Sockets 4: IP Addresses & Hostnames

1. Introduction and Addresses
2. Basic System Calls
3. Socket I/O Syscalls
4. **IP Addresses and Hostnames**
   - **address format conversion**
   - **hostnames**
   - **address translation**
5. Endianess, Options, Servers

IP Address Format Conversion

Often an "IP address" will be available as a *string* in dotted quad (IPv4) or colon (IPv6) format.

This is typical if an address must be passed to a program as a command-line argument.

True IP addresses are *binary numbers* (e.g., 32-bit unsigned int for IPv4).

(Note that true IP addresses must also be stored in **network byte order** rather than **host byte order**.)

Thus, it is common to have to convert "IP address" *strings* into true IP addresses.

IP Address Format Conversion (contd.)

There are a number of conversion calls:

- **inet_aton** – convert dotted quad string to IPv4 address:
  ```
  int inet_aton(const char *cp, struct in_addr *inp)
  ```
- **inet_ntoa** – convert IPv4 address to dotted quad string:
  ```
  char *inet_ntoa(struct in_addr in)
  ```
- **inet_pton** – converts a string address into a network address:
  ```
  int inet_pton(int af, const char *src, void *dst)
  ```
- **inet_ntop** – converts network address into a string:
  ```
  const char *inet_ntop(int af, const void *src,
                        char *dst, socklen_t cnt)
  ```

IP Address Format Conversion (contd.)

An example call to `inet_aton` is:

```
struct sockaddr_in servaddr;
inet_aton("131.230.133.20", &servaddr.sin_addr);
```

An example call to `inet_pton` with an IPv4 address is:

```
struct sockaddr_in servaddr;
inet_pton(AF_INET, "131.230.133.20", &servaddr.sin_addr);
```

An example call to `inet_pton` with an IPv6 address is:

```
struct sockaddr_in6 servaddr;
inet_pton(AF_INET6, "::ffff:131.230.133.20",
                    &servaddr.sin6_addr);
```

Address Translation/DNS

Because humans prefer to use **hostnames** rather than IP addresses, network-related programs will often have to translate hostnames to IP addresses.

There are **address translation** functions that can translate from hostnames to IP addresses and the reverse.

These calls do not directly/specifically invoke the **DNS** system, but will do so as part of the standard lookup procedure specified for your OS (e.g., `hosts` file, then NIS, then DNS).

Unfortunately, while the older calls are relatively straightforward, the latest, IPv6-supporting calls are quite complex.

Address Translation/DNS (contd.)

`gethostbyname` is the now older, now deprecated call, though modern versions should work for both IPv4 and IPv6:

```
struct hostent *gethostbyname(const char *name)
```

The `hostent` struct contains the returned address(es):

```
struct hostent {
  char  *h_name;            /* Official name of host */
  char **h_aliases;         /* Array of aliases, NULL terminated */
  int    h_addrtype;        /* Host address type */
  int    h_length;          /* Length of each IP address */
  char **h_addr_list;       /* Array of IP addresses, NULL terminated */
}

#define h_addr h_addr_list[0] /* First address */
```

Returns `NULL` on error and sets `h_errno` global (use `herror()` or `hstrerror()` for error string).

Address Translation/DNS (contd.)

Here is an example showing how a client could lookup a server's IP address and use it with `connect()`:

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0));

//Lookup server's IP address info using gethostbyname:
struct hostent *hostinfo;
hostinfo = gethostbyname("pc00.cs.siu.edu");

//Set up server address:
struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(1234);
servaddr.sin_addr = *(struct in_addr *)hostinfo->h_addr;  //cast is required

connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
```

Address Translation/DNS (contd.)

`getaddrinfo` is the newer call:

```
int getaddrinfo(const char *host, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **results)
```

- `host` is typically the target hostname (as a C string), else `NULL` (must have `host` or `service` non-`NULL`)
- `service` is the target *service name* or port (as a C string), else `NULL`
- `hints` is an `addrinfo` struct with certain fields set to limit the addresses returned
- `results` is a *linked list* of `addrinfo` structs containing the result addresses
- returns 0 on success else an error code (use `gai_strerror()` to get error string)

Address Translation/DNS (contd.)

The addrinfo struct is defined as:

```
struct addrinfo {
  int              ai_flags;     /* Flags (for hints)*/
  int              ai_family;    /* Address family */
  int              ai_socktype;  /* socket() type*/
  int              ai_protocol;  /* socket() protocol */
  size_t           ai_addrlen;   /* Size of ai_addr struct */
  struct sockaddr *ai_addr;      /* sockaddr struct pointer*/
  char            *ai_canonname; /* Canonical hostname */
  struct addrinfo *ai_next;      /* Next struct addrinfo in list */
  }
```

Address Translation/DNS (contd.)

By default, `service` is to be the service name for the server, as can be found in `/etc/services`. The port number (as a string) can be supplied as well, but then the `ai_flags` field of `hints` must include the value: `AI_NUMERICSERV`.

The `hints` struct must have at least the following two fields set:

- `ai_family` – `AF_INET` or `AF_INET6`
- `ai_socktype` – `SOCK_STREAM` (TCP) or `SOCK_DGRAM` (UDP)

`results` is a linked list of addresses, because machines may have multiple IP addresses and there can be multiple results given the particular `hints`.

The memory for the list is dynamically allocated, and must be reclaimed when no longer needed, using `freeaddrinfo()`.

Address Translation/DNS (contd.)

Example showing how a client can get an IP address(es):
(uses only first address obtained)

```
struct addrinfo hints;
struct addrinfo *servinfo;

//Setup hints and lookup server's IP address:
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;     //Allow IPv4 or IPv6 address
hints.ai_socktype = SOCK_STREAM; //TCP connection
hints.ai_flags = AI_NUMERICSERV; //To allow numeric port rather than service name
getaddrinfo("www.cs.siu.edu", "1234", &hints, &servinfo);

//Create socket using results:
int sockfd = socket(servinfo->ai_family, servinfo->ai_socktype, servinfo->ai_protocol);

//Connect to server using first address found:
connect(sock_fd, (struct sockaddr *)servinfo->ai_addr, servinfo->ai_addrlen);

freeaddrinfo(servinfo);  //Free the address linked list from getaddrinfo
```