Network Programming for Windows 03:

Internet Protocol

jintaeks@dongseo.ac.kr
Division of Digital Contents, DSU
May 2018

Outline

- ✓ IPv4
- ✓ IPv6
- ✓ Address and Name Resolution
- ✓ ~~Writing IP Version-independent Programs~~

- ✓ IPv4 is commonly known as the **network protocol** that the Internet uses.
  - The background,
  - addressing scheme,
  - name resolution,
  - and Winsock specifics for both IPv4 and ~~IPv6~~.

IPv4

- ✓ IPv4 was developed by the U.S. Department of Defense's Advanced Research Project Agency (ARPA).
- ✓ This research eventually led to IPv4 as well as TCP.

Addressing

- ✓ In IPv4, computers are assigned an address that is represented as a 32-bit number, formally known as an **IPv4 address**.
- ✓ IPv4 addresses are divided into **classes** that describe the portion of the address assigned to the network and the portion assigned to endpoints.

Class	Network Portion	First Number	Number of Endpoints	Default Subnet Mask
A	8 bits	0–127	16,777,216	255.0.0.0
B	16 bits	128–191	65,536	255.255.0.0
C	24 bits	192–223	256	255.255.255.0
D	N/A	224–239	N/A	n/a
E	N/A	240–255	N/A	n/a

slash notation

- ✓ The address 172.31.28.120**/16** indicates that the first 16 bits make up the network portion of the address.
- ✓ This is equivalent to a **subnet mask** of 255.255.0.0
- ✓ Class D addresses are reserved for IPv4 **multicasting**
- ✓ Class E addresses are experimental.
- ✓ Reserved for private:
  - 10.0.0.0–10.255.255.255 (10.0.0.0/8)
  - 172.16.0.0–172.31.255.255 (172.16.0.0/12)
  - **192.168.0.0**–192.168.255.255 (192.168.0.0/16)
- ✓ The **loopback** address (**127.0.0.1**)
  - special address that refers to the local computer

Unicast, Multicast addressed

- ✓ **Unicast addresses** are those addresses that are assigned to an individual computer interface.
- ✓ Classes A, B, and C comprise the unicast address space for IPv4.
- ✓ Typically, an interface on a host is assigned an IPv4 (unicast) address either statically or by a configuration protocol like Dynamic Host Configuration Protocol (**DHCP**).
- ✓ **Multicast addresses** are not assigned to a specific interface.
  - Instead, multiple computers may "join" a **multicast group** listening on a particular multicast address.

Broadcast

- ✓ The data sent to the limited broadcast address, **255.255.255.255**, will be received and processed by every machine on the **local network**.
  - bad practice
- ✓ If applications require broadcasting, it is better to use **subnet directed broadcasts**.
  - UDP datagram

IPv4 Management Protocols

- ✓ The IPv4 protocol relies on several other protocols to function.
- ✓ The three support protocols we are most interested in is the **Address Resolution Protocol** (ARP), the **Internet Control Message Protocol** (ICMP), and the **Internet Group Management Protocol** (IGMP).
- ✓ **ARP** is used to resolve the 32-bit IPv4 address into a physical or hardware address so the IPv4 packet can be wrapped in the appropriate media frame (such as an Ethernet frame).
  - C:₩> arp -a
- ✓ **ICMP** is designed to send status and error messages between IPv4 hosts.
  - C:₩> ping 127.0.0.1
- ✓ **IGMP** is used to manage multicast group membership.

ARP

- ✓ **ARP** is used to resolve the 32-bit IPv4 address into a physical or hardware address so the IPv4 packet can be wrapped in the appropriate media frame (such as an Ethernet frame).

C:\WINDOWS\system32\cmd.exe

```
C:\>arp

Displays and modifies the IP-to-Physical address translation tables used by
address resolution protocol (ARP).

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr]

  -a            Displays current ARP entries by interrogating the current
                protocol data.  If inet_addr is specified, the IP and Physical
                addresses for only the specified computer are displayed.  If
                more than one network interface uses ARP, entries for each ARP
                table are displayed.
  -g            Same as -a.
  inet_addr     Specifies an internet address.
  -N if_addr    Displays the ARP entries for the network interface specified
                by if_addr.
  -d            Deletes the host specified by inet_addr. inet_addr may be
                wildcarded with * to delete all hosts.
  -s            Adds the host and associates the Internet address inet_addr
                with the Physical address eth_addr.  The Physical address is
                given as 6 hexadecimal bytes separated by hyphens. The entry
                is permanent.
  eth_addr      Specifies a physical address.
  if_addr       If present, this specifies the Internet address of the
                interface whose address translation table should be modified.
                If not present, the first applicable interface will be used.
Example:
  > arp -s 157.55.85.212   00-aa-00-62-c6-09  .... Adds a static entry.
  > arp -a                                    .... Displays the arp table.

C:\>_
```

ICMP

✓ The **ping** command is based on the ICMP protocol.

C:\WINDOWS\system32\cmd.exe

```
C:\>ping /?

Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
            [-r count] [-s count] [[-j host-list] | [-k host-list]]
            [-w timeout] target_name

Options:
    -t             Ping the specified host until stopped.
                   To see statistics and continue - type Control-Break;
                   To stop - type Control-C.
    -a             Resolve addresses to hostnames.
    -n count       Number of echo requests to send.
    -l size        Send buffer size.
    -f             Set Don't Fragment flag in packet.
    -i TTL         Time To Live.
    -v TOS         Type Of Service.
    -r count       Record route for count hops.
    -s count       Timestamp for count hops.
    -j host-list   Loose source route along host-list.
    -k host-list   Strict source route along host-list.
    -w timeout     Timeout in milliseconds to wait for each reply.


C:\>
```

IPv6

✓ **netsh** interface ipv6 show interface

C:\WINDOWS\system32\cmd.exe - netsh

```
C:\>netsh
netsh>?

The following commands are available:

Commands in this context:
..              - Goes up one context level.
?               - Displays a list of commands.
abort           - Discards changes made while in offline mode.
add             - Adds a configuration entry to a list of entries.
alias           - Adds an alias.
bridge          - Changes to the `netsh bridge' context.
bye             - Exits the program.
commit          - Commits changes made while in offline mode.
delete          - Deletes a configuration entry from a list of entries.
diag            - Changes to the `netsh diag' context.
dump            - Displays a configuration script.
exec            - Runs a script file.
exit            - Exits the program.
firewall        - Changes to the `netsh firewall' context.
help            - Displays a list of commands.
interface       - Changes to the `netsh interface' context.
offline         - Sets the current mode to offline.
online          - Sets the current mode to online.
popd            - Pops a context from the stack.
pushd           - Pushes current context on stack.
quit            - Exits the program.
ras             - Changes to the `netsh ras' context.
routing         - Changes to the `netsh routing' context.
set             - Updates configuration settings.
show            - Displays information.
unalias         - Deletes an alias.
winsock         - Changes to the `netsh winsock' context.

The following sub-contexts are available:
 bridge diag firewall interface ras routing winsock

To view help for a command, type the command, followed by a space, and
 type ?.

netsh>
```

Addressing IPv6 from Winsock

```
struct sockaddr_in6 {
    short          sin6_family;
    u_short                sin6_port;
    u_long         sin6_flowinfo;
    struct in6_addr sin6_addr;
    u_long         sin6_scope_id;
};
```

Address and Name Resolution

- ✓ How to assign both **literal string addresses** and resolve names to the address specific **structures** for both IP protocols.
- ✓ Name resolution APIs
  - getaddrinfo()
  - getnameinfo()
- ✓ Winsock APIs for converting between string literal addresses and socket address structure.
  - WSAAddressToString()
  - WSAStringToAddress()

Name Resolution Routines

- ✓ The legacy functions like **gethostbyname**() and **inet_addr**() work with IPv4 addresses only.
- ✓ New name resolution routines are defined in **WS2TCPIP.H**.
- ✓ The **getaddrinfo**() function provides protocol-independent name resolution.

```
int getaddrinfo(
    const char FAR *nodename,
    const char FAR *servname,
    const struct addrinfo FAR *hints,
    struct addrinfo FAR *FAR *res
    );
```

```
struct addrinfo {
    int                 ai_flags;
    int                 ai_family;
    int                 ai_socktype;
    int                 ai_protocol;
    size_t              ai_addrlen;
    char                *ai_canonname;
    struct sockaddr *ai_addr;
    struct addrinfo *ai_next;
};
```

- ✓ **ai_flags**: AI_PASSIVE, AI_CANONNAME, or AI_NUMERICHOST.
  - AI_CANONNAME indicates that nodename is a computer name like www.microsoft.com
  - AI_NUMERICHOST indicates that it is a literal string address such as "10.10.10.1".
  - AI_PASSIVE will be discussed later.
- ✓ **ai_family**: AF_INET, AF_INET6, or AF_UNSPEC.
  - if AF_UNSPEC is given, then the addresses returned could be either IPv4 or IPv6 or both.
- ✓ **ai_socktype**: specifies the desired socket type, such as SOCK_DGRAM, SOCK_STREAM.
- ✓ **ai_protocol**: specifies the desired protocol, such as IPPROTO_TCP.

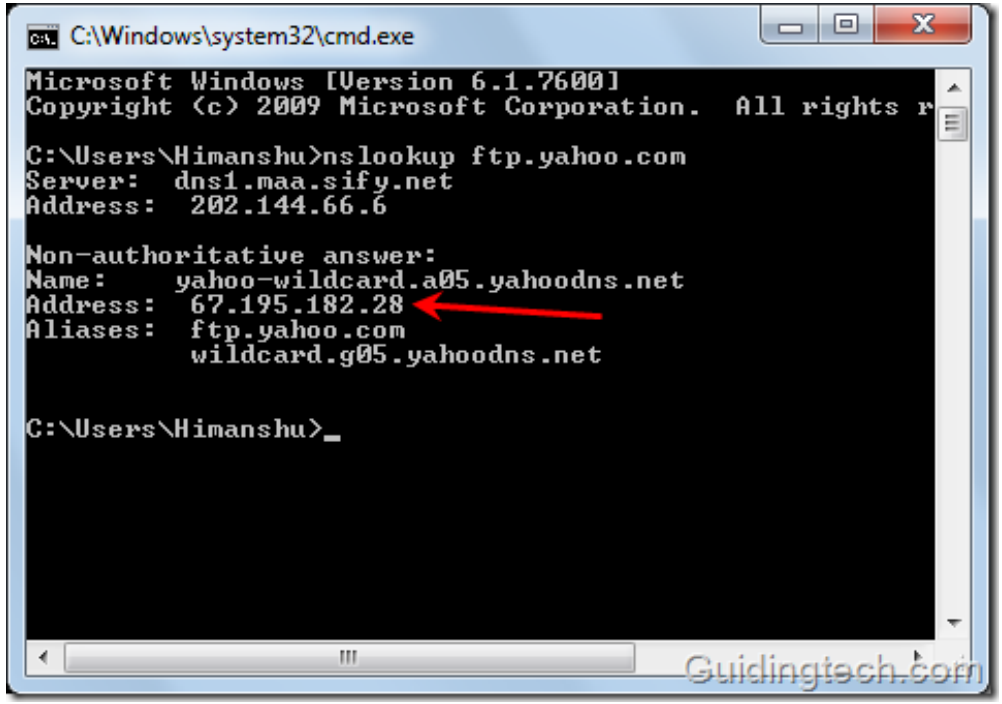
```
// Declare and initialize variables.
char* ip = "127.0.0.1";
char* port = "7777";
struct addrinfo aiHints;
struct addrinfo *aiList = NULL;
int retVal;

// Setup the hints address info structure
// which is passed to the getaddrinfo() function
memset(&aiHints, 0, sizeof(aiHints));
aiHints.ai_family = AF_INET;
aiHints.ai_socktype = SOCK_STREAM;
aiHints.ai_protocol = IPPROTO_TCP;

// Call getaddrinfo(). If the call succeeds, the aiList variable
// will hold a linked list f addrinfo structures containing
// response information about the host
if ((retVal = getaddrinfo(ip, port, &aiHints, &aiList)) != 0)
{
    printf("getaddrinfo() failed with error code %d.₩n", retVal);
}
```

- ✓ **getnameinfo**() takes a socket address structure already initialized and returns the host and service name corresponding to the address and port information.

```
int getnameinfo(
    const struct sockaddr FAR *sa,
    socklen_t salen,
    char FAR *host,
    DWORD hostlen,
    char FAR *serv,
    DWORD servlen,
    int flags
    );
```



- ✓ **nslookup** command line tool.
  - get ip from domain name

Simple Address Conversion

- ✓ To convert between string literal addresses and socket address structures, the **WSAStringToAddress**() and **WSAAddressToString**() helper APIs are available.

```
INT WSAStringToAddress(
    LPTSTR AddressString,
    INT AddressFamily,
    LPWSAPROTOCOL_INFO lpProtocolInfo,
    LPSOCKADDR lpAddress,
    LPINT lpAddressLength
    );
```

- ✓ The API functions **getservbyname**() and WSAAsyncGetServByName() take the name of a well-known service like “FTP” and return the port number that the service uses.

```
struct servent FAR * getservbyname(
    const char FAR * name,
    const char FAR * proto
    );
```

```
struct servent {
    char FAR *          s_name;
    char FAR * FAR *    s_aliases;
    short               s_port;
    char FAR *          s_proto
};
```

~~Writing IP Version-independent Program~~

- ✓ Windows IPv6 stack is a dual stack.
- ✓ That is, there is a separate stack for IPv4 and IPv6, so if a server wishes to accept both IPv4 and IPv6 connections, it must create a socket for each one.
  - Call **getaddrinfo**() with hints containing AI_PASSIVE, AF_UNSPEC, and the desired socket type and protocol along with the desired local port to listen or receive data on.
  - This will return **two addrinfo** structures: one containing the listening address for IPv4 and the other containing the listening address for IPv6.
  - For every addrinfo structure returned, **create a socket** with the ai_family, ai_socktype, and ai_protocol fields followed by calling bind() with the ai_addr and ai_addrlen members

Practice

- ✓ chapter03 → resolve project
  - ~~getaddrinfo~~
  - getnameinfo
  - getservbyname
  - gethostbyname

References

- ✓ http://www.winsocketdotnetworkprogramming.com/winsock2programming/winsock2advancedInternet3chap.html

MY **BRIGHT** FUTURE

DSU Dongseo University 동서대학교