Network Programming: Ch. 14: Advanced I/O Functions

Li-Hsing Yen
NYCU
Ver. 1.0.0

Ch. 14: Advanced I/O Functions

- Socket timeouts
- `recv` and `send` functions
- `readv` and `writev` functions
- `recvmsg` and `sendmsg` functions
- Ancillary data

14.2 Socket Timeouts

- Three ways to place a timeout on an I/O operation involving a socket
  1. Call `alarm` before calling socket functions. Kernel will generate `SIGALRM` when time is up.
  2. Block waiting for I/O in `select` which has a time limit built in (Ch. 6)
  3. Set `SO_RCVTIMEO` and `SO_SNDTIMEO` socket options by `setsockopt` (Ch. 7)

Socket Timeout Examples

- `connect` with a Timeout Using `SIGALRM` (pp. 5~6)
- `recvfrom` with a Timeout Using `SIGALRM` (p. 7)
- `recvfrom` with a Timeout Using `select` (pp. 8~10)
- `recvfrom` with a Timeout Using `SO_RCVTIMEO` Socket Option (pp. 11~12)

`connect` with a Timeout Using `SIGALRM`

- `alarm` causes the generation of `SIGALRM` signal after a specified time has expired
- Steps to `connect` with a time out
  1. specify a signal handler for the `SIGALRM`
  2. set `alarm` before calling `connect`
  3. call `connect`
  4. `connect` will return negative value (with **`errno`** = `EINTR`) if the alarm expires before `connect` succeeds

connect with a Timeout Using SIGALRM

lib/connect_timeo.c

```
#include "unp.h"

int connect_timeo(int sockfd, const SA *saptr, socklen_t salen, int nsec)
{
        Sigfunc   *sigfunc;
        int    n;

        sigfunc = Signal(SIGALRM, connect_alarm);
        if (alarm(nsec) != 0)
                err_msg("connect_timeo: alarm was already set");
        if ( (n = connect(sockfd, (struct sockaddr *) saptr, salen)) < 0) {
                close(sockfd);
                if (errno == EINTR)
                        errno = ETIMEDOUT;
        }
        alarm(0);                       /* turn off the alarm */
        Signal(SIGALRM, sigfunc);/* restore previous signal handler */
        return(n);
}
```

```
static void connect_alarm(int signo)
{   return;   }
```

儲存原先的signal handler

原先的alarm還在跑

alarm timeout造成中斷

關掉alarm

restore原先的signal handler

recvfrom with a Timeout Using SIGALRM

advio/dgclitimeo3.c

```
void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
        int    n;
        charsendline[MAXLINE], recvline[MAXLINE + 1];

        Signal(SIGALRM, sig_alrm);
        while (Fgets(sendline, MAXLINE, fp) != NULL) {
            Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
            alarm(5);                    ←—— 設定alarm為5秒
            if ( (n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL)) < 0) {
                if (errno == EINTR)      ←—— alarm timeout造成中斷
                        fprintf(stderr, "socket timeout\n");
                else
                        err_sys("recvfrom error");
            } else {
                alarm(0);                ←—— 關掉alarm
                recvline[n] = 0;     /* null terminate */
                Fputs(recvline, stdout);
            } } }
```

```
static void sig_alrm(int signo)
{
        return;
}
```

`recvfrom` with a Timeout Using `select`

- `select` lets us specify the maximal time to wait in reading
  - should prepare arguments before calling `select` (the read descriptor set & `timeval` struct)
- we do not call `recvfrom` until function `readable_timeo` tells us that the descriptor is readable
  - Therefore, `recvfrom` will not block

Function readable_timeo

lib/readable_timeo.c

```
int readable_timeo(int fd, int sec)
{
    fd_set      rset;
    struct timeval tv;

    FD_ZERO(&rset);
    FD_SET(fd, &rset);      <-- 設定要test fd for readability

    tv.tv_sec = sec;        <-- 設定最多要等待的秒數
    tv.tv_usec = 0;

    return(select(fd+1, &rset, NULL, NULL, &tv));
        /* > 0 if descriptor is readable */
}
```

dgi_cli that calls readable_timeo

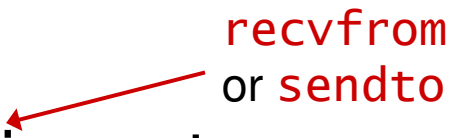
advio/dgclitimeo1.c

```
#include       "unp.h"

void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
       int     n;
       char    sendline[MAXLINE], recvline[MAXLINE + 1];

       while (Fgets(sendline, MAXLINE, fp) != NULL) {

              Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

              if (Readable_timeo(sockfd, 5) == 0) {
                     fprintf(stderr, "socket timeout\n");
              } else {
                     n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
                     recvline[n] = 0;       /* null terminate */
                     Fputs(recvline, stdout);
              } } }
```

do not call `recvfrom` until function `readable_timeo` tells us that the descriptor is readable

recvfrom with a Timeout Using SO_RCVTIMEO Socket Option

- We can set SO_RCVTIMEO option to specify the timeout value for all read operations on that descriptor
- Similarly, SO_SNDTIMEO option applies to all write operations
- If the I/O operation times out, the function returns negative value with **errno** = EWOULDBLOCK



recvfrom with a Timeout Using SO_RCVTIMEO Socket Option

advio/dgclitimeo2.c

```
tv.tv_sec = 5;
tv.tv_usec = 0;
Setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

while (Fgets(sendline, MAXLINE, fp) != NULL) {

Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
if (n < 0) {
        if (errno == EWOULDBLOCK) {
                fprintf(stderr, "socket timeout\n");
                continue;
        } else
                err_sys("recvfrom error");
}
```

設定socket option

等待5秒後尚未收到資料

14.3: recv and send functions: similar to read and write

多一個參數

```
#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
```

flags for I/O functions	Description	recv	send
MSG_DONTROUTE	bypass routing table lookup		V
MSG_DONTWAIT	only this operation is nonblocking	V	V
MSG_OOB	send or receive out-of-band data	V	V
MSG_PEEK	peek at incoming message	V	
MSG_WAITALL	wait for all the data	V	

Set 0, a constant, or logically ORed constants

not a value-result argument

Not return until the requested number of bytes have been read (like readn)

14.4: readv and writev Functions scatter read and gather write:

#include <sys/uio.h>　讀至多個buffer或寫出多個buffer內容

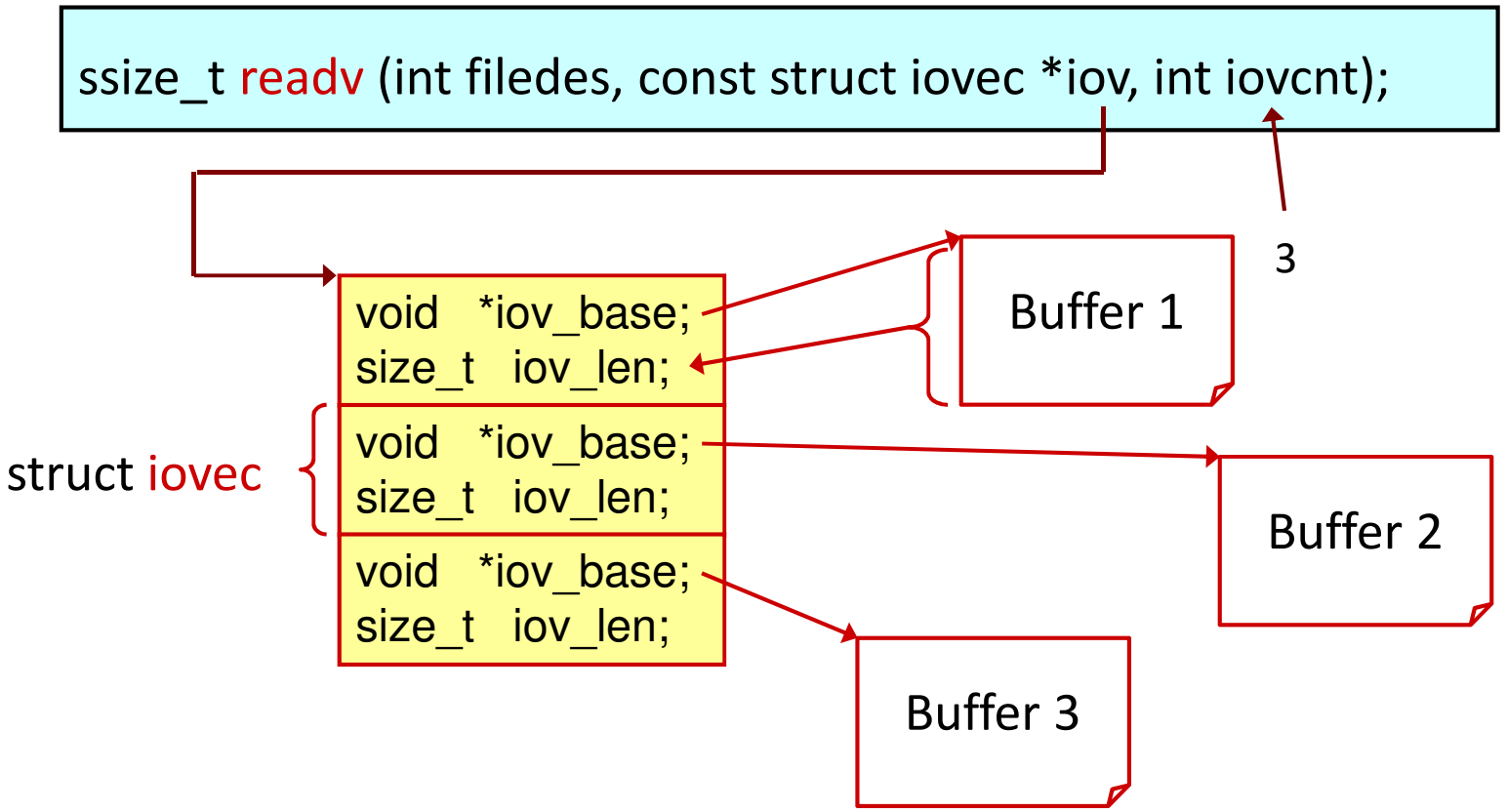
```
ssize_t readv (int filedes, const struct iovec *iov, int iovcnt);
ssize_t writev (int filedes, const struct iovec *iov, int iovcnt);
```

both return: number of bytes read or written, -1 on error

*iov: a pointer to an arrary of iovec structure

```
struct iovec {
  void   *iov_base;   /* starting address of buffer */
  size_t   iov_len;     /* size of buffer */
};
```

Example: 讀data至多個buffer



14.5: recvmsg and sendmsg: the most general socket I/O functions

```
#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

同recv的flags

不一樣 (flags只能設定不能回傳)

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

used with UDP

和readv/writev同

pass per-datagram contl info.

Fields in msghdr Structure

used with UDP

- msg_name指向 socket address structure.
  msg_namelen是此 structure 的 size
- msg_iov指向 array of iovec structure 的位置，
  msg_iovlen是此array 的 element 數目

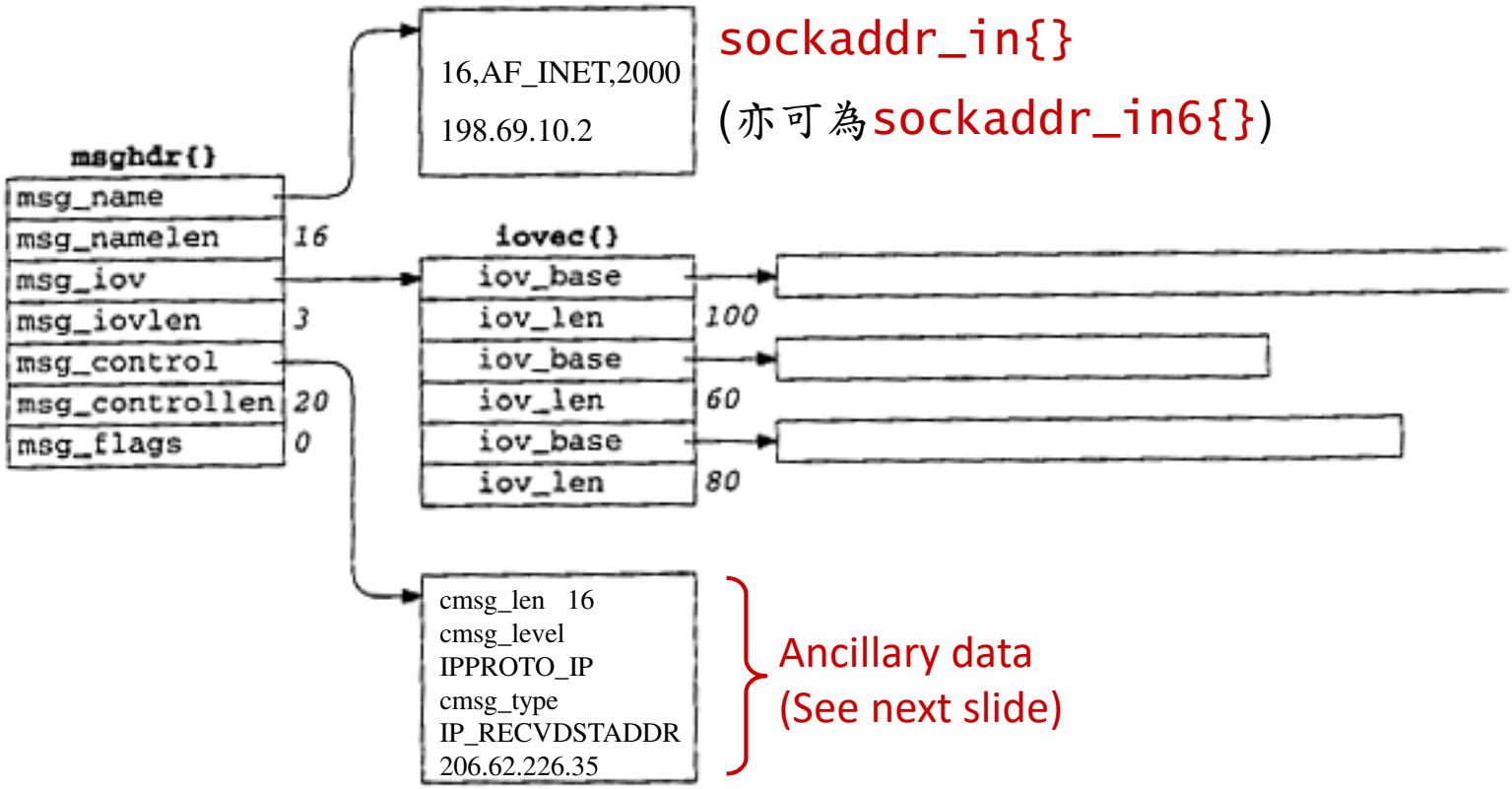
和readv writev同

- msg_control指向第一個ancillary data object 的位置，
  msg_controllen是所有 ancillary data 的 byte 數目
- msg_flags: recvmsg可能回傳的 flag (not used by sendmsg)，詳如下頁

Summary of I/O Flags by Various I/O Functions

Flag	Examined by: *send flags* *sendto flags* *sendmsg flags*	Examined by: *recv flags* *recvfrom flags* *recvmsg flags*	Returned by: recvmsg msg_flags
MSG_DONTROUTE	X		
MSG_DONTWAIT	X	X	
MSG_PEEK		X	
MSG_WAITALL		X	
MSG_EOR	X		X
MSG_OOB	X	X	X
MSG_BCAST	message truncated (因程式準備的buffer太小)		X
MSG_MCAST			X
MSG_TRUNC			X
MSG_CTRUNC	存ancillary data的buffer太小		X

msghdr after recvmsg returns (An Example)

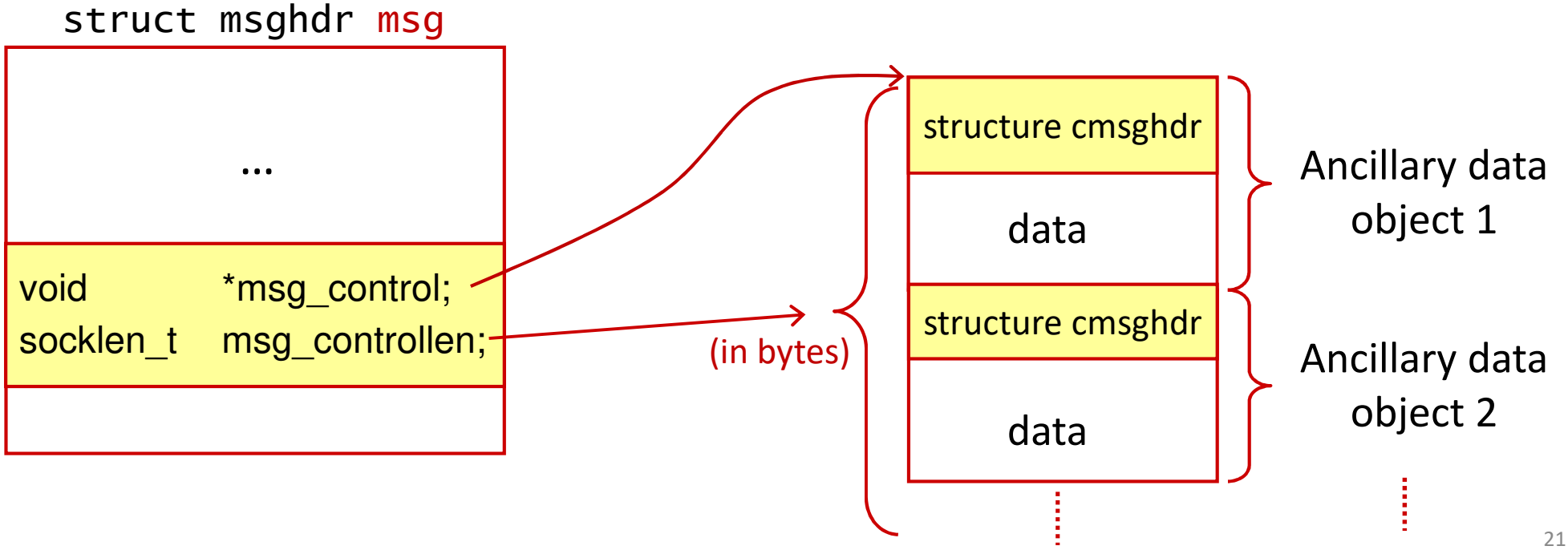


Comparison of Five Groups of I/O Functions

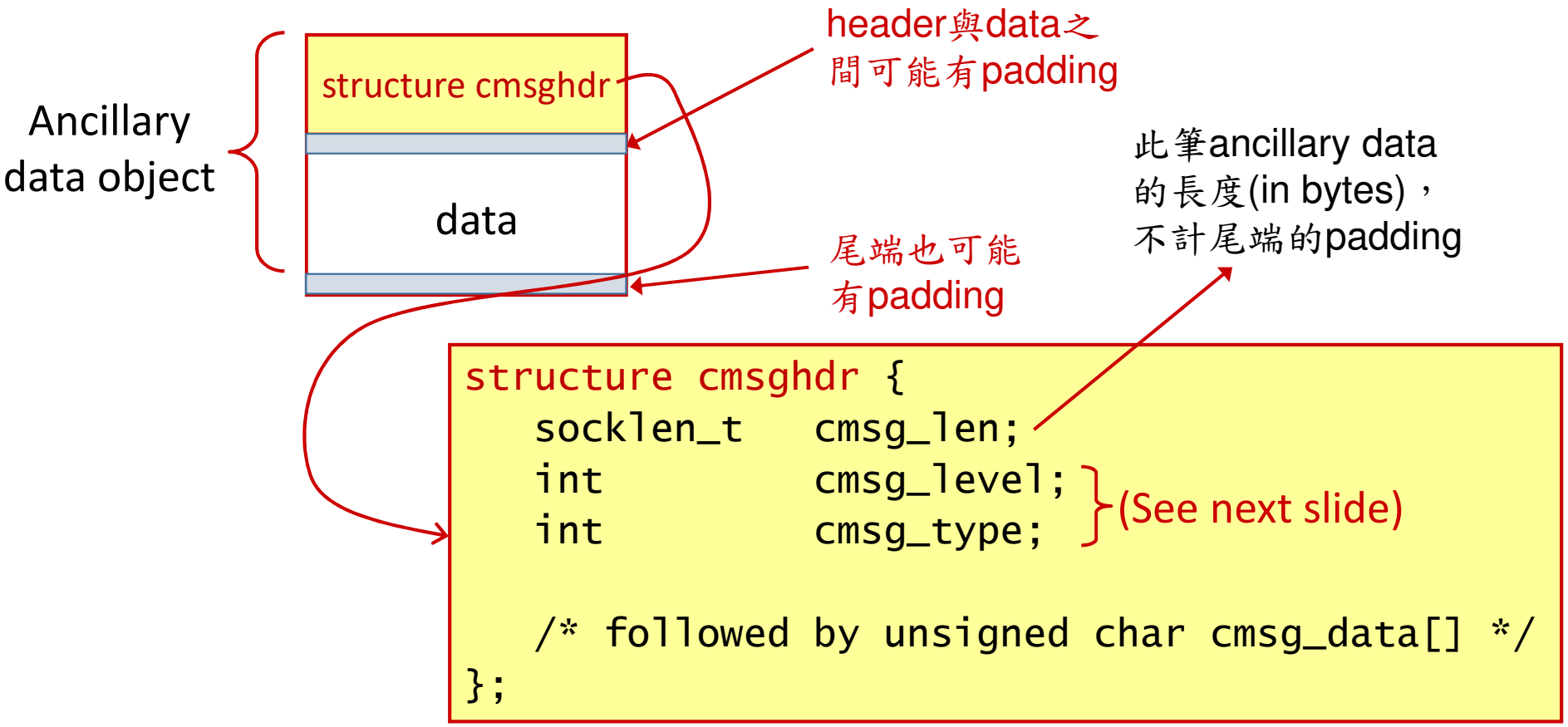
Functions	Any descriptor	Only socket descriptor	Single buffer	Scatter buffer	Optional flags	Optional peer address	Optional control info.
read, write	v		v				
readv, writev	v			v			
recv, send		v	v		v		
recvfrom, sendto		v	v		v	v	
recvmsg, sendmsg		v		v	v	v	v

14.6 Ancillary Data

- Ancillary data consists of one or more *ancillary data objects*, each one beginning with a `cmsghdr` structure (後面還有 data)



cmsghdr Structure




```
structure cmsghdr {
    socklen_t   cmsg_len;
    int         cmsg_level;
    int         cmsg_type;

    /* followed by unsigned char cmsg_data[] */
};
```

Ancillary Data (Control Info)

Uses of Ancillary data in our text

Protocol	cmsg_level	cmsg_type	Description
IPv4	IPPROTO_IP	IP_RECVDSTADDR	receive dest addr with UDP data
		IP_RECVIF	receive interface index with UDP
IPv6	IPPROTO_IPV6	IPV6_DSTOPTS	specify/receive dest options
		IPV6_HOPLIMIT	specify/receive hop limit
		IPV6_HOPOPTS	specify/receive hop-by-hop options
		IPV6_NEXTHOP	specify next-hop addr
		IPV6_PKTINFO	specify/receive packet info
		IPV6_RTHDR	specify/receive routing header
Unix domain	SOL_SOCKET	SCM_RIGHTS	send/receive descriptors
		SCM_CREDS	send/receive user credentials

We can get source addr. via `msg_name`

處理Ancillary Data的五個巨集(Macros)

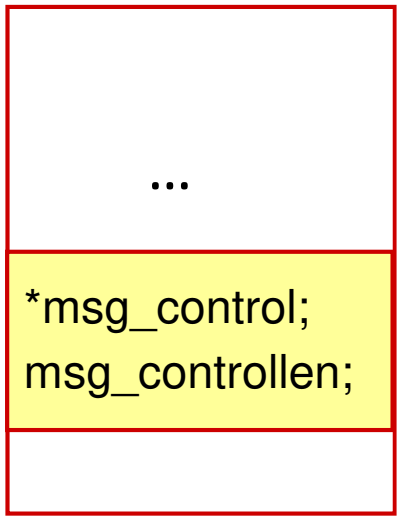
include <sys/socket.h> and <sys/param.h>

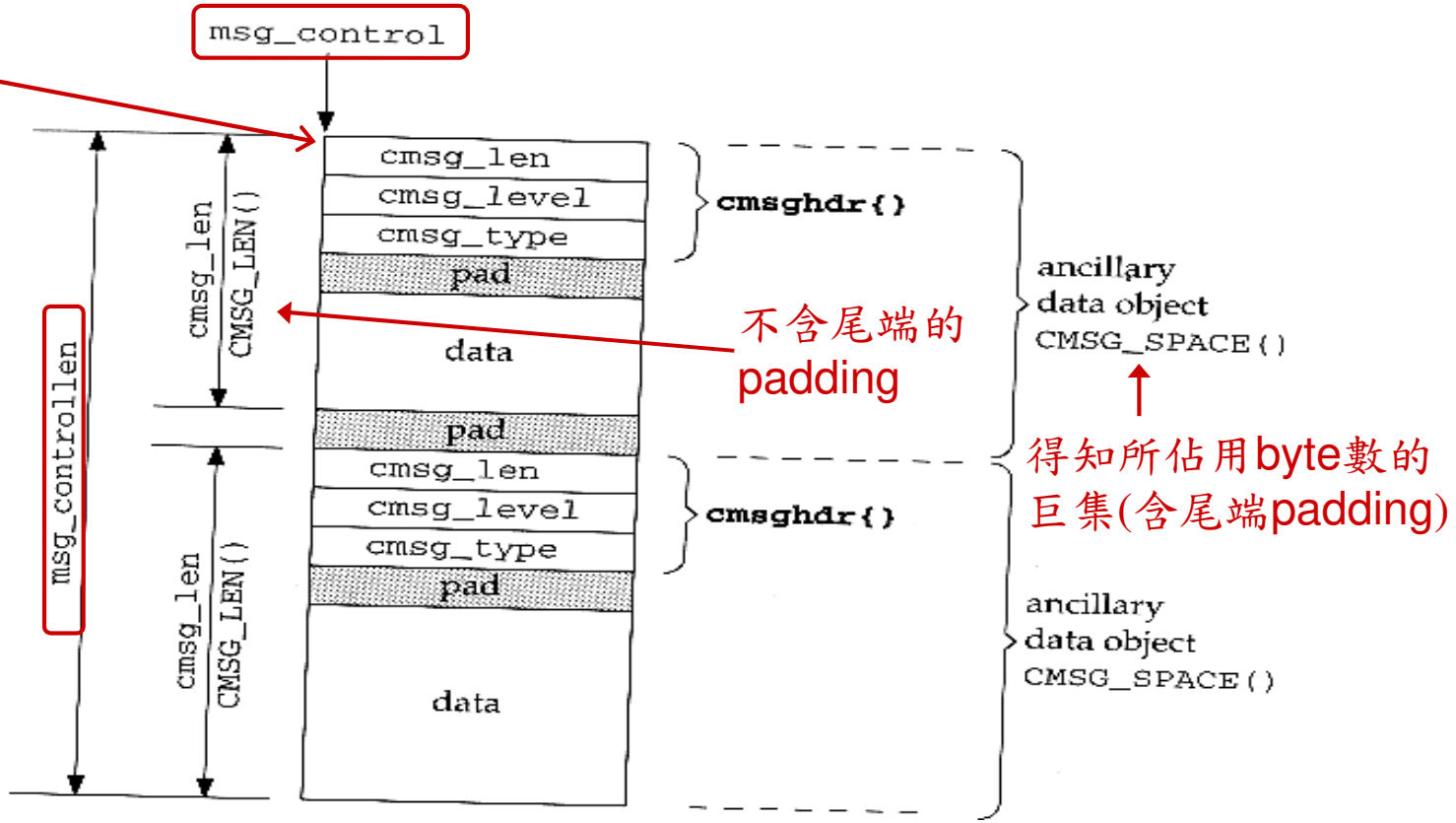
- struct cmsghdr *CMSG_FIRSTHDR(struct msghdr **mhdrptr*) 指向第一個cmsghdr結構的指標
- struct cmsghdr *CMSG_NXTHDR(struct msghdr **mhdrptr*, struct cmsghdr **cmsgptr*) 指向下一個cmsghdr結構的指標(NULL表沒有下一個了)
- unsigned char *CMSG_DATA(struct cmsghdr **cmsgrptr*) 指向cmsgptr指過去結構的data的first byte
- unsigned int CMSG_LEN(unsigned int length) 應存放在cmsg_len欄位的值(不含data尾端的padding)
- unsigned int CMSG_SPACE(unsigned int length) 真正data object的長度(含data尾端的padding)

Ancillary Data Containing Two Ancillary Data Objects

CMSG_FIRSTHDR(msg)

```
struct msghdr msg
```





Macros Used to Handle Ancillary Data

```
struct msghdr msg;
struct cmsghdr *cmsghdr;

/* fill in msg structure and call recvmsg () */
for (cmsghdr = CMSG_FIRSTHDR(&msg); cmsghdr != NULL;
        cmsghdr = CMSG_NXTHDR(&msg, cmsghdr)) {
          if (cmsghdr->cmsg_level == … && cmsghdr->cmsg_type == … ) {
                u_char *ptr;

                ptr = CMSG_DATA(cmsgptr);
                /* process data pointed to by ptr */
          }
}
```

14.7 How Much Data is Queued?

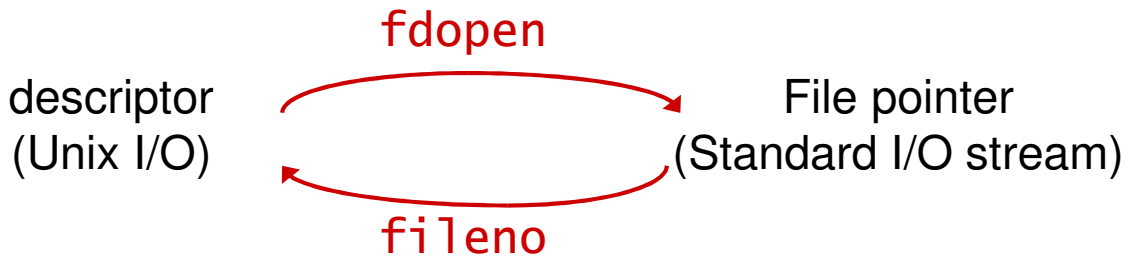
- Examine the data but still leave it on the receiving queue
  - Use the MSG_PEEK flag (p. 18)
- But not sure whether data is ready to be read (and we don't want to block in this)
  - Combine MSG_PEEK with a non-blocking socket, or
  - ORed with the MSG_DONTWAIT flag (p. 18)

The Amount of Data

- For TCP, the amount of data on the receive queue can change between two successive calls to `recv` (第一次 PEEK，第二次真正讀取)
  - More data can be received by TCP between the two calls
- For UDP, the return values from both calls will be the same

14.8 Socket and Standard I/O

- We have used *Unix I/O*, normally implemented as system calls within Unix
  - read, write, recv, send, …
- *Standard I/O library* specified by ANSI C standard provides portability
  - fgets, fputs, fseek, rewind, …



Example: `str_echo` Using Standard I/O

advio/str_echo_stdio02.c

```
#include      "unp.h"

void
str_echo(int sockfd)
{
        char            line[MAXLINE];
        FILE            *fpin, *fpout;

        fpin = Fdopen(sockfd, "r");
        fpout = Fdopen(sockfd, "w");

        while (Fgets(line, MAXLINE, fpin) != NULL)
                Fputs(line, fpout);
}
```

轉成標準I/O

使用標準I/O讀寫

The Full-Duplex Issue

- TCP and UDP are full-duplex (全雙工)
- Standard I/O streams can also be full-duplex (open the stream with a type of r+)
  - 但有許多限制，如read後不能直接write，中間需間隔有fseek, fsetpos, 或 rewind
  - 這些functions在socket上行不通
  - 將socket轉成standard I/O stream又要保持full-duplex性質需要一些技巧

Using Standard I/O on a Full-Duplex Socket

- Open two standard I/O streams for a given socket
  - One for reading and one for writing

```
FILE *fpin, *fpout;

fpin = Fdopen(sockfd, “r”);
fpout = Fdopen(sockfd, “w”);
…
fgets(line, MAXLINE, fpin);
…
fputs(line, fpout);
```

The Buffering Problem

- Standard I/O is fully buffered
  - I/O takes place only when the buffer is full (unless the process call `fflush`)
  - Causes problems in typical network applications
- Solutions
  - Force the output stream to be line buffered by calling `setvbuf`
  - Or call `fflush` after each call to `fputs`