Byte Ordering and Byte Manipulation Functions

Byte Ordering Functions

- Unfortunately, not all computers store the bytes that comprise a multibyte value in the same order.
- Consider a 16-bit integer that is made up of 2 bytes. There are two ways to store this value.
  - **Little Endian** − In this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next address (A + 1).
  - **Big Endian** − In this scheme, high-order byte is stored on the starting address (A) and low-order byte is stored on the next address (A + 1).

Byte Ordering Functions

- To allow machines with different byte order conventions communicate with each other, the Internet protocols specify a canonical byte order convention for data transmitted over the network. This is known as Network Byte Order.
- While establishing an Internet socket connection, you must make sure that the data in the sin_port and sin_addr members of the sockaddr_in structure are represented in Network Byte Order.

Byte Ordering Functions

- Functions for converting data between a host's internal representation and Network Byte Order are as follows −

- **unsigned short htons(unsigned short hostshort)** − This function converts 16-bit (2-byte) quantities from host byte order to network byte order.
- **unsigned long htonl(unsigned long hostlong)** − This function converts 32-bit (4-byte) quantities from host byte order to network byte order.
- **unsigned short ntohs(unsigned short netshort)** − This function converts 16-bit (2-byte) quantities from network byte order to host byte order.
- **unsigned long ntohl(unsigned long netlong)** − This function converts 32-bit quantities from network byte order to host byte order.

Byte Ordering Functions

- These functions are macros and result in the insertion of conversion source code into the calling program.
  - On little-endian machines, the code will change the values around to network byte order.
  - On big-endian machines, no code is inserted since none is needed; the functions are defined as null.

Program to Determine Host Byte Order

```c
#include <stdio.h>
 int main(int argc, char **argv)
 {
 union {
short s;
char c[sizeof(short)];
 }un;

un.s = 0x0102;  //hexadecimal
if (sizeof(short) == 2)
{
   if (un.c[0] == 1 && un.c[1] == 2)
      printf("big-endian\n");
   else if (un.c[0] == 2 && un.c[1] == 1)
    printf("little-endian\n");
   else printf("unknown\n");
}
```

```c
else {
printf("sizeof(short) = %d\n",
sizeof(short));
 }
exit(0);
 }
```

- The elements of a union occupy a common "piece" of memory. So un.s and un.c[] refer to same memory

- s= 0x0102, therefore
- If c={0x02, 0x01}→ Little Endian
- If c={0x01, 0x02}→ Big Endian

Byte Manipulation Functions

- Unix provides various function calls to help you manipulate IP addresses.
- These functions convert Internet addresses between ASCII strings (what humans prefer to use) and network byte ordered binary values (values that are stored in socket address structures).
- The following three function calls are used for IPv4 addressing −
  - **int inet_aton(const char *strptr, struct in_addr *addrptr)**
  - **in_addr_t inet_addr(const char *strptr)**
  - **char *inet_ntoa(struct in_addr inaddr)**

Byte Manipulation Functions

int inet_aton(const char *strptr, struct in_addr *addrptr)

- This function call converts the specified string in the Internet standard dot notation to a network address, and stores the address in the structure provided.
- The converted address will be in Network Byte Order (bytes ordered from left to right). It returns 1 if the string was valid and 0 on error.
- example –

```
#include <arpa/inet.h>
(...)
int retval;
struct in_addr addrptr;
memset(&addrptr, '\0', sizeof(addrptr));
retval = inet_aton("68.178.157.132", &addrptr);
(...)
```

Byte Manipulation Functions

in_addr_t inet_addr(const char *strptr)

- This function call converts the specified string in the Internet standard dot notation to an integer value suitable for use as an Internet address.
- The converted address will be in Network Byte Order (bytes ordered from left to right). It returns a 32-bit binary network byte ordered IPv4 address and INADDR_NONE on error.
- example –

```
#include <arpa/inet.h>
(...)
struct sockaddr_in dest;
memset(&dest, '\0', sizeof(dest));
dest.sin_addr.s_addr
            =inet_addr("69.172.156.131");
(...)
```

Byte Manipulation Functions

char *inet_ntoa(struct in_addr inaddr)

- This function call converts the specified Internet host address to a string in the Internet standard dot notation.
- Example

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

char *ip;
ip = inet_ntoa(dest.sin_addr);
 printf("IP Address is: %s\n",ip);

(...)
```

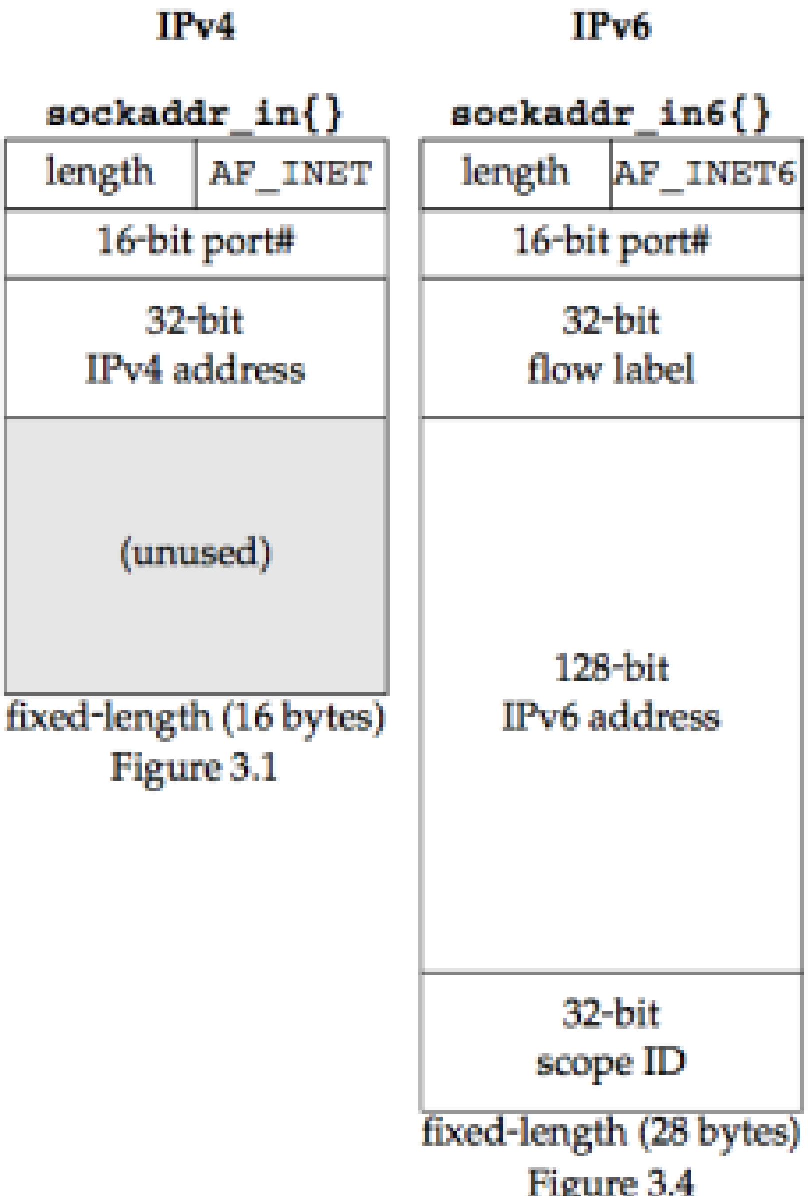
IPv6 Address Structure

```
struct in6_addr {
  uint8_t  s6_addr[16];            /* 128-bit IPv6 address */
                                   /* network byte ordered */
};

#define SIN6_LEN      /* required for compile-time tests */

struct sockaddr_in6 {
  uint8_t         sin6_len;      /* length of this struct (28) */
  sa_family_t     sin6_family;   /* AF_INET6 */
  in_port_t       sin6_port;     /* transport layer port# */
                                 /* network byte ordered */
  uint32_t        sin6_flowinfo; /* flow information, undefined */
  struct in6_addr sin6_addr;     /* IPv6 address */
                                 /* network byte ordered */
  uint32_t        sin6_scope_id; /* set of interfaces for a scope */
};
```

IPv6 Address Structure

- The SIN6_LEN constant must be defined if the system supports the length member for socket address structures.
- The IPv6 family is AF_INET6, whereas the IPv4 family is AF_INET.
- The members in this structure are ordered so that if the sockaddr_in6 structure is 64-bit aligned, so is the 128-bit sin6_addr member. On some 64-bit processors, data accesses of 64-bit values are optimized if stored on a 64-bit boundary.
- The sin6_flowinfo member is divided into two fields:
  - The low-order 20 bits are the flow label
  - The high-order 12 bits are reserved
- The use of the flow label field is still a research topic.

IPv6 Address Structure



Compare IPv4 and IPv6 Address Structures

IPV6 Address Structure

Example:

```
int listen_sock_fd;
struct sockaddr_in6 server_addr;  // IPV6 address

/* Create socket for listening (client requests) */
        listen_sock_fd = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

/* Assign Values to  IPV6 address*/

        server_addr.sin6_family = AF_INET6;
        server_addr.sin6_addr = in6addr_any;  //#
        server_addr.sin6_port = htons(SERVER_PORT);

        /* Bind address and socket together */
bind(listen_sock_fd, (struct sockaddr*)&server_addr, sizeof(server_addr));

/*rest all code is same as of IPv4*/
```

\# instead following may be used to assign loopback address only

```
inet_pton(AF_INET6, "::1", &server_addr.sin6_addr);
```