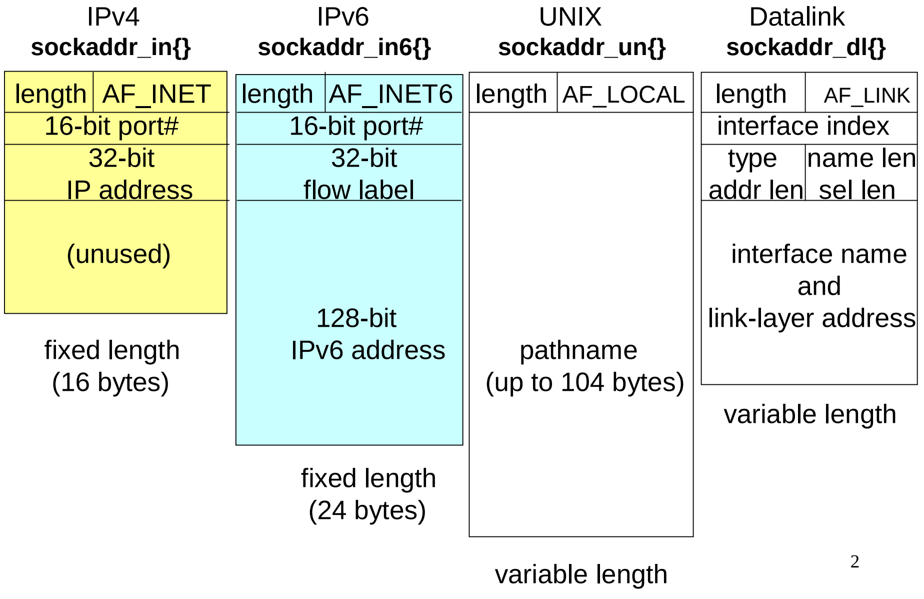
Sockets Introduction

- Socket address structures
- Value-result arguments
- Byte ordering and manipulation functions
- Address conversion functions: *inet_aton*, *inet_addr*, *inet_ntoa*, *inet_pton*, *inet_ntop*, *sock_ntop*
- Stream socket I/O functions: *readn*, *writen*, *readline*
- File descriptor testing function: isfdtype

1

Some materials in these slides are taken from Prof. Ying-Dar Lin with his permission of usage

Comparison of Various Socket Address Structures



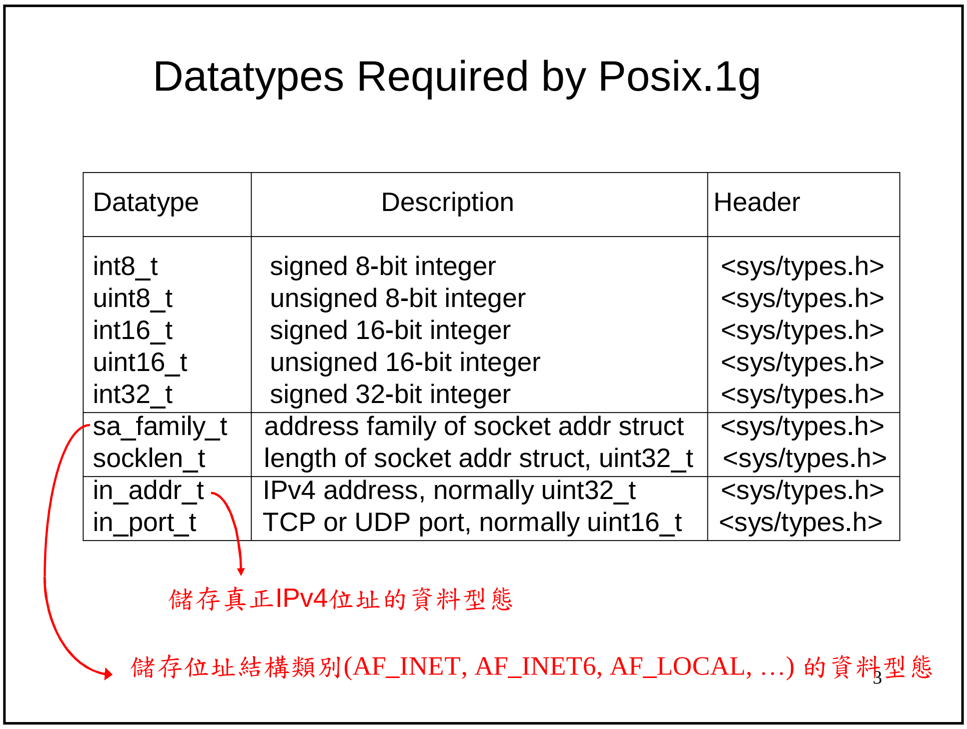
IPv4
sockaddr_in{}
length | AF_INET
16-bit port#
32-bit IP address
(unused)
fixed length (16 bytes)

IPv6
sockaddr_in6{}
length | AF_INET6
16-bit port#
32-bit flow label
128-bit IPv6 address
fixed length (24 bytes)

UNIX
sockaddr_un{}
length | AF_LOCAL
pathname (up to 104 bytes)
variable length

Datalink
sockaddr_dl{}
length | AF_LINK
interface index
type | name len
addr len | sel len
interface name and link-layer address
variable length

2



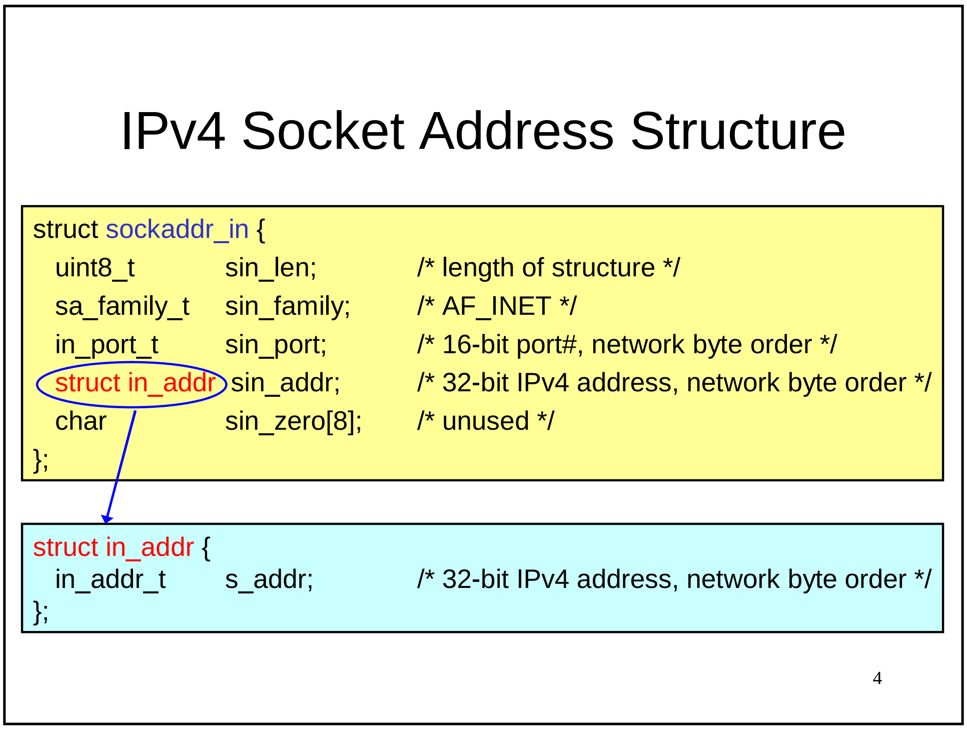
Datatypes Required by Posix.1g

Datatype	Description	Header
int8_t uint8_t int16_t uint16_t int32_t	signed 8-bit integer unsigned 8-bit integer signed 16-bit integer unsigned 16-bit integer signed 32-bit integer	<sys/types.h> <sys/types.h> <sys/types.h> <sys/types.h> <sys/types.h>
sa_family_t socklen_t	address family of socket addr struct length of socket addr struct, uint32_t	<sys/types.h> <sys/types.h>
in_addr_t in_port_t	IPv4 address, normally uint32_t TCP or UDP port, normally uint16_t	<sys/types.h> <sys/types.h>

儲存真正IPv4位址的資料型態

儲存位址結構類別(AF_INET, AF_INET6, AF_LOCAL, …) 的資料型態

3



IPv4 Socket Address Structure

```
struct sockaddr_in {
   uint8_t          sin_len;          /* length of structure */
   sa_family_t      sin_family;       /* AF_INET */
   in_port_t        sin_port;         /* 16-bit port#, network byte order */
   struct in_addr   sin_addr;         /* 32-bit IPv4 address, network byte order */
   char             sin_zero[8];      /* unused */
};
```

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

4

IPv6 Socket Address Structure

```
struct sockaddr_in6 {
  uint8_t         sin6_len;        /* length of this struct [24] */
  sa_family_t     sin6_family;     /* AF_INET6 */
  in_port_t       sin6_port;       /* port#, network byte order */
  uint32_t        sin6_flowinfo;   /* flow label and priority */
  struct in6_addr sin6_addr;       /* IPv6 address, network byte order */
};
```

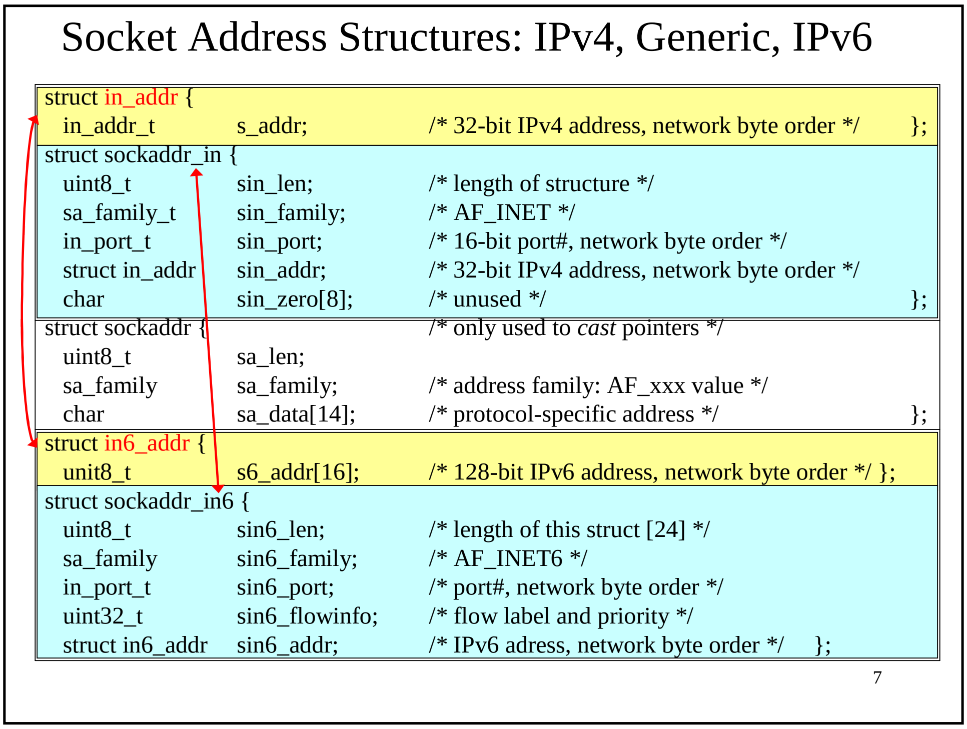
```
struct in6_addr {
  unit8_t         s6_addr[16];     /* 128-bit IPv6 address, network byte order */
};
```

5

Generic (泛用) Socket Address Structure

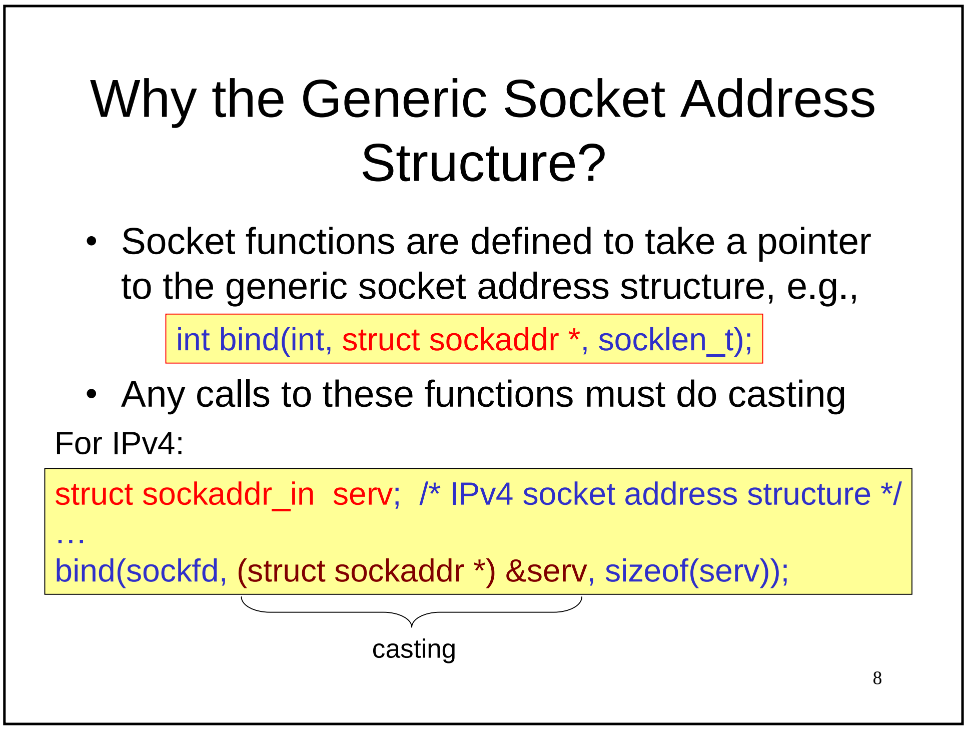
```
struct sockaddr {                  /* only used to cast pointers */
  uint8_t      sa_len;
  sa_family_t  sa_family;          /* address family: AF_xxx value */
  char         sa_data[14];        /* protocol-specific address */
};
```

6



Socket Address Structures: IPv4, Generic, IPv6

```
struct in_addr {
   in_addr_t          s_addr;          /* 32-bit IPv4 address, network byte order */       };
struct sockaddr_in {
   uint8_t            sin_len;         /* length of structure */
   sa_family_t        sin_family;      /* AF_INET */
   in_port_t          sin_port;        /* 16-bit port#, network byte order */
   struct in_addr     sin_addr;        /* 32-bit IPv4 address, network byte order */
   char               sin_zero[8];     /* unused */                                        };
struct sockaddr {                      /* only used to cast pointers */
   uint8_t            sa_len;
   sa_family          sa_family;       /* address family: AF_xxx value */
   char               sa_data[14];     /* protocol-specific address */                     };
struct in6_addr {
   unit8_t            s6_addr[16];     /* 128-bit IPv6 address, network byte order */ };
struct sockaddr_in6 {
   uint8_t            sin6_len;        /* length of this struct [24] */
   sa_family          sin6_family;     /* AF_INET6 */
   in_port_t          sin6_port;       /* port#, network byte order */
   uint32_t           sin6_flowinfo;   /* flow label and priority */
   struct in6_addr    sin6_addr;       /* IPv6 adress, network byte order */     };
```



Why the Generic Socket Address Structure?

- Socket functions are defined to take a pointer to the generic socket address structure, e.g.,

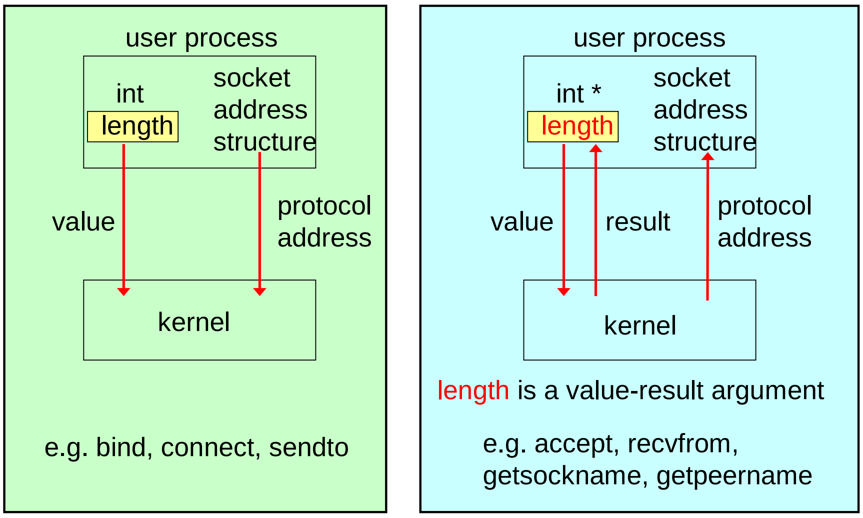
```
int bind(int, struct sockaddr *, socklen_t);
```

- Any calls to these functions must do casting

For IPv4:

$$\texttt{struct sockaddr\_in serv; /* IPv4 socket address structure */}$$
$$\texttt{...}$$
$$\texttt{bind(sockfd, }\underbrace{\texttt{(struct sockaddr *) \&serv}}_{\text{casting}}\texttt{, sizeof(serv));}$$

Value-Result Argument



user process

int

length

socket address structure

value

protocol address

kernel

e.g. bind, connect, sendto

user process

int *

length

socket address structure

value

result

protocol address

kernel

length is a value-result argument

e.g. accept, recvfrom, getsockname, getpeername

應用程式只需將資料傳給kernal　　kernel亦需將資料回傳給應用程式

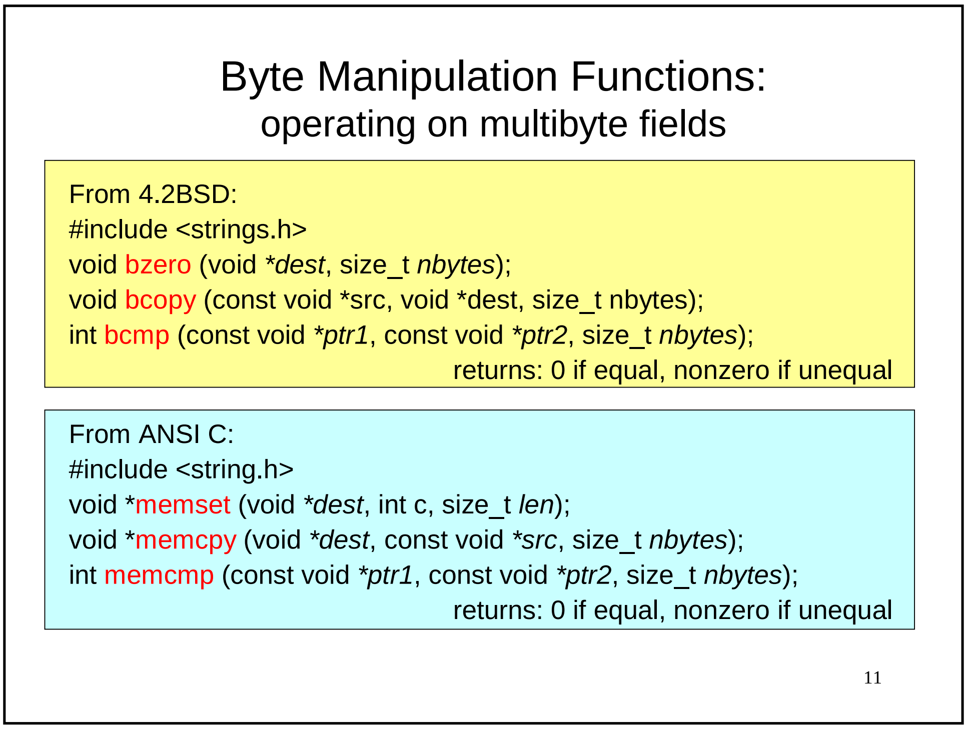
Byte Ordering Functions:

converting between the host byte order to the network byte order

	address A+1	address A
little-endian byte order:	high-order byte	low-order byte
big-endian byte order:	high-order byte	low-order byte
(for a 16-bit integer)	address A	address A+1

Some machines use the little-endian host byte order while the others use the big-endian. The Internet protocols use the *big-endian* network byte order. Hence, conversion functions should be added in all cases.

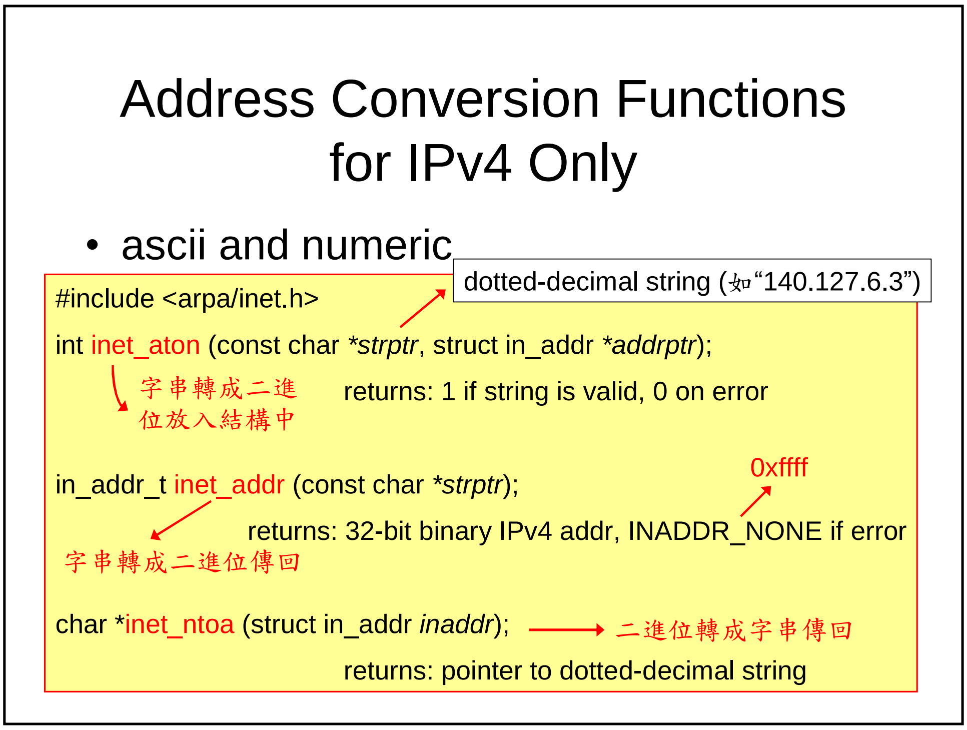
```
#include <netinet/in.h>
uint16_t htons(uint16_t host16bitvalue); returns: value in network byte order
uint32_t htonl(uint32_t host32bitvalue); returns: value in network byte order
uint16_t ntohs(uint16_t net16bitvalue); returns: value in host byte order
uint32_t ntohl(uint32_t net32bitvalue);  returns: value in host byte order
```



Byte Manipulation Functions:
operating on multibyte fields

```
From 4.2BSD:
#include <strings.h>
void bzero (void *dest, size_t nbytes);
void bcopy (const void *src, void *dest, size_t nbytes);
int bcmp (const void *ptr1, const void *ptr2, size_t nbytes);
                              returns: 0 if equal, nonzero if unequal
```

```
From ANSI C:
#include <string.h>
void *memset (void *dest, int c, size_t len);
void *memcpy (void *dest, const void *src, size_t nbytes);
int memcmp (const void *ptr1, const void *ptr2, size_t nbytes);
                              returns: 0 if equal, nonzero if unequal
```



Address Conversion Functions for IPv4 Only

- ascii and numeric

dotted-decimal string (如“140.127.6.3”)

```
#include <arpa/inet.h>
int inet_aton (const char *strptr, struct in_addr *addrptr);
            returns: 1 if string is valid, 0 on error
```

字串轉成二進位放入結構中

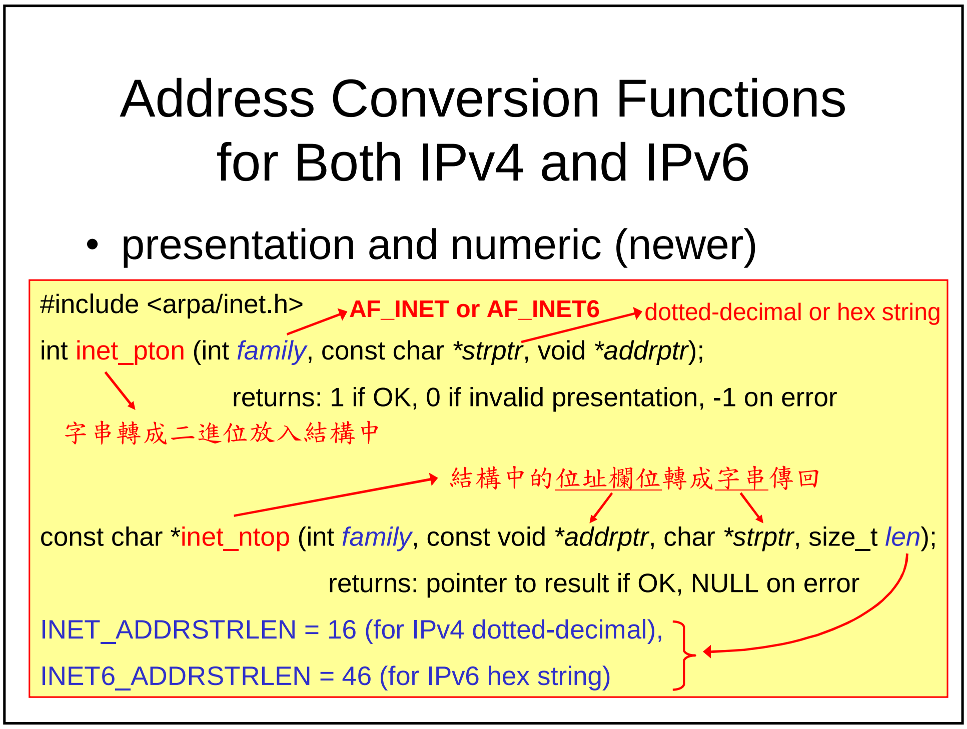
```
in_addr_t inet_addr (const char *strptr);
       returns: 32-bit binary IPv4 addr, INADDR_NONE if error
```

0xffff

字串轉成二進位傳回

```
char *inet_ntoa (struct in_addr inaddr);
            returns: pointer to dotted-decimal string
```

二進位轉成字串傳回



Address Conversion Functions for Both IPv4 and IPv6

- presentation and numeric (newer)

```
#include <arpa/inet.h>
int inet_pton (int family, const char *strptr, void *addrptr);
                returns: 1 if OK, 0 if invalid presentation, -1 on error
```

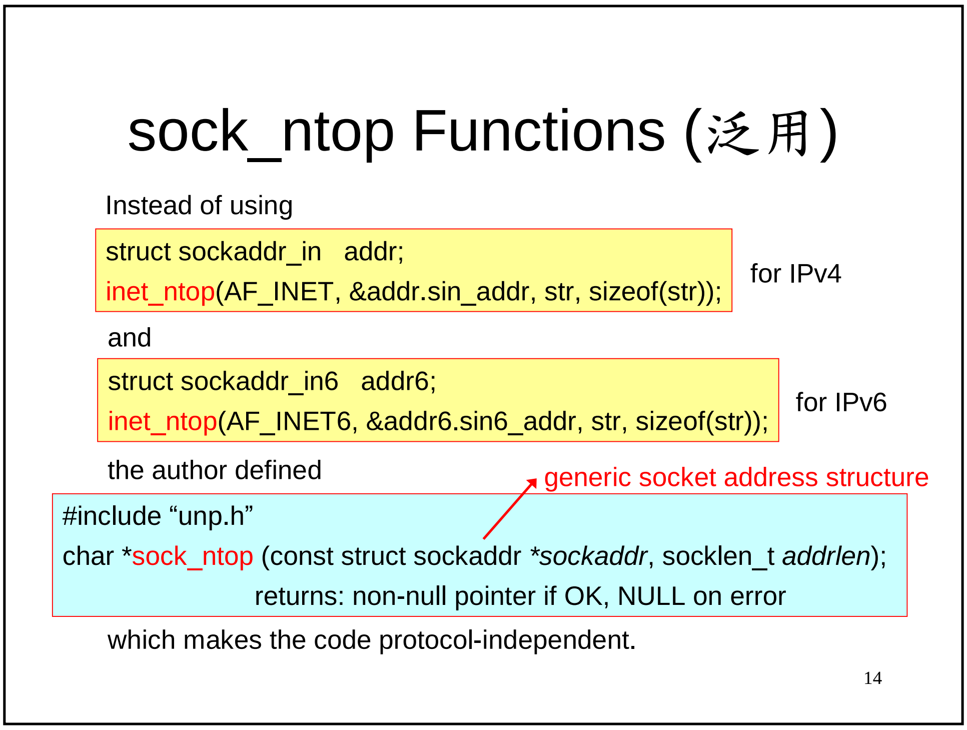
AF_INET or AF_INET6

dotted-decimal or hex string

字串轉成二進位放入結構中

結構中的位址欄位轉成字串傳回

```
const char *inet_ntop (int family, const void *addrptr, char *strptr, size_t len);
                returns: pointer to result if OK, NULL on error
INET_ADDRSTRLEN = 16 (for IPv4 dotted-decimal),
INET6_ADDRSTRLEN = 46 (for IPv6 hex string)
```



sock_ntop Functions (泛用)

Instead of using

```
struct sockaddr_in   addr;
inet_ntop(AF_INET, &addr.sin_addr, str, sizeof(str));
```

for IPv4

and

```
struct sockaddr_in6   addr6;
inet_ntop(AF_INET6, &addr6.sin6_addr, str, sizeof(str));
```

for IPv6

the author defined

generic socket address structure

```
#include “unp.h”
char *sock_ntop (const struct sockaddr *sockaddr, socklen_t addrlen);
                returns: non-null pointer if OK, NULL on error
```

which makes the code protocol-independent.

14

Other sock_ Functions

```
#include “unp.h”
int sock_bind_wild (int sockfd, int family);

sock_cmp_addr, sock_cmp_port, sock_get_port,
sock_ntop_host, sock_set_addr, sock_set_port,
sock_set_wild, etc.
```

Short Count Problem with `read` or `write`

- A **`read`** or **`write`** on a stream socket (TCP) might input or output fewer bytes than requested => short count
  - Because buffer limit might be reached for the socket in the kernel
  - The caller needs to invoke the **`read`** or **`write`** function again for the remaining bytes

readn, writen, and readline

- defined by the author to handle short count (保證讀寫的byte數)

```
#include “unp.h”
ssize_t readn (int filedes, void *buff, size_t nbytes);
                         returns: #bytes read, -1 on error
ssize_t writen (int filedes, const void *buff, size_t nbytes);
                         returns: #bytes written, -1 on error
ssize_t readline (int filedes, void *buff, size_t maxlen);
                         returns: #bytes read, -1 on error
```

17

File Descriptor Testing Function:
testing a descriptor of a specific type

使用Unix的open函數開啟檔案或其它裝置亦會return file descriptor

to test a socket: *fdtype* is S_IFSOCK

```
#include <sys/stat.h>
int isfdtype (int sockfd, int fdtype);
   returns: 1 if descriptor of specified type, 0 if not, -1 on error
```

isfdtype is implemented by calling fstat and testing the returned st_mode value using the S_IFMT macro.

18