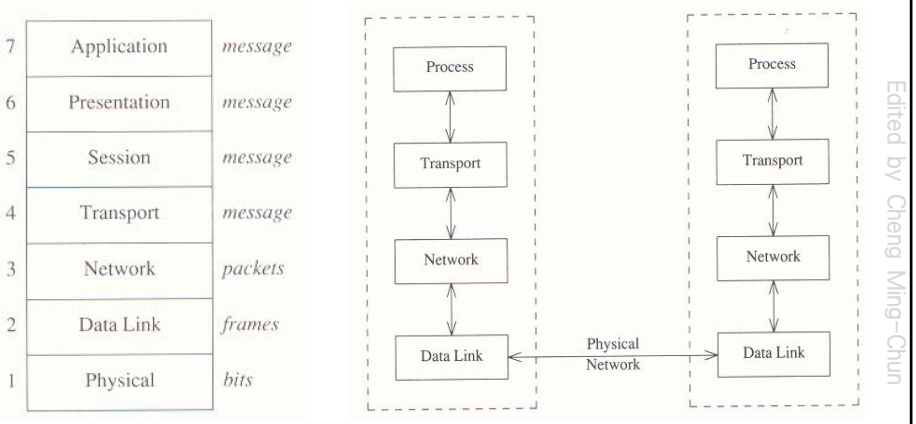
十、Socket 程式設計

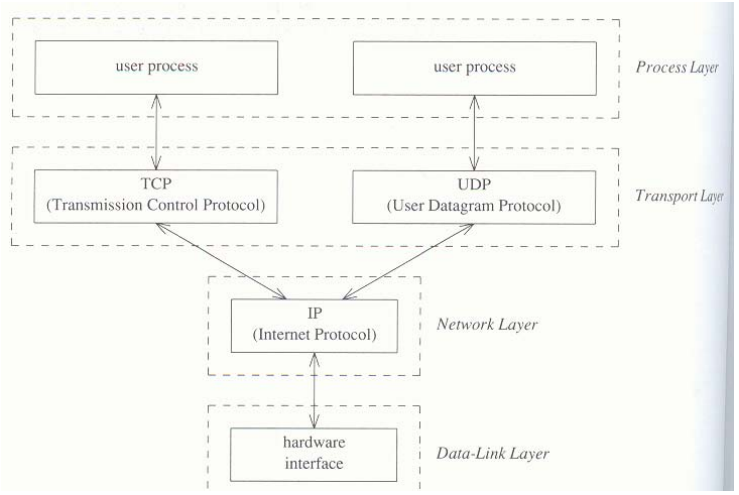
網路概述(1/4)

- OSI 七層與簡化的四層架構



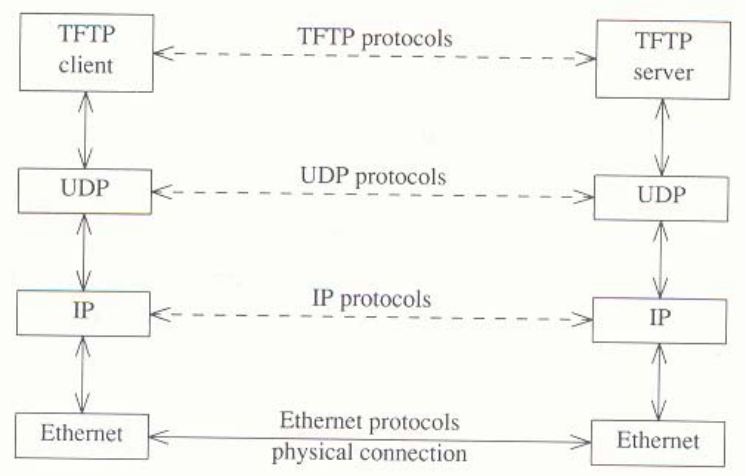
網路概述(2/4)

- TCP/IP四層架構



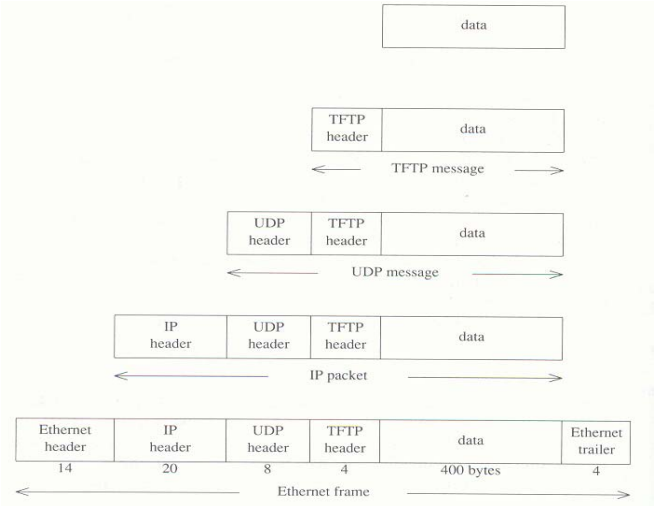
網路概述(3/4)

- TFTP應用程式



網路概述(4/4)

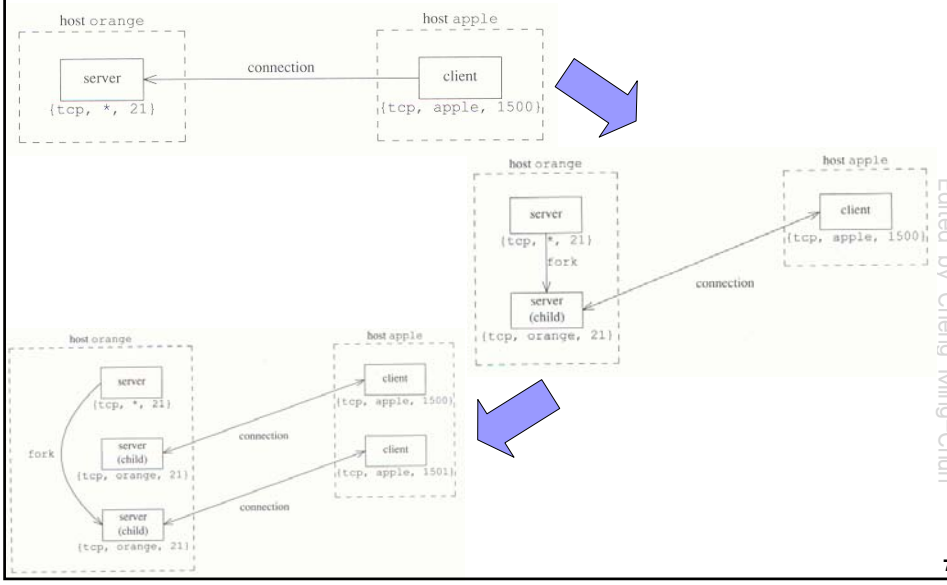
- 各層的header



連線關係(1/2)

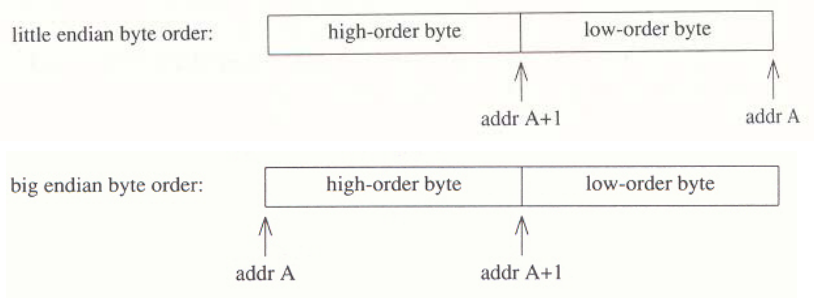
- 可以用一個5-tuple來代表兩個行程間的連線關係
  {protocol, local-addr, local-process, foreign-addr, foreign-process}
  - local-addr和foreign-addr代表網路位址
  - local-process和foreign-process代表port(每個行程使用不同的port)
  - 範例
    {tcp, 192.43.235.2, 1500, 192.43.235.6, 21}
- 可以用一個3-tupe來代表socket的位址
  - local address: {protocol, local-addr, local-process}
  - foreign address: {protocol, foreign-addr, foreign-process}
  - 上面兩個socket位址可以組合成完整的連線關係

連線關係(2/2)



Big endian 與 little endian

- 佔用兩個位元組以上的資料型態必須注意該硬體架構是big endian或是little endian
- 在x86的機器上採用的是Least Significant Byte First，也就是little endian，而網路上傳輸的資料採用的是Most Significant Byte First，也就是big endian



Socket概述

- linux的socket介面支援下列通訊協定

名稱	目的	Man pages
PF_UNIX PF_LOCAL	UNIX domain	unix(7)
PF_INET	IPv4 Internet	ip(7)
PF_INET6	IPv6 Internet	Ipv6(7)
PF_IPX	IPX – Novell	
PF_NETLINK	Kernel user interface device	netlink(7)
PF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
PF_AX25	Amateur radio AX.25 protocol	
PF_ATMPVC	Access to raw ATM PVCs	
PF_APPLETALK	Appletalk	ddp(7)
PF_PACKET	Low level packet interface	packet(7)

Unix Domain Protocol介紹

- 跟其它通訊協定不同，只能用在同一個host上的行程通訊。事實上前一章IPC中的pipe就是用這種方式來實作
- 同時提供connection-oriented與connectionless的介面，由於它並沒有透過真正的線路連到另一台機器，所以傳送的訊息一定是正確的(不用checksum)。一般使用上建議使用前者，因為connectionless沒有flow-control，傳送的速度可能超過緩衝區消耗的速度，因此當緩衝區滿時，發送端必須重送。connection-oriented的方式由於有flow-control所以沒有這個問題
- Unix domain protocol有一個只有它有，其它協定都沒有的功能，它可以將某個行程的存取權限(SCM_RIGHTS)或是執行身份(SCM_CREDENTIALS)傳給其它行程

不同通訊協定的位址資訊(1/2)

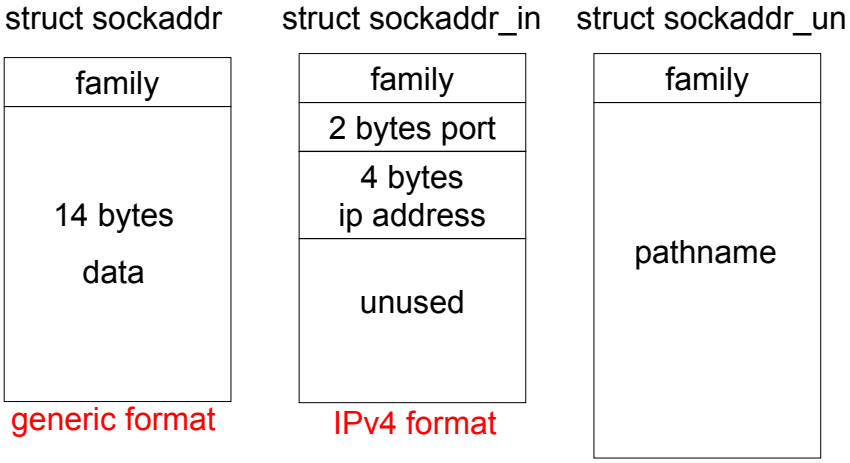
- 不同的通訊domain會有不同的位址資訊，因此會有不同的資料結構來存放這些位址，例如PF_INET採用sockaddr_in，PF_UNIX採用sockaddr_un

```
struct sockaddr {
        sa_family_t  sa_family;          /* address family */
        char sa_data[14];                /* protocol-specific address */
};
```

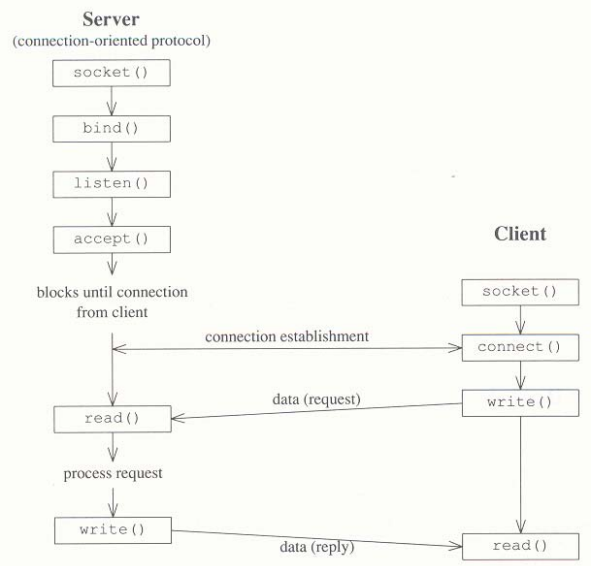
```
struct sockaddr_in {
        sa_family_t  sin_family;         /* address family: AF_INET */
        u_int16_t  sin_port;             /* port in network byte order */
        struct in_addr  sin_addr;        /* internet address */
};
struct in_addr {                         /* Internet address. */
        u_int32_t  s_addr;               /* address in network byte order */
};
```

```
#define UNIX_PATH_MAX    108
struct sockaddr_un {
        sa_family_t  sun_family;         /* AF_UNIX */
        char  sun_path[UNIX_PATH_MAX];   /* pathname */
};
```

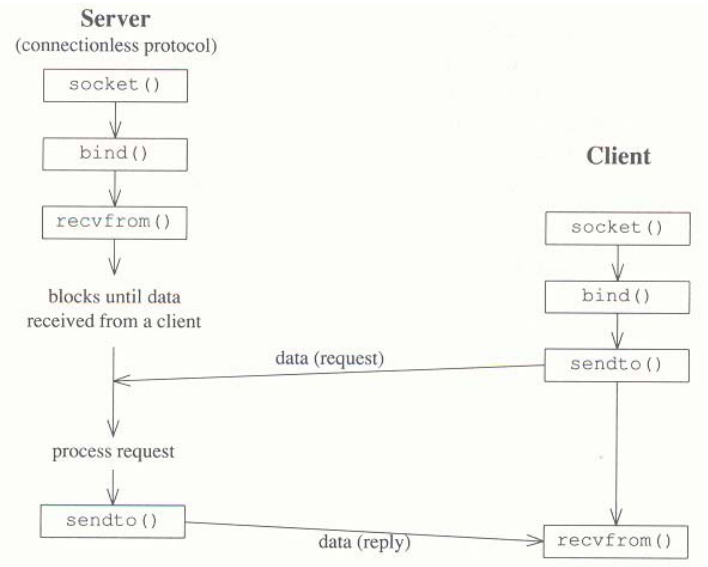

不同通訊協定的位址資訊(2/2)



Connection-oriented protocol



Connectionless protocol



建立socket descriptor

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

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

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- socket函式用來建立通訊端點，執行成功後會傳回file descriptor回來。這個函式通常是使用socket介面時所呼叫的第一個函式
- socket函式有三個參數，其中第一個參數指定使用何種通訊家族(例如是TCP/IP或是UNIX Domain)，第二個參數指定通訊的方式(例如是connection-oriented或是connectionless)，第三個參數指定使用特定的某個通訊協定，通常第一個參數加上第二個參數就只剩一種通訊協定可用，所以第三個參數通常填0即可
- 第一個參數可以的值包括PF_UNIX，PF_LOCAL，PF_INET，PF_INET6，PF_IPX，PF_NETLINK，PF_X25，PF_AX25，PF_ATMPVC，PF_APPLETALK，PF_PACKET
- 第二個參數根據不同的第一個參數能有不同的值，第三個參數可用的值也跟第一個與第二個參數相關，下表列出它們之間的關係

domain	type	protocol	實際上的協定
PF_INET	SOCK_DGRAM	IPPROTO_UDP或0	UDP
	SOCK_STREAM	IPPROTO_TCP或0	TCP
	SOCK_RAW	IPPROTO_ICMP	ICMP
		IPPROTO_RAW	raw
PF_UNIX	SOCK_DGRAM	0	Unix domain
	SOCK_STREAM	0	Unix domain

將socket命名(設定local address)

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

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

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- bind函式用來將socket命名，也就是指定該socket的近端位址(local address)
- bind函式有三個參數，第一個參數為要命名的socket descriptor，第二個參數為要命名的網路位址(socket address)，每一種通訊domain有不同的位址結構，第三個參數為該位址結構的長度
- 使用bind函式有下列幾個時機
  - 如果是server端程式，會使用bind函式來註冊所要的網路位址。這相當於告訴作業系統，以後傳送到該位址的資料要給我，無論是connection-oriented或connectionless的server端程式都需要呼叫這個函式
  - 如果是client端程式，也可以指定特定的local address
  - 如果connectionless的client端程式需要server回傳某些資料時，必須使用bind註冊一個網路位址，這樣server才可以送資料回來

連線到另一端的socket(設定foreign address)

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

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

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- client端程式透過呼叫connect函式與另一端的socket建立連線
- connect函式有三個參數，第二個參數為另一端的位址資訊
- connect函式可以用在connection-oriented的協定(TCP)也可以用在connectionless的協定(UDP)上，但有不同的效果
  - connection-oriented
    - 它會真正的與另一端點建立連線(例如TCP three-way handshake)，當連線建立完成後才會返回，如果sockfd設定為non-blocking，其errno會設定為EINPROGRESS
    - 如果在呼叫connect函式之前沒有呼叫bind函式來指定local address，則connect函式會動態指定local address
  - connectionless
    - 用來指定之後的資料要送到哪裡去，透過connect函式設定之後就不需在每個傳送或接收的指令(read,write,recv,send)中重新指定，由於沒有實際建立連線，因此呼叫connect後會馬上返回
    - 可以呼叫多次來改變傳送的目的地
    - 使用connect函式還是另一個好處，當目的地的port沒有開啟時，會收到ICMP port unreachable message，之後呼叫的相關socket函式會傳回錯誤，errno會設為ECONNREFUSED

開始監聽連線請求

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

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

- 這個函式只有connection-oriented (socket type為SOCK_STREAM)的server才需要呼叫，它告知作業系統將開始接收新的連線
- 它只有兩個參數，第一個參數為操作的socket descriptor，第二個參數為能夠queue住多少個connection request
  - 在linux kernel 2.2之後，第二個參數指定的是能夠queue住多少個"已完成"連線的請求，未完成的連線請求可以透過設定tcp_max_syn_backlog來限制
- 每一個connection request會被accept函式處理，尚未處理的connection request會queue住，當queue已經滿時，就會產生connection refused

處理新連線

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

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

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- accept函式用來處理連線請求，這個函式只有在connection-oriented (socket type為SOCK_STREAM)的server端呼叫，通常接在listen函式之後呼叫
- accept函式會從queue中拿取第一個connection request來處理，它會建立一個新的socket descriptor，該socket descriptor為之後用來操作該連線所用
- accept的第二個參數必須事先配置好空間，accept函式會填入另一端點的網路位址(不同family用不同的資料結構)。至於第三個參數有兩個用途，在呼叫accept時，第三個參數用來指定第二個參數的長度(以免buffer overflow)，當呼叫完accept時，第三個參數會由accept函式所設定，代表填入addr的長度
- 當queue中沒有connection request時，accept函式會block住，如果s設定為non-blocking，當沒有connection request時，其errno會設為EAGAIN

傳送與接收訊息

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

    int send(int s, const void *msg, size_t len, int flags);
    int sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);
    int recv(int s, void *buf, size_t len, int flags);
    int recvfrom(int  s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
```

- 可以直接用read或write函式對socket descriptor做讀寫，但是這頁的函式可以設定更多的參數。這些函式的前三個參數與read或write相同
- send與recv只能用在connection-oriented的socket上，其它函式則沒有限制
- sendto函式中的to參數用來指定訊息要傳送到那邊(可以用connect函式先設定好)
- 當sending buffer的空間小於訊息的大小時，寫入相關函式都會被block住(除非該socket為nonblocking)，直到可以完全寫入為止(atomic operation)
- sendto的flags參數可以有下列值(有些值只有在特定的通訊協定才有)
  - MSG_OOB: 傳送Out-of-band的訊息
  - MSG_DONTROUTE:不將訊息送給gateway，而直接送給host，通常用來診斷或是route程式中使用
  - MSG_DONTWAIT:採用non-blocking的方式(也可以透過fcntl, ioctl,setsockopt等函式來設定)
  - MSG_NOSIGNAL:當寫入一個沒有reader的socket時，不產生SIGPIPE訊號，但errno還是會設定成EPIPE
- recvfrom函式的from參數用來傳回接收到訊息的來源位址，該參數可以為NULL
- 當沒有訊息可以讀時，讀取相關函式都會被block住(除非該socket為nonblocking)
- recvfrom的flags參數可以有下列值(有些值只有在特定的通訊協定才有)
  - MSG_OOB:接收Out-of-band的訊息
  - MSG_PEEK:拿取receiving queue中的第一個訊息，但是不將之discard掉(也就是下一次的讀取還是會讀到它)
  - MSG_TRUNC:如果超過設定的長度則截斷訊息

TCP server 範例

CODE. 10-1

```
#include <sys/socket.h>
#include <netinet/in.h>

int main(void){
        struct sockaddr_in server,client;
        int sock,csock,readSize,addressSize;
        char buf[256];

        bzero(&server,sizeof(server));
        server.sin_family=PF_INET;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        server.sin_port=htons(6789);

        sock=socket(PF_INET,SOCK_STREAM,0);                          //產生socket descriptor
        bind(sock,(struct sockaddr *)&server,sizeof(server));

        listen(sock,5);                                              //listen
        addressSize=sizeof(client);
        csock=accept(sock,(struct sockaddr *)&server, &addressSize);  //處理連線，這邊產生新socket

        readSize=read(csock,buf,sizeof(buf));                        //開始讀寫socket
        buf[readSize]=0;
        printf("read message:%s\n",buf);
}
```


TCP client 範例

CODE. 10-1

```
#include <sys/socket.h>
#include <netinet/in.h>

int main(void){
        struct sockaddr_in server;
        int sock;
        char buf[]="TCP TEST";

        bzero(&server,sizeof(server));
        server.sin_family=PF_INET;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        server.sin_port=htons(6789);

        sock=socket(PF_INET,SOCK_STREAM,0);                  //產生socket
        connect(sock,(struct sockaddr *)&server,sizeof(server));  //與server連線

        write(sock,buf,sizeof(buf));                        //寫出資料
}
```

UDP server 範例

CODE. 10-2

```
#include <sys/socket.h>
#include <netinet/in.h>

int main(void){
        struct sockaddr_in server,client;
        int sock,readSize,addressSize;
        char buf[256];

        bzero(&server,sizeof(server));
        server.sin_family=PF_INET;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        server.sin_port=htons(5678);

        sock=socket(PF_INET,SOCK_DGRAM,0);
        bind(sock,(struct sockaddr *)&server,sizeof(server));      //設定local address

        addressSize=sizeof(client);
        readSize=recvfrom(sock,buf,sizeof(buf),0,(struct sockaddr *)&client,&addressSize);
        buf[readSize]=0;
        printf("read message:%s\n",buf);
}
```


UDP client 範例(1/3)

CODE. 10-2

```
#include <sys/socket.h>
#include <netinet/in.h>

int main(void){
        struct sockaddr_in server;
        int sock;
        char buf[]="UDP TEST";

        bzero(&server,sizeof(server));
        server.sin_family=PF_INET;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        server.sin_port=htons(5678);

        sock=socket(PF_INET,SOCK_DGRAM,0);

        sendto(sock,buf,sizeof(buf),0,(struct sockaddr *)&server,sizeof(server));
}
```

UDP client 範例(2/3)

CODE. 10-2

```
#include <sys/socket.h>
#include <netinet/in.h>

int main(void){
        struct sockaddr_in server;
        int sock;
        char buf[]="UDP TEST";

        bzero(&server,sizeof(server));
        server.sin_family=PF_INET;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        server.sin_port=htons(5678);

        sock=socket(PF_INET,SOCK_DGRAM,0);

        connect(sock,(struct sockaddr *)&server,sizeof(server));
        sendto(sock,buf,sizeof(buf),0,0,0);          //不需在指定要送往哪
}
```


UDP client 範例(3/3)

CODE. 10-2

```
#include <sys/socket.h>
#include <netinet/in.h>

int main(void){
        struct sockaddr_in server;
        int sock;
        char buf[]="UDP TEST";

        bzero(&server,sizeof(server));
        server.sin_family=PF_INET;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        server.sin_port=htons(5678);

        sock=socket(PF_INET,SOCK_DGRAM,0);

        connect(sock,(struct sockaddr *)&server,sizeof(server));
        write(sock,buf,sizeof(buf));        //也可以用read, write函式來讀寫
}
```

網路與機器位元組順序轉換函式

```
#include <netinet/in.h>

    uint32_t htonl(uint32_t hostlong);
    uint16_t htons(uint16_t hostshort);
    uint32_t ntohl(uint32_t netlong);
    uint16_t ntohs(uint16_t netshort);
```

- 由於網路(network)位元組順序可能與機器(host)的位元組順序不同，因此在處理兩位元組以上的資料時必須透過這些函式來轉換，才不會有移植性的問題產生
- htonl將一個long(4 bytes)資料，從host轉成network的位元組順序
- ntohl將一個long(4 bytes)資料，從network轉成host的位元組順序
- htons將一個short(2 bytes)資料，從host轉成network的位元組順序
- ntohs將一個short(2 bytes)資料，從network轉成host的位元組順序

位元組資料操作函式

BSD

```
#include <string.h>

    void bzero(void *s, size_t n);
    void bcopy(const void *src, void *dest, size_t n);
    int bcmp(const void *s1, const void *s2, size_t n);
```

System V

```
#include <string.h>

    void *memset(void *s, int c, size_t n);
    void *memcpy(void *dest, const void *src, size_t n);
    int memcmp(const void *s1, const void *s2, size_t n);
```

- 在socket程式設計中，常常會操作某些資料結構，例如位址資訊(struct sockaddr)，因此需要這些位元組資料操作函式
  - bzero函式用來將從s啟始，長度為n的記憶體空間設為0
  - bcopy函式從dest拷貝n的位元組到src所指的空間
  - bcmp比較s1與s2的前n個位元組是否相同
- 在linux中，bzero，bcopy與bcmp三個函式已經被取消(但還是可以用)，請分別用memset，memcpy與memcmp來取代
- 特別注意參數的順序與傳回值所代表的意義，例如bcopy是將第一個參數拷貝到第二個參數，而memcpy是將第二個參數拷貝到第一個參數

網路位址轉換函式

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

    in_addr_t inet_addr(const char *cp);
    char *inet_ntoa(struct in_addr in);
```

- 這頁的函式用來轉換IPv4的網路位址，IPv4的位址形式為a.b.c.d，稱為dotted-decimal format，例如"140.113.23.3"，但在struct addr中的格式卻是一個struct in_addr的格式，稱為binary format。這兩個函式就是在這兩個格式間做轉換
- inet_addr函式將dotted-decimal format轉換成binary format
- inet_ntoa函式將binary format轉換成dotted-decimal format

readv與writev函式在socket上的應用

- 當檔頭(header)與資料(payload)分開時可以使用readv或是writev函式，以增加讀寫效率，參考下列例子
- 這兩個函式都是atomic operation

```
#include <sys/uio.h>

int write_hrd(int fd, char *buf, int nbytes){
        struct hdr_info header;
        struct iovec iov[2];

        iov[0].iov_base=&header;
        iov[0].iov_len=sizeof(header);

        iov[1].iov_base=buf;
        iov[1].iov_len=nbytes;

        writev(fd,&iov[0],2);
}
```

取得socket的名稱

CODE. 10-3

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

    int getsockname(int s, struct sockaddr *name, socklen_t *namelen);
    int getpeername(int s, struct sockaddr *name, socklen_t *namelen);
```

- 可以透過這兩個函式分別取得socket的local address與foreign address
- getsockname函式用來取得socket的local address，取得的資料會透過name所指的空間傳回
- getpeername函式用來取得socket的foreign address，取得的資料會透過name所指的空間傳回

關閉socket

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

    int shutdown(int s, int how);
```

- 與close函式相同，都可以用來關閉socket，但是它可以指定關閉的方向
- 由於socket為雙向的，透過shutdown函式可以分別指定要關閉哪一邊。shutdown函式的第二個參數就是用來指定關閉方向，可以有下列值
  - SHUT_RD: 關閉讀取的方向，關閉後該socket就不能讀
  - SHUT_WR:關閉寫入的方向，關閉後就不能寫入該socket
  - SHUT_RDWR:關閉雙向，就跟close函式相同

Unnamed stream pipes

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

    int socketpair(int d, int type, int protocol, int sv[2]);
```

- socketpair函式跟pipe函式相當類似，但是它是傳回兩個socket descriptor而不是file descriptor。此外socketpair函式所建立的socket descriptor是雙向的，不像pipe是單向的
- 在linux下只支援PF_UNIX的family(也就是Unix domain)，所以只能有下列兩種用法

  int sockfd[2];
  socketpair(PF_UNIX, SOCK_STREAM, 0, sockfd);
  socketpait(PF_UNIX, SOCK_DGRAM, 0, sockfd);
- 用socketpair函式產生的通道可以稱為unnamed stream pipes

Named stream pipes

CODE. 10-4

```
struct sockaddr_un server_addr;
int sock,len;

sock=socket(PF_UNIX, SOCK_STREAM, 0);          //產生socket descriptor

bzero(&server_addr,sizeof(server_addr));       //設定unix domain addr.
server_addr.sun_family=PF_UNIX;
strcpy(server_addr.sun_path,"stream");
len=sizeof(server_addr.sun_family)+strlen(server_addr.sun_path);

unlink("stream");
bind(sock,(struct sockaddr *)&server_addr,len);    //bind到檔案系統上
```

- 上面的程式片段可以產生named stream pipes，也就是在檔案系統中會出現代表該stream pipes的檔案
- 產生named stream pipes的方式為先用socket產生socket descriptor，然後再透過bind函式將該socket descriptor連接到檔案系統上

設定與取得socket的參數(1/2)

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

    int getsockopt(int  s, int level, int optname, void *optval, socklen_t *optlen);
    int setsockopt(int s, int  level,  int  optname,  const  void  *optval, socklen_t optlen);
```

- getsockopt函式用來取得某個socket descriptor的屬性，setsockopt用來設定某個socket descriptor的屬性。
- 第一個參數s代表已開啟所要操作的socket descriptor，第二個參數level代表要設定哪一層的資料(例如是要設定socket code，還是要設定TCP/IP code或是其它)，第三個參數是所要設定或讀取的屬性名稱，第四個參數為所要設定或回傳的資料內容，第五個參數為第四個參數的長度
- 不同level，會有不同的屬性可以設定，請看下一頁的介紹

設定與取得socket的參數(2/2)

Level	optname	description
SOL_IP	IP_OPTIONS	設定或取得IP header中的options
	IP_TTL	設定或取得Time to live
SOL_SOCKET	SO_KEEPALIVE	保持連線狀態
	SO_OOBINLINE	將OOB的資料當成in-line資料來送
	SO_REUSEADDR	允許重複使用local address
	SO_BROADCAST	允許接收或傳送broadcast的資料
	SO_SNDBUF	設定或取得sending buffer的大小
	SO_RCVBUF	設定或取得receiving buffer的大小
	SO_PASSCRED	允許接收ancillary訊息
SOL_TCP	TCP_INFO	取得TCP的資訊

fcntl與ioctl函式在socket上的應用

```
#include <unistd.h>
#include <fcntl.h>

    int fcntl(int fd, int cmd, long arg);
```

```
#include <sys/ioctl.h>

    int ioctl(int d, int request, ...);
```

- fcntl與ioctl函式都可以用來設定socket的狀態，但是前者要一次設定整個狀態，後者可以分開來設定。
- fcntl函式中的cmd可以用F_SETFL與F_GETFL，前者用來設定socket狀態，後者用來取得目前socket狀態。對於socket來說，只有下列兩種狀態有意義

fcntl的arg參數	ioctl的request參數	意義
FASYNC	FIOASYNC	將socket descriptor設為async.
FNDELAY	FIONBIO	將socket descriptor設為nonblocking

sendmsg與recvmsg函式

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

    int sendmsg(int s, const struct msghdr *msg, int flags);
    int recvmsg(int s, struct msghdr *msg, int flags);
```

```
struct msghdr {
        void  * msg_name;       /* optional address */
        socklen_t  msg_namelen; /* size of address */
        struct iovec  * msg_iov;/* scatter/gather array */
        size_t  msg_iovlen;     /* # elements in msg_iov */
        void  * msg_control;    /* ancillary data */
        socklen_t  msg_controllen; /* ancillary data buffer len */
        int  msg_flags;         /* flags on received message */
};
```

- sendmsg與recvmsg是所有傳送接收函式中最通用(generic)的兩個函式，也就是所有讀寫動作都可以透過它來完成
- 其中的flags參數跟sendto與recvfrom相同
- 可以透過struct msghdr中的msg_control與msg_controllen兩個member來傳送接收ancillary訊息(man 7 unix與man 3 cmsg)

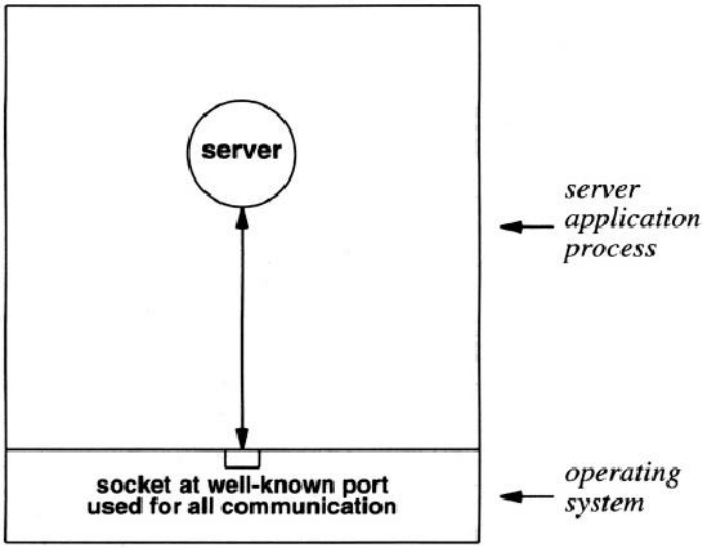
select函式在socket上的應用

CODE. 10-5

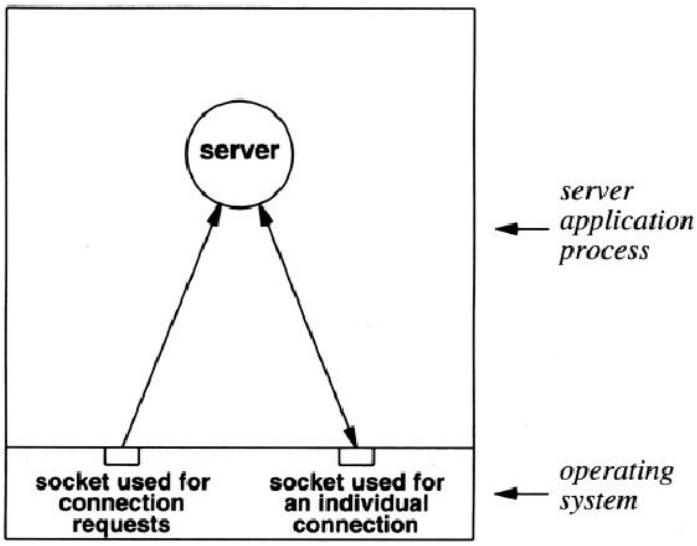
event	Poll flag	發生時機
可讀	POLLIN	新資料當達時
可讀	POLLIN	連線完成時(server端)
可讀	POLLHUP	另一端切斷連線時
可讀	POLLHUP	斷線時
可寫	POLLOUT	當sending buffer足夠大時
可寫	POLLOUT	當建立連線完成時(client端)
可讀/可寫	POLLERR	非同步發生錯誤時
可讀/可寫	POLLHUP	令一端關閉某個方向的連線時
Exception	POLLPRI	收到OOB資料時

- 可以將socket設定成nonblocking，再由發生的evnet來判斷該做什麼事。
- 當socket為nonblocking時，connect函式會馬上返回，其errno會設為EINPROGRESS，當連線完成後會產生一個可寫的event，此時必須用getsockopt來讀取SO_ERROR的屬性來判斷連線是否成功

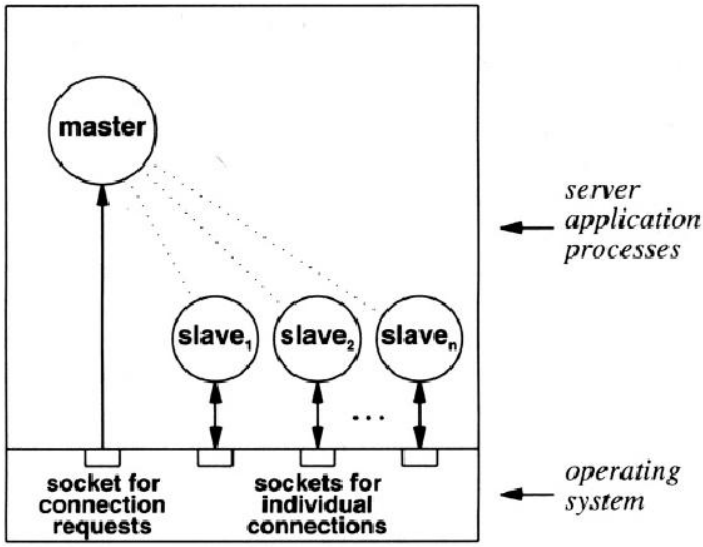
Iterative, connectionless server



Iterative, connection-oriented server



Concurrent, connection-oriented server



Single-process, connection-oriented server