NETWORK PROGRAMMING

ELEMENTARY NAME, ADDRESS CONVERSIONS AND DESIGN DECISIONS

BY

MR.PRASAD SAWANT

ELEMENTARY NAME AND ADDRESS CONVERSIONS

1. Domain name system
2. gethostbyname Function
3. RES_USE_INET6 resolver option
4. gethostbyname2 Function and IPv6 support
5. gethostbyaddr Function
6. uname and gethostname Functions
7. getservbyname and getservbyport Functions
8. Other networking information



DOMAIN NAME SYSTEM

Entries in DNS: resource records (RRs) for a host

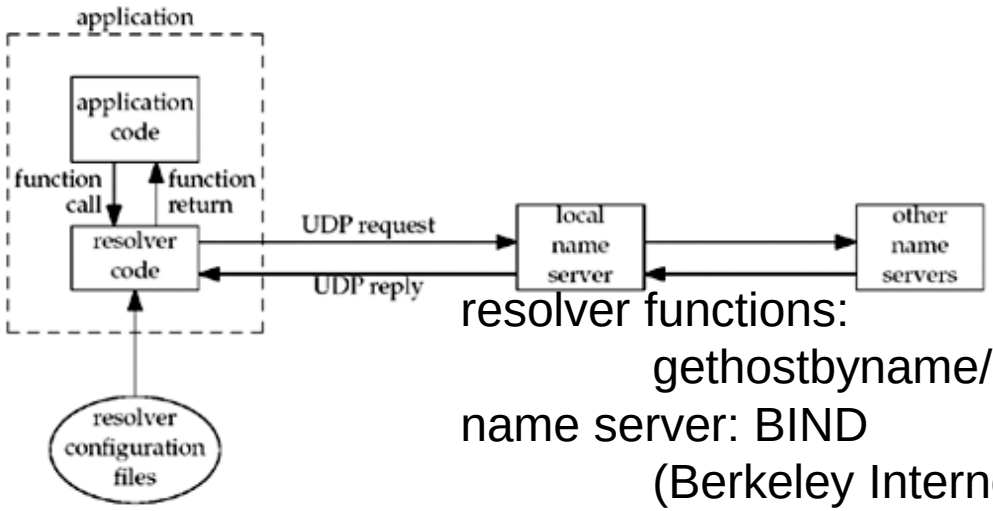
- A record: maps a hostname to a 32-bit IPv4 addr
- AAAA (quad A) record: maps to a 128-bit IPv6 addr
- PTR record: maps IP addr to hostname
- MX record: specifies a mail exchanger of the host
- CNAME record: assigns canonical name for common services

e.g.	solaris	IN	A	206.62.226.33
		IN	AAAA	5f1b:df00:ce3e:e200:0020:0800:2078:e3e3
		IN	MX	5 solaris.kohala.com
		IN	MX	10 mailhost.kohala.com
		IN	PTR	33.226.62.206.in-addr.arpa
	www	IN	CNAME	bsdi.kohala.com

DNS: APPLICATION, RESOLVER, NAME SERVERS

A **name server** is a computer server that hosts a network service for providing responses to queries against a directory service

Figure 11.1. Typical arrangement of clients, resolvers, and name servers.



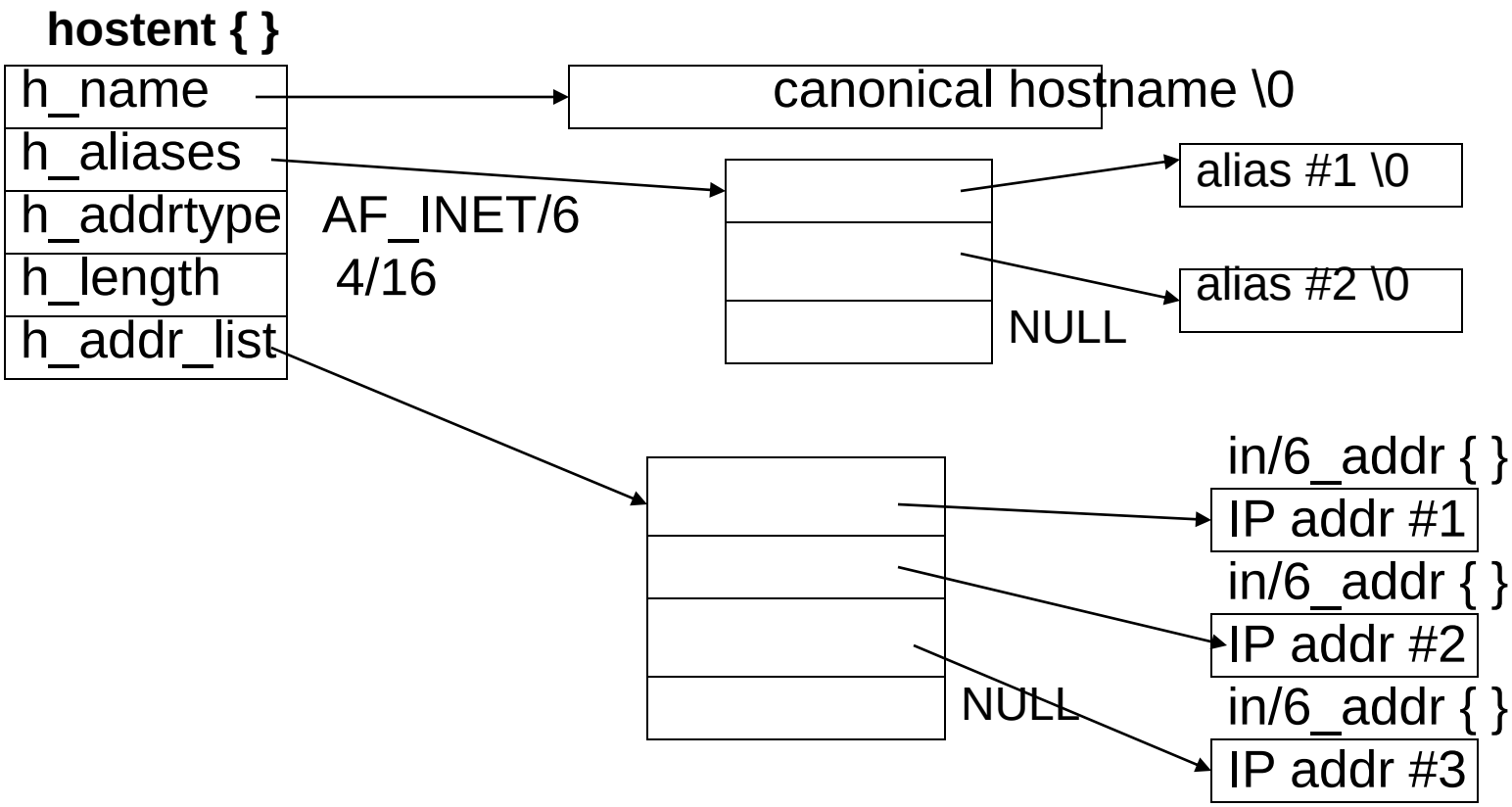
resolver functions:
gethostbyname/gethostbyaddr
name server: BIND
(Berkeley Internet Name Domain)
static hosts files (DNS alternatives):
/etc/hosts
resolver configuration file (specifies name server IPs):
/etc/resolv.conf

gethostbyname Function

performs a DNS query for an A record or a AAAA record

```
#include <netdb.h>
struct hostent *gethostbyname (const char *hostname);
        returns: nonnull pointer if OK, NULL on error with h_errno set

struct hostent
{
        char    *h_name;        /* official (canonical) name of host */
        char    **h_aliases;    /* ptr to array of ptrs to alias names */
        int     h_addrtype;     /* host addr type: AF_INET or AF_INET6 */
        int     h_length;       /* length of address: 4 or 16 */
        char    **h_addr_list;  /* ptr to array of ptrs with IPv4/IPv6 addrs */
};
#define h_addr h_addr_list[0]       /* first address in list */
```

hostent Structure Returned by gethostbyname

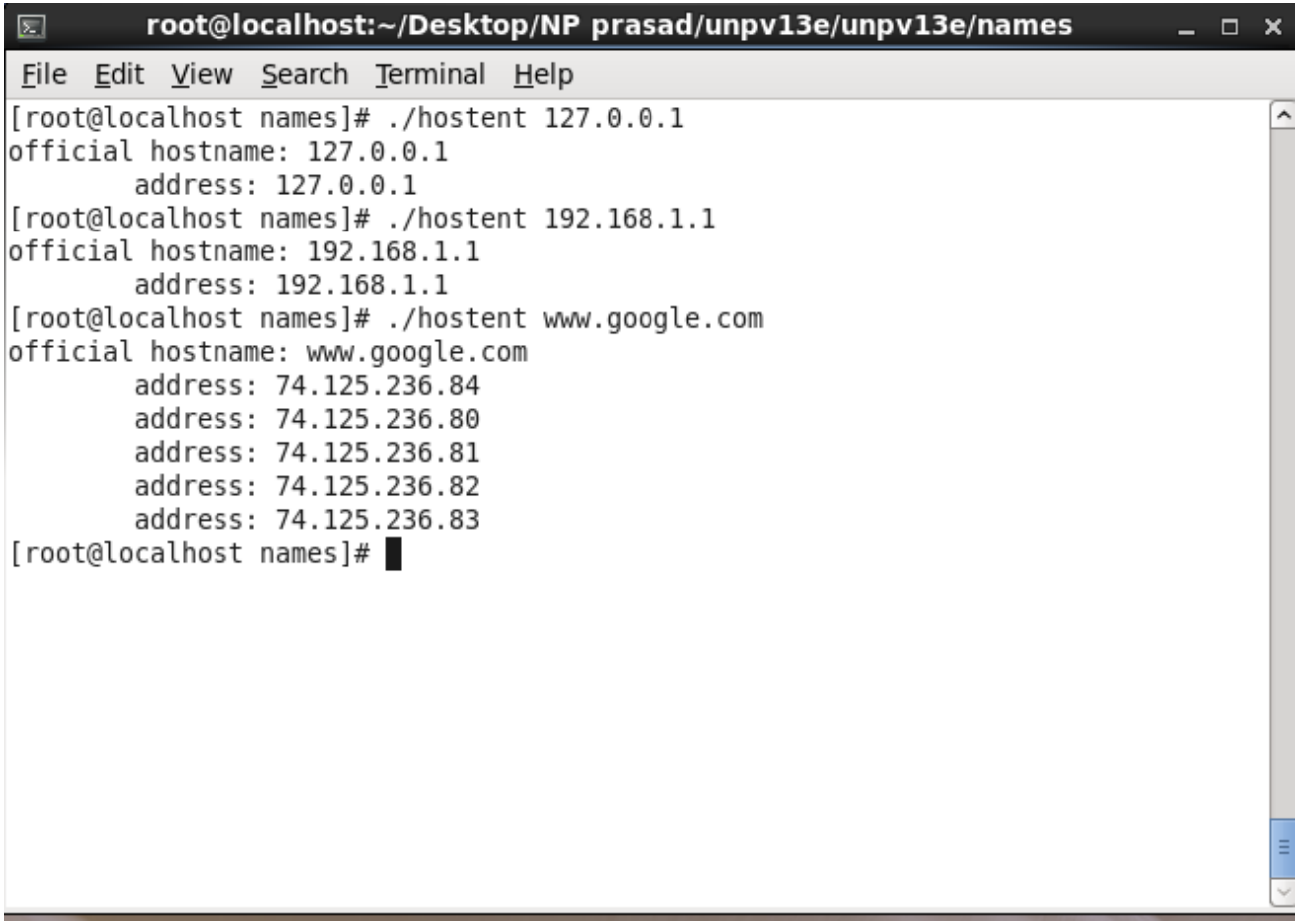


Call gethostbyname and Print Returned Info

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     char   *ptr, **pptr;
 6     char     str [INET_ADDRSTRLEN];
 7     struct hostent *hptr;

 8     while (--argc > 0) {
 9          ptr = *++argv;
10          if ( (hptr = gethostbyname (ptr) ) == NULL) {
11              err_msg ("gethostbyname error for host: %s: %s",
12                       ptr, hstrerror (h_errno) );
13              continue;
14          }
15          printf ("official hostname: %s\n", hptr->h_name);

16          for (pptr = hptr->h_aliases; *pptr ! = NULL; pptr++)
17              printf ("\talias: %s\n", *pptr);

18          switch (hptr->h_addrtype) {
19          case AF_INET:
20              pptr = hptr->h_addr_list;
21              for ( ; *pptr != NULL; pptr++)
22                  printf ("\taddress: %s\n",
23                          Inet_ntop (hptr->h_addrtype, *pptr, str, sizeof (str)));
24              break;

25          default:
26              err_ret ("unknown address type");
27              break;
28          }
29      }
30      exit(0);
31 }
```

8–14 gethostbyname is called for each command-line argument.
15–17 The official hostname is output followed by a list of alias names
18–24 pptr points to the array of pointers to the individual addresses.
For each address, we call inet_ntop and print the returned string.

O/P



RES_USE_INET6 RESOLVER OPTION

Per-application: call res_init

```
#include <resolv.h>
res_init ( );
_res.options |= RES_USE_INET6
```

Per-user: set environment variable RES_OPTIONS

```
export RES_OPTIONS=inet6
```

Per-system: update resolver configuration file

```
options inet6 (in /etc/resolv.conf)
```

For a host without a AAAA record, IPv4-mapped IPv6 addresses are returned.

gethostbyname2 FUNCTION

```
#include <sys/socket.h>
#include <netdb.h>
struct hostent *gethostbyname2 (const char *name, int af) ;
```

Returns: non-null pointer if OK, NULL on error with h_errno set

	RES_USE_INET6 option	
	off	On
Gethostbyname (host)	Search for A records. IF found, return IPv4 address (h_length=4) Else error This provides backward compatibility for all existing IPv4 applications	Search for AAAA records, If found, return IPv6 addresses (h_length = 16). Else search for A records. If found, return IPv4 mapped IPv6 addresses (h_length=16). Else error.
Gethostbyname2 (host, AF_INET)	Search for A record. IF found return IPv4 addresses (h_length = 4). Else error	Search for A record. IF found return IPv4 mapped IPv6 addresses (h_length = 16). Else error
Gethostbyname2 (host, AF_INET6)	Search for AAAA record. IF found return IPv6 addresses (h_length = 16). Else error	Search for AAAA record. IF found return IPv6 addresses (h_length = 16). Else error

gethostbyaddr FUNCTION

```
#include <netdb.h>
struct hostent *gethostbyaddr (const char *addr, socklen_t len, int family);
```

Returns: non-null pointer if OK, NULL on error with h_errno set

uname and gethostname Functions

Uname :get *name* and information about current kernel

```
#include <sys/utsname.h>
int uname (struct utsname *name);
                        returns: nonnegative value if OK, -1 on error
struct utsname {
        char    sysname[_UTS_NAMESIZE]; /* name of OS */
        char    nodename[_UTS_NODESIZE]; /* name of this node */
        char    release[_UTS_NAMESIZE]; /* OS release level */
        char    version[_UTS_NAMESIZE]; /* OS version level */
        char    machine[_UTS_NAMESIZE]; /* hardware type */
```

```
#include <unistd.h>
int gethostname (char *name, size_t namelen);
                        returns: 0 if OK, -1 on error
```

getservbyname and getservbyport Functions

```
#include <netdb.h>
struct servent *getservbyname (const char *servname, const char *protoname);
                    returns: nonnull pointer if OK, NULL on error
struct servent *getservbyport (int port, const char *protoname);
                    returns: nonnull pointer if OK, NULL on error
struct servent {
        char    *s_name;        /* official service name */
        char    **s_aliases;    /*alias list */
        int     s_port;         /* port number, network-byte order */
        char    *s_proto;       /* protocol, TCP or UDP, to use */
```

Mapping from name to port number: in /etc/services

Services that support multiple protocols often use the same TCP and UDP port number. But it's not always true:

```
        shell   514/tcp
        syslog  514/udp
```

OTHER NETWORKING INFO

Four types of info:

- hosts (gethostbyname, gethostbyaddr)
    - through DNS or /etc/hosts, hostent structure
- networks (getnetbyname, getnetbyaddr)
    - through DNS or /etc/networks, netent structure
- protocols (getprotobyname, getprotobynumber)
    - through /etc/protocols, protoent structure
- services (getservbyname, getservbyport)
    - through /etc/servies, servent structure

TCP VS. UDP

1. *Reliability*
2. *Packet Ordering*
3. *Flow Control*
4. *Full Duplex*
5. *Overhead*

APPLICATION PROTOCOL CHOICES

When you choose the application protocol, you have two important choices to make. First, you'll need to decide whether to create the protocol from scratch or use an existing protocol. Second, if you create a protocol from scratch, you'll need to determine whether it should be plain text or binary.

CLIENT-SERVER ARCHITECTURE

1. *Two-Tier Architecture*
2. *Three-Tier Architecture*

CLIENT-SIDE CONSIDERATIONS

When developing your client-server application, keep in mind how (and by whom) the client will be developed. Traditional Internet servers tend to force the user to develop a monolithic client. Some newer servers, such as database servers, provide a client API to ease the development of a client.