

Networking: IPv4 - AF_INET



Instructor: Linyi Li
Slides adapted from Dr. B. Fraser

Topics

- How can we use sockets **on a network** (**AF_INET**)?
- How do **different computer architectures** agree on **a data format**?

AF_INET

Data Structure

AF_INET and AF_INET6

.IP Addresses

-IPv4: AF_INET uses 4 bytes for IP addresses:

e.g., 192.168.7.2

-IPv6: AF_INET6 uses 16 bytes for IP addresses.

e.g., 2F10:C203:A135:DC3F:35:6F2:1:F603

-More info:

man 7 ip

man 7 ipv6

-We'll focus on AF_INET.

.AF_INET addresses use
struct sockaddr_in

-.. "in" means Internet

```
struct in_addr {  
    in_addr_t s_addr;  
};  
  
struct sockaddr_in {  
    sa_family_t  sin_family;  
    in_port_t    sin_port;  
    struct in_addr sin_addr;  
    unsigned char __pad[X];  
}
```

sockaddr_in Field: sin_addr

.Binary Format

-Humans write IPv4 addresses as “192.168.7.1”

-.. Computer represents as 4-byte value

.Convert address

-inet_pton() “192.168.0.1” --> binary

.. presentation to network

-inet_ntop() binary --> “192.168.0.1”

.. network to presentation

-These handle both IPv4 and IPv6

.Presentation String Lengths

-Max string length defined in `<netinet/in.h>`

-IPv4: `INET_ADDRSTRLEN`

IPv6: `INET6_ADDRSTRLEN`

```
struct in_addr {
    in_addr_t s_addr;
};

struct sockaddr_in {
    sa_family_t sin_family;
    in_port_t sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[X];
}
```

sin_addr - Two special addresses

.loopback address: 127.0.0.1

```
sin_addr.s_addr = INADDR_LOOPBACK;
```

-Local communication, similar to UNIX domain sockets.

-.. Data sent/received locally

i.e., nothing onto network.

.Wildcard address

```
sin_addr.s_addr = INADDR_ANY;
```

-A machine can have multiple network cards,

-e.g., wireless & wired (Ethernet) card:
each with an IP address.

-.. bind() to socket with wildcard
address listens to any address.

```
struct in_addr {  
    in_addr_t s_addr;  
};  
  
struct sockaddr_in {  
    sa_family_t  sin_family;  
    in_port_t    sin_port;  
    struct in_addr sin_addr;  
    unsigned char __pad[X];  
}
```

sockaddr_in Field: sin_port

- **bind()** needs IP address and port

- .. Port number identifies a specific socket on the machine.

- Some ports are well known, such as:

- **SSH**: 22

- **HTTP**: 80

- Clients know to look at these ports.

- .. **Ephemeral Port**

- If we don't **bind()** our socket to a specific port, then TCP or UDP picks an unused "random" port.

```
struct in_addr {
    in_addr_t s_addr;
};

struct sockaddr_in {
    sa_family_t  sin_family;
    in_port_t    sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[X];
}
```

Byte Order & Hosts

Byte Order

- Different computer architectures use different byte orders:

- e.g., consider the number $12345 = 0x3039$

- Little Endian:

- .. Store the little part (least-significant byte=LSB) first (at lower address).

- Big Endian:

- Store the big part (MSB) first (at lower address).

- Network Byte Order

- Different computers communicate, so network must have established byte order.

- .. Network Byte Order is Big Endian

- E.g., port number and the IP address are multi byte, so they are sent MSB first.

Network Byte Order

- Byte-order translation functions

man byteorder

```
#include <arpa/inet.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);
```

–.. “Host To Network Long”, etc.

- Only for multi-byte values

–single byte data (chars) just sent one at a time.

Host Names

- .Can use a host name instead of an IP address

- Host name is the computer name.

- `getaddrinfo()`

Converts host name (string) to set of all possible options (structs containing an IP and a port number)

- `getnameinfo()`

performs reverse---IP to host name.

Activity

- Create two programs, server and client.

- Implement the socket sequence (TCP stream) using **AF_INET**

- Send messages from the client and print them out from the server.

- Use **port 8000** on the server.

- Recall

- **AF_INET** uses **sockaddr_in**

```
struct in_addr {
    in_addr_t s_addr;
};

struct sockaddr_in {
    sa_family_t  sin_family;
    in_port_t    sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[X];
}
```

recv() and send()

• `ssize_t recv(int sockfd, void *buf, size_t len, int flags);`

– Similar to `read()` but socket specific.

– Provides more control, e.g.:

• `MSG_DONTWAIT`: Non-blocking

• `MSG_PEEK`: read but don't remove

• `ssize_t send(int sockfd, const void *buf, size_t len, int flags);`

– Similar to `write()` but socket specific.

– Provides more control

e.g.: `MSG_DONTWAIT`: Non-blocking

Summary

• Use AF_INET for IPv4

```
struct in_addr {
    in_addr_t s_addr;
};

struct sockaddr_in {
    sa_family_t  sin_family;
    in_port_t    sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[X];
}
```

• Network Byte Order

– **Big-Endian:** Biggest byte is first.