cấu trúc địa chỉ socket của mỗi giao diện
- Kích thước dữ liệu không xác định được trước
  - Tăng dần kích thước buffer cho đến khi dữ liệu lấy được chứa vừa trong buffer
- Các tham số khác của giao diện lấy bằng các yêu cầu khác nhau
  - SIOCGIFFLAGS: cờ giao diện mô tả trạng thái, đặc tả giao diện
  - SIOCGIFMTU: trả về MTU
- Dữ liệu của mỗi giao diện tùy thuộc vào hệ điều hành
  - hỗ trợ/không hỗ trợ

Xử lý tín hiệu POSIX

Giới thiệu

- Đôi khi hệ điều hành muốn thông báo cho tiến trình về một sự kiện
  - sự kiện I/O, kết thúc tiến trình, lỗi phần cứng, …

Hey, I have something for you

process

Giới thiệu (2)

- Đôi khi một tiến trình muốn truyền một thông báo nào đó cho một tiến trình khác hoặc đến chính nó

Send to my future

Tín hiệu

- Một thông báo đến tiến trình về một sự kiện xuất hiện
  - còn được gọi là ngắt mềm (software interrupts)
- Xảy ra không đồng bộ
  - Một tiến trình không biết trước chính xác thời gian một tín hiệu xuất hiện

Ba cách thức xử lý tín hiệu

- Gọi hàm xử lý mỗi khi nhận được tín hiệu
  - Một số ít tín hiệu như SIGIO, SIGPOLL, and SIGURG đòi hỏi một số xử lý thêm khi bắt tín hiệu
  - Không thể bắt được hai tín hiệu SIGKILL và SIGSTOP
- Bỏ qua tín hiệu
  - Hai tín hiệu SIGKILL và SIGSTOP không thể bỏ qua
- Thiết lập cách xử lý tín hiệu mặc định
  - Thường là xử lý kết thúc tiến trình khi nhận được tín hiệu
  - Có một số tín hiệu như SIGCHLD and SIGURG không xử lý bằng mặc định

Thiết lập xử lý tín hiệu bằng hàm *sigaction*

```
int sigaction (int signum, const struct sigaction *act,
                                    struct sigaction *oact)
```

- *signum*: số hiệu của tín hiệu
- *act*: cách thức xử lý tín hiệu *(có thể là null)*
  - SIG_IGN: bỏ qua
  - SIG_DFL: xử lý mặc định
- *oact:* cách thức xử lý tín hiệu trước đó *(có thể là null)*

Cấu trúc sigaction

- struct sigaction {
  void (*sa_handler)(int);
  void (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t sa_mask;
  int sa_flags;
  void (*sa_restorer)(void);
  }
- *sa_handler:* con trỏ trỏ đến hàm xử lý khi có tín hiệu
  - Hàm xử lý chỉ có một tham số là một số nguyên chỉ số hiệu của tín hiệu và không trả về giá trị
- *sa_sigaction:* cũng chỉ định cách thức xử lý tín hiệu nhưng có các tham số khác
- *sa_mask:* danh sách các tín hiệu khác sẽ bị chặn khi đang xử lý tín hiệu
- *sa_flags:* chỉ định tập các cờ quy định sự thay đổi trong xử lý tín hiệu
  - SA_RESTART
- *sa_restorer*: không dùng

Hàm signal

- Một cách thức đơn giản hơn để thiết lập cách xử lý tín hiệu
  - phụ thuộc vào OS

```
#include        "unp.h"
Sigfunc * signal(int signo, Sigfunc *func){
    struct sigaction        act, oact;

    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```

Xuất nhập dữ liệu nâng cao

Giới thiệu

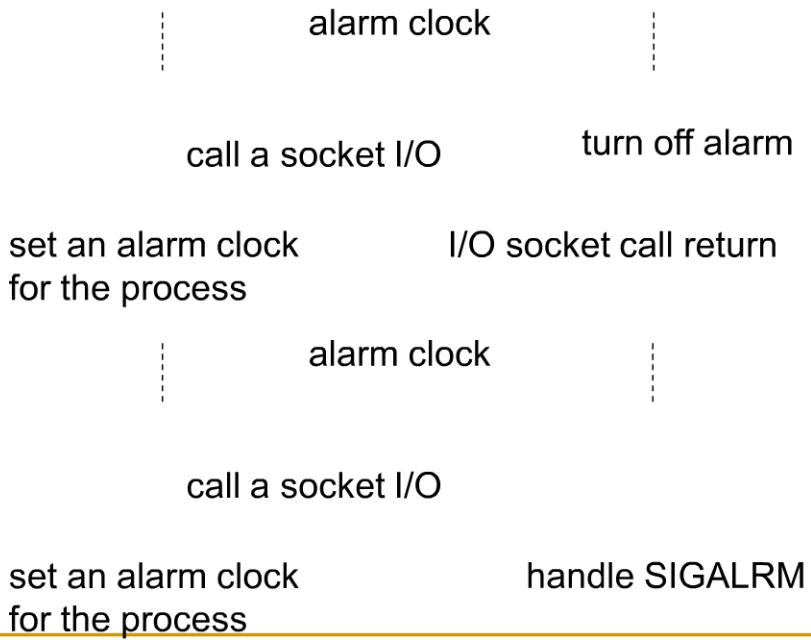
- Chúng ta có thể xuất nhập dữ liệu qua socket với các hàm read(), write() nhưng…
  - Chúng ta không có tùy biến
- Chúng ta có thể cần
  - thiết lập timeout cho một lệnh gọi xuất nhập socket
  - tiến trình không bị dừng khi xuất nhập dữ liệu socket
  - gửi và nhận dữ liệu từ một hoặc nhiều buffers
  - …

Socket Timeouts

- Các cách đặt timeout cho một cuộc gọi xuất nhập dữ liệu socket
  - Gửi thông báo (alarm)
    - sinh ra tín hiệu SIGALRM khi hết một thời gian định trước
  - Sử dụng hàm select chờ xuất nhập dữ liệu trong một thời gian định trước
    - học trong các bài sau
  - Sử dụng tùy chọn socket SO_RCVTIMEO và SO_SNDTIMEO

Xuất nhập dữ liệu socket với timeout sử dụng SIGALRM

- Thiết lập việc gửi tín hiệu SIGALRM và thiết lập hàm xử lý tín hiệu SIGALRM



connect with a timeout using SIGALRM

- lib/connect_timeo.c

recvfrom with a Timeout Using SIGALRM

- advio/dgclitimeo3.c

Xuất nhập dữ liệu socket với timeout sử dụng tùy biến socket SO_RCVTIMEO

- Tùy biến socket SO_RCVTIMEO
  - cho phép đặt timeout cho việc nhận dữ liệu tại socket
  - Tham số của hàm *sockopt* là con trỏ trỏ đến cấu trúc thời gian *timeval*
  - Thiết lập được áp dụng cho tất cả thời gian

Xuất nhập dữ liệu socket với timeout sử dụng tùy biến socket SO_RCVTIMEO

- advio/dgclitimeo2.c

Các vấn đề với thiết lập timeout

- Thiết lập thời gian timeout như thế nào?
  - thời gian trễ giữa hai peer
  - Thời gian xử lý tại server
  - …
- Xử lý khi hết thời gian timeout như thế nào?
  - thực hiện lại phiên truyền tin?
  - hoặc gửi lại thông báo?

Ba phiên bản của read() và write()

- recv() and send()
  - cho phép thiết lập tham số thứ 4 chứa các cờ gửi đến kernel
- readv() and writev()
  - cho phép chỉ định chuỗi buffer để nhận hoặc gửi dữ liệu
- recvmsg() and sendmsg()
  - kết hợp tất cả các đặc điểm của các hàm xuất nhập dữ liệu khác

Các hàm recv and send

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buff, size_t nbytes, int flags);
ssize_t send(int sockfd, const void *buff, size_t nbytes, int flags);
                    Both return: number of bytes read or written if OK, –1 on error
```

- thiết lập cờ chức năng cho việc xuất nhập dữ liệu
  - MSG_DONTROUTE: bỏ qua truy vấn bảng định tuyến
  - MSG_DONTWAIT: không chờ khi xuất nhập dữ liệu
  - MSG_OOB: gửi và nhận dữ liệu ngoài luồng (out-of-band)
  - MSG_PEEK: kiểm tra dữ liệu vào nhưng vẫn giữ nguyên dữ liệu trong buffer
  - MSG_WAITALL: đợi tất cả dữ liệu yêu cầu

Các hàm readv and writev

- Cho phép gửi và nhận dữ liệu từ một hoặc nhiều bộ đệm trong một lần gọi hàm
- Có thể dùng với các đặc tả file khác nhau, không chỉ với socket

```
#include <sys/uio.h>
ssize_t readv(int fd, const struct iovec *iov, int iovcnt);
ssize_t writev(int fd, const struct iovec *iov, int iovcnt);
                        Both return: number of bytes read or written, –1 on error
```

```
struct iovec
 {
    void *iov_base; /* địa chỉ bắt đầu của buffer */
    size_t iov_len; /* kích thước buffer */
};
```

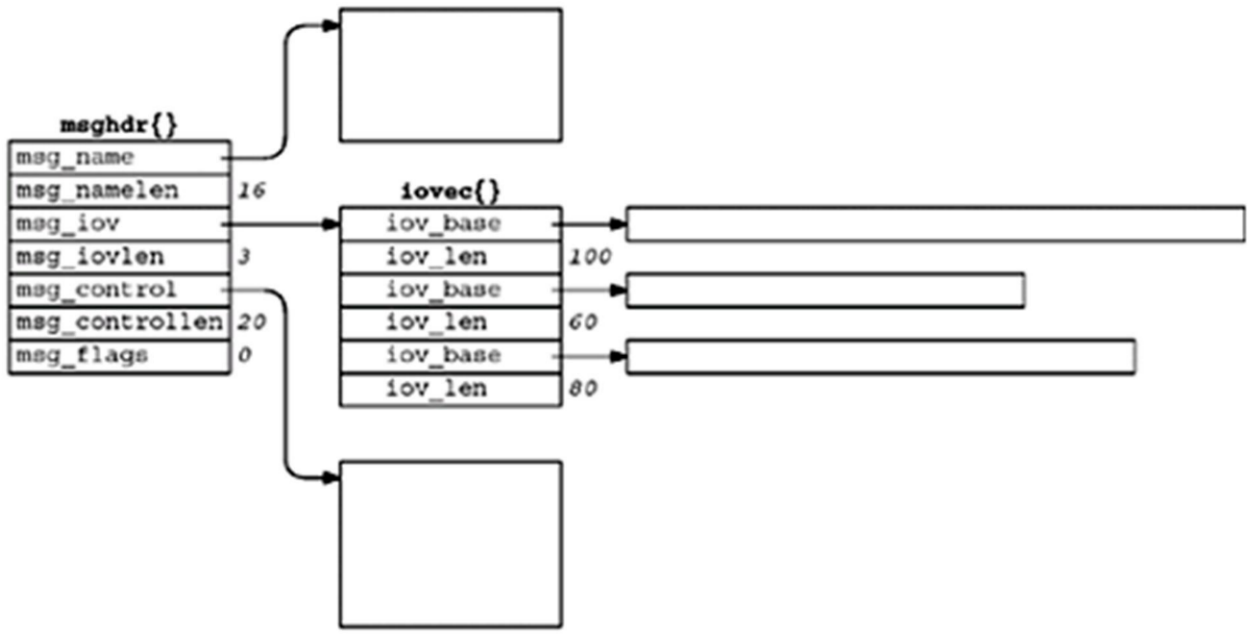
Hàm recvmsg và sendmsg

- Dạng tổng quát của tất cả các hàm xuất nhập

```
#include <sys/socket.h>
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
ssize_t sendmsg(int sockfd, struct msghdr *msg, int flags);
                    Both return: number of bytes read or written if OK, –1 on error
```

```
struct msghdr {
        void *msg_name;             /* protocol address */
        socklen_t msg_namelen;     /* size of protocol address */
        struct iovec *msg_iov;      /* scatter/gather array */
        int msg_iovlen;             /* # elements in msg_iov */
        void *msg_control;          /* ancillary data (cmsghdr struct) */
        socklen_t msg_controllen;  /* length of ancillary data */
        int msg_flags;              /* flags returned by recvmsg() */
};
```

Trước khi gọi hàm recvmsg



Sau khi hàm recvmsg trả kết quả về