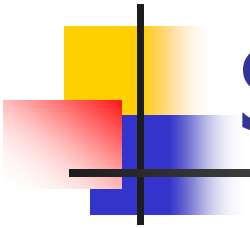
Computer Network Programming

Socket Address Structures

Dr. Sam Hsu
Computer Science & Engineering
Florida Atlantic University



Socket Address Structures

- IPv4 Socket Address Structures
- IPv6 Socket Address Structures
- Value-result Arguments
- Network Byte Order
- Byte Manipulation Functions
- Address Conversion Functions

Generic IPv4 Socket Address Structure

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

struct  sockaddr {
      uint8_t   sa_len;
      sa_family_t   sa_family;    /* address family; AF_XXX */
      char   sa_data[14];          /* protocol–specific address */
 }
```

- Note: This generic socket structure is primarily for declaring variables. “cast” is needed in the actual use of a socket address structure.

IPv4 Socket Address Structure

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

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

struct   sockaddr_in {
    uint8_t   sin_len;          /* length of this struct (16 bytes) */
    sa_family_t   sin_family    /* AF_INET */
    in_port_t   sin_port;       /* 16-bit port number in NW byte order */
    struct  in_addr  sin_addr;  /* 32-bit IPv4 addr in NW byte order */
    char   sin_zero[8];         /* unused, set to zero */
}
```

Generic IPv6 Socket Address Structure

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

struct   sockaddr_storage {
      uint8_t   ss_len;              /* impl.-dependent struct length */
      sa_family_t   ss_family;    /* address family; AF_XXX */
      /* implementation-dependent elements to provide:
         a) alignment sufficient to fulfill the alignment requirements of all
            socket address types that the system supports.
         b) enough storage to hold any type of socket address that the
            system supports.
      */
  }
```

- Note: This socket address structure is used primarily for “cast”.

IPv6 Socket Address Structure

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

struct   in6_addr {
    uint8_t   s6_addr;          /* 128-bit IPv6 address */
}                               /* in network byte order */

struct   sockaddr_in6 {
    uint8_t  sin6_len;          /* length of this struct (28 bytes) */
    sa_family_t  sin6_family    /* AF_INET6 */
    in_port_t  sin6_port;       /* 16-bit port number in NW byte order */
    uint32_t   sin6_flowinfor;  /* flow label, priority */
    struct  in6_addr  sin6_addr; /* 128-bit IPv6 addr in NW byte order */
    uint32_t   sin6_scope_id;   /* set of interfaces for a scope */
}
```

Various Socket Address Structures

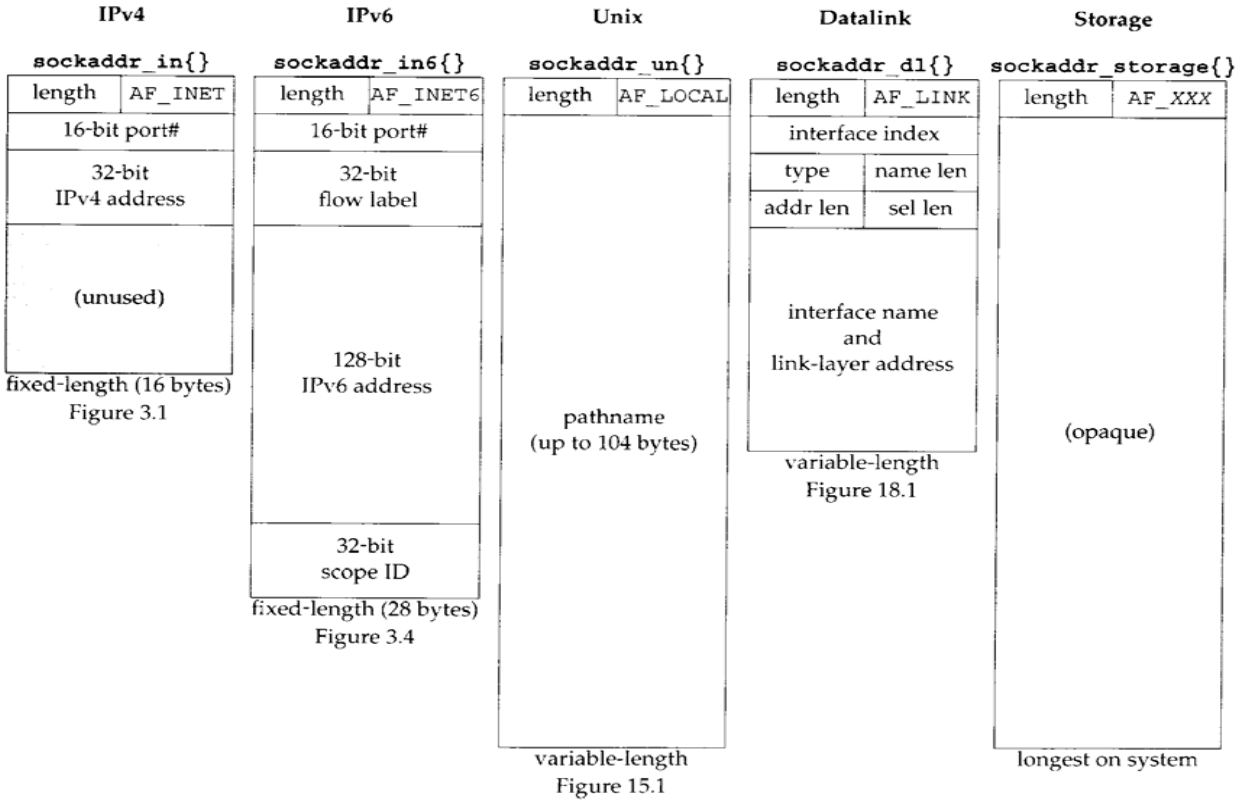


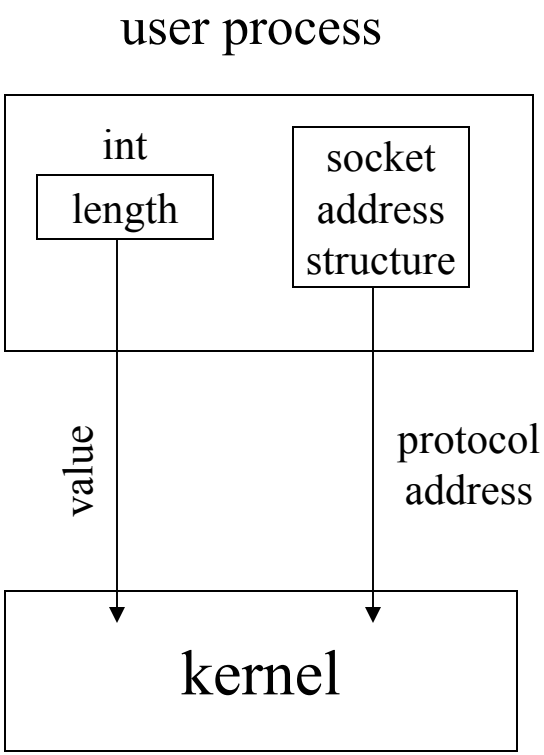
Figure 3.6 Comparison of various socket address structures.

Relevant POSIX Datatypes*

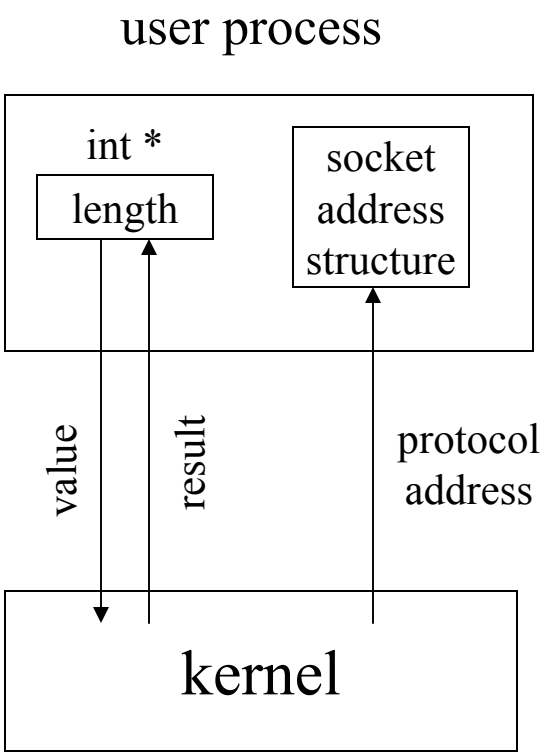
Datatype	Description	Header
int8_t	Signed 8-bit integer	<sys/types.h>
uint8_t	Unsigned 8-bit integer	<sys/types.h>
int16_t	Signed 16-bit integer	<sys/types.h>
uint16_t	Unsigned 16-bit integer	<sys/types.h>
int32_t	Signed 32-bit integer	<sys/types.h>
uint32_t	Unsigned 32-bit integer	<sys/types.h>
sa_family_t	Address family of socket address structure	<sys/sockets.h>
socklen_t	Length of socket address structure, normally uint32_t	<sys/sockets.h>
in_addr_t	IPv4 address, normally uint32_t	<netinet/in.h>
in_port_t	TCP or UDP port, normally uint16_t	<netinet/in.h>

Ref: *UNIX Networking Programming*, vol 1, ed 3, Stevens et al., 2004, Addison Wesley, p. 69.

Value-Result Arguments



e.g., bind, connect, sendto

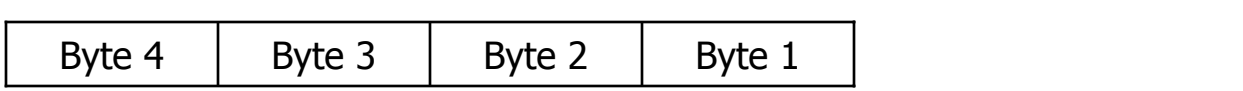


e.g., accept, recvfrom, getsockname, getpeername

Ref: *UNP*, Stevens et al., vol 1, ed 3, 2004, AW, pp. 75-76

Network Byte Order

- Two different hardware implementations of storing integers on a host:
  - Big-endian byte order



  - Little-endian byte order

- Conversions between host byte order and network byte order may be needed.
- Network byte order uses big-endian ordering.

Byte Manipulation Functions (1/2)

- For operating on multi-byte fields, without interpreting the data, and without assuming the data being null-terminated.
- 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, nozero if unequal
```

Byte Manipulation Functions (2/2)

- From ANSI C:

```
#include <string.h>

void *memset(void *dest,  int c,  size_t  len);

void memcpy(void *dest,  void  const void *src,  size_t nbytes);

int memcmp(const void *ptr1,  const void *ptr2,  size_t nbytes);

                                Returns: 0 if equal, < 0 or > 0 if unequal
```

  - Note: bcopy() correctly handles overlapping fields, whereas the behavior of memcpy() is undefined if overlapping happens.

Address Conversion Functions (1/2)

- For IPv4 and IPv6: Between **p**resentation (ASCII string) and its **n**etwork byte ordered binary value.

```
#include <arpa/inet.h>

void inet_pton(int family, const char *strptr,  void *addrptr);

        Returns: 1 if OK, 0 if input not valid, -1 on error

const char *inet_ntop(int family, const void *addrptr2,
                                  char *strptr, size_t len);

                Returns: pointer to result if OK, NULL on error
```

Address Conversion Functions (2/2)

- For IPv4 only: Between a dotted-decimal ASCII string and its network byte ordered binary value.

```
#include <arpa/inet.h>

void inet_aton(const char *strptr,  struct in_addr *addrptr);

                    Returns: 1 if string was valid, 0 on error

In_addr_t  inet_addr(const char *strptr);

     Returns: 32-bit IPv4 address, INADDR_NONE if error

char *inet_ntoa(struct in_addr inaddr);

                         Returns: pointer to dotted-decimal string
```

Address Conversion Summary

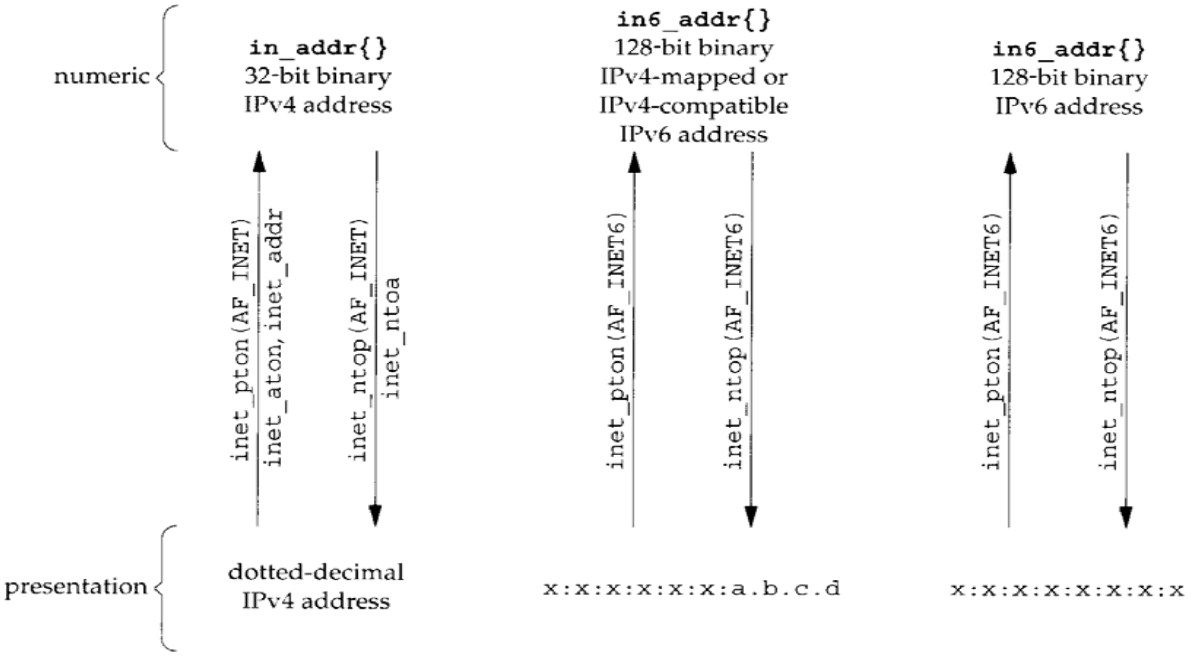
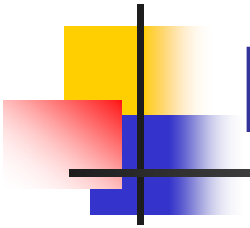


Figure 3.11 Summary of address conversion functions.



Reading Assignment

- Read Chapter 3.