IE2012

System & Network Programming

Socket DNS Functions

.

Ms. Narmada Gamage

Faculty of Computing

Department of CSE

SLIIT

Discover Your Future

IE2012 – System & Network Programming

Socket DNS Functions

References:

- Stevens, Fenner, Rudoff, *UNIX Network Programming, vol. 1*, Chapters 7 and 11.

OBJECTIVE:

- Domain Name System (DNS)
- DNS Functions

Domain Name System

- Domain Name System (DNS) is used to map between hostnames and IP addresses.

  ark@cs.curtin.edu.au →134.7.1.100

Reasons for using names:

- Names are easier to remember.
- Numeric address can change, but name can remain the same.
- IPv6 numeric addresses become much longer (128 bits).
- IP address: not unique due to multihoming, IP aliases.

The binary forms of an IP address, however:

- More compact than symbolic names ❼ convenient for computer.
- Requires less computation to manipulate.
- The address occupies less memory.
- Requires less time to transmit across a network.

- Syntactically, each computer name consists of a sequence of alpha-numeric segments separated by period.
- The DNS does not specify an exact number of segments in each name nor does specify what those segments represent.
- Top level of DNS → the most significant segment.

Domain Name System (cont.)

Domain name	Assigned to
com	Commercial organization
edu	Educational institution
gov	Government organization
mil	Military group
net	Major network support center
org	Organization other than those above
arpa	Temporary arpa domain
int	International organization
country code	A country (ISO 3166)

New top-level domains are: aero, name, coop, pro, biz, info, firm, store, web, arts.

To obtain a domain, an organization must register with the Internet authority (IANA – Internet Assigned Numbers Authority) →A unique domain suffix is assigned to each organization.

Once an organization owns a particular domain, it can decide whether to introduce additional hierarchical structure.

Domain names are case insensitive → EDU = edu.

Domain names can be either a *simple* (*relative*) name (e.g., ark), or a *fully qualified domain name* (FQDN, *absolute*, e.g., ark@cs.curtin.edu.au).

The DNS client-server Model

- The entire naming system operates as a large, distributed database.
  Most organizations that have an Internet connection run a domain name server.
- Each server contains information that links the server to other domain name servers
  This results in sets of servers→ large coordinated database of names.
- Whenever an application needs to translate a name to an IP address, the application becomes a *client* of the naming system (the *server*).
- Steps taken to map a name onto an IP address:
  - An application program (*client*) calls a library procedure called the **resolver**, passing it the name as a parameter.
  - The resolver sends a UDP packet to a local DNS *server* which will look up the name and returns the IP address to the resolver.
  - The resolver then returns the address to the caller.

The DNS server hierarchy

- DNS servers are arranged in a hierarchy that matches the naming hierarchy, with each being the authority for part of the naming hierarchy.
  - **Authoritative record** is one that comes from the authority that manages the record, and is thus always correct.
  - A *root server* (the top of the hierarchy) is an authority for the top level domain (e.g., .com).
- The *root server* does not contain all possible names, but it contains information about how to reach other servers.
- A DNS server does not know which other server is an authority for a given name.
  - But each server knows the address of the *root server*.
- Stepping through the hierarchy of servers to find the server that is an authority for a name is called ***iterative query resolution***.
- All domain name servers are linked together to form a unified system.
  - Each server knows how to reach a root server.
  - Each server knows how to reach servers that are authorities for names further down the hierarchy.

Optimization of DNS performance

- Measurements have shown that the DNS as described is hopelessly inefficient
  - The traffic at a root server would be intolerable.
  - The root server would receive a request each time someone mentioned a name of a remote computer.
- Two primary optimizations used:
  - Replication: each root server is replicated.
    - Many copies of the server exist around the world.
  - Caching: Each server maintains a cache names:
    - Whenever a server looks up a new name, the server places a copy of the binding in its cache.
    - When there is a request for that binding (from client), the server checks its cache.
    - If the cache contains the answer, the server uses the cached answer to generate a reply
    - This answer is NOT an **authoritative record** ❼ Needs *time_to_live*.

- Caching works well because
  - Name resolution shows strong tendency toward *temporal locality of reference*

    In a day, a user is likely to look up the same name repeatedly.
- The *locality of reference principle* applies in the domain name system.
  - User tends to look up names of local computers more often than the names of remote computers.
  - User tends to look up the same set of domain names repeatedly.

Resource Records

- Entries in the DNS are known as ***resource records*** (RRs).
- Every domain (can be a single host or a top-level domain) can have a set of RRs associated with it.
- When a resolver gives a domain name to DNS, it gets back RRs associated with that name.
    - The real function of DNS is to map domain names onto RRs.
- A resource record is five tuple:
- ***Domain_name***: tells the domain to which this record applies.
    - ***Class***: For Internet information ❼ IN
    - ***Time_to_live***: gives an indication of how stable the record is.
    - ***Value***: Can be a number, a domain name, or an ASCII string.
    - ***Type***: Tells what kind of record this is.
- Few *types*:
    - **A** An **A** (**A**ddress) record maps a host name into a 32-bit IPv4 address. *value*: 32-bit integer.
    - **AAAA** An **AAAA** record maps a host name into a 128-bit IPv6 address. *value*: 128-bit integer.
    - **PTR** Pointer records map IP addresses into hostnames . *value*: alias for an IP address.
    - **MX** specifies a host to act as a 'mail exchanger' for the specified host. *value*: priority, domain willing to accept email.
    - **CNAME** canonical name; to assign CNAME records for common services, such as ftp and www.
        - *value*: domain name.

Example

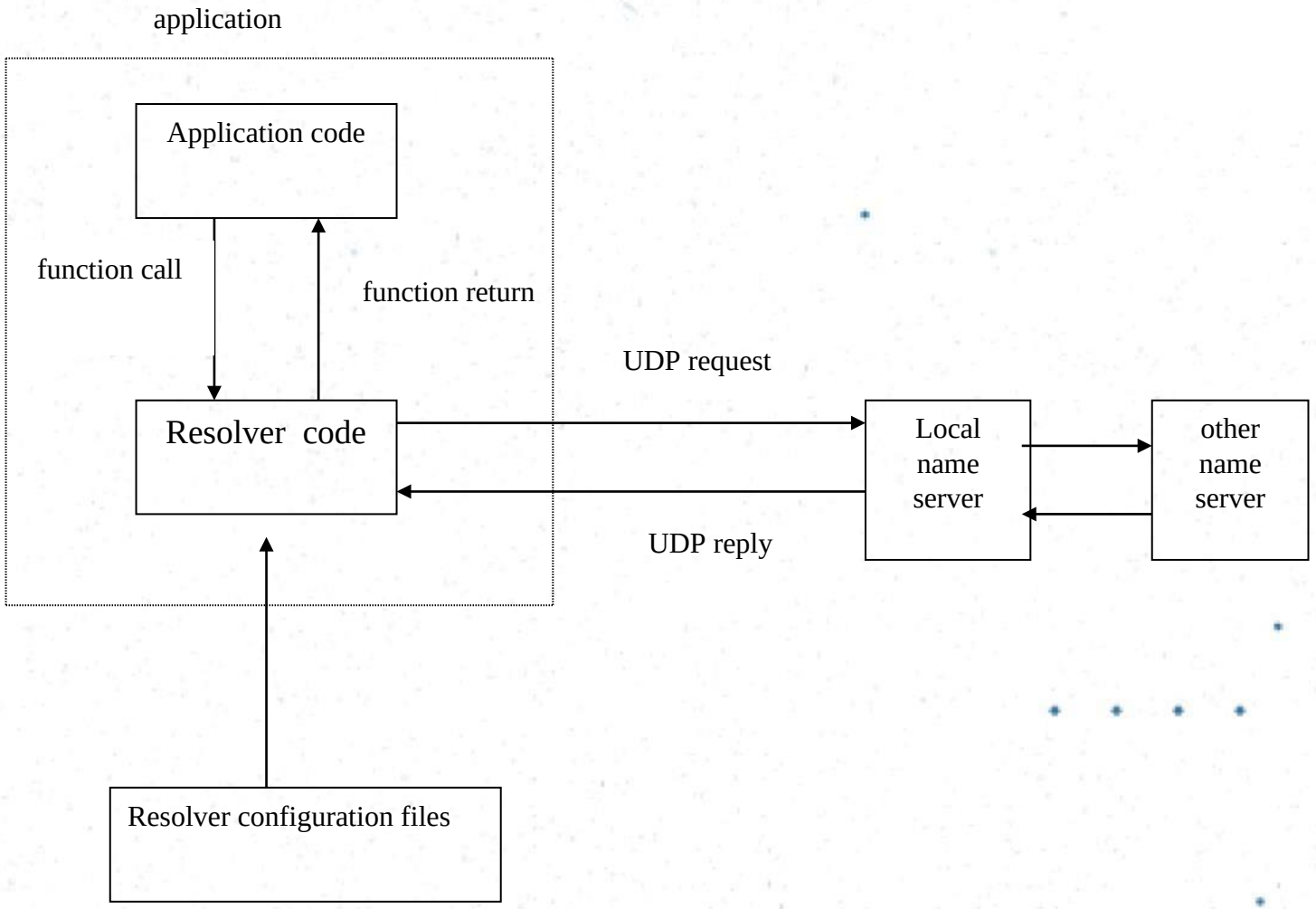
DNS records for the host solaris in the kohala.com domain (from textbook)

solaris	IN	A		206.62.226.33
	IN	MX	5	solaris.kohala.com
	IN	MX	10	mailhost.kohala.com
ftp	IN	CNAME		bsdi.kohala.com
www	IN	CNAME		bsdi.kohala.com
mailhost	IN	CNAME		bsdi.kohala.com

Resolver and name servers

- The translation of a domain name into an equivalent IP address is called **name resolution**.
- Organizations run one or more name servers.
  - The program is often known as BIND (Berkeley Internet Name Domain).
- Application programs contact a DNS server by calling functions in a library, known as *resolver*.
- Common resolver functions:
  - gethostbyname()→ maps a hostname into its IP address.
  - gethostbyaddr()→ maps an IP address into a hostname.

Typical clients, resolvers, and name servers



gethostbyname() Function

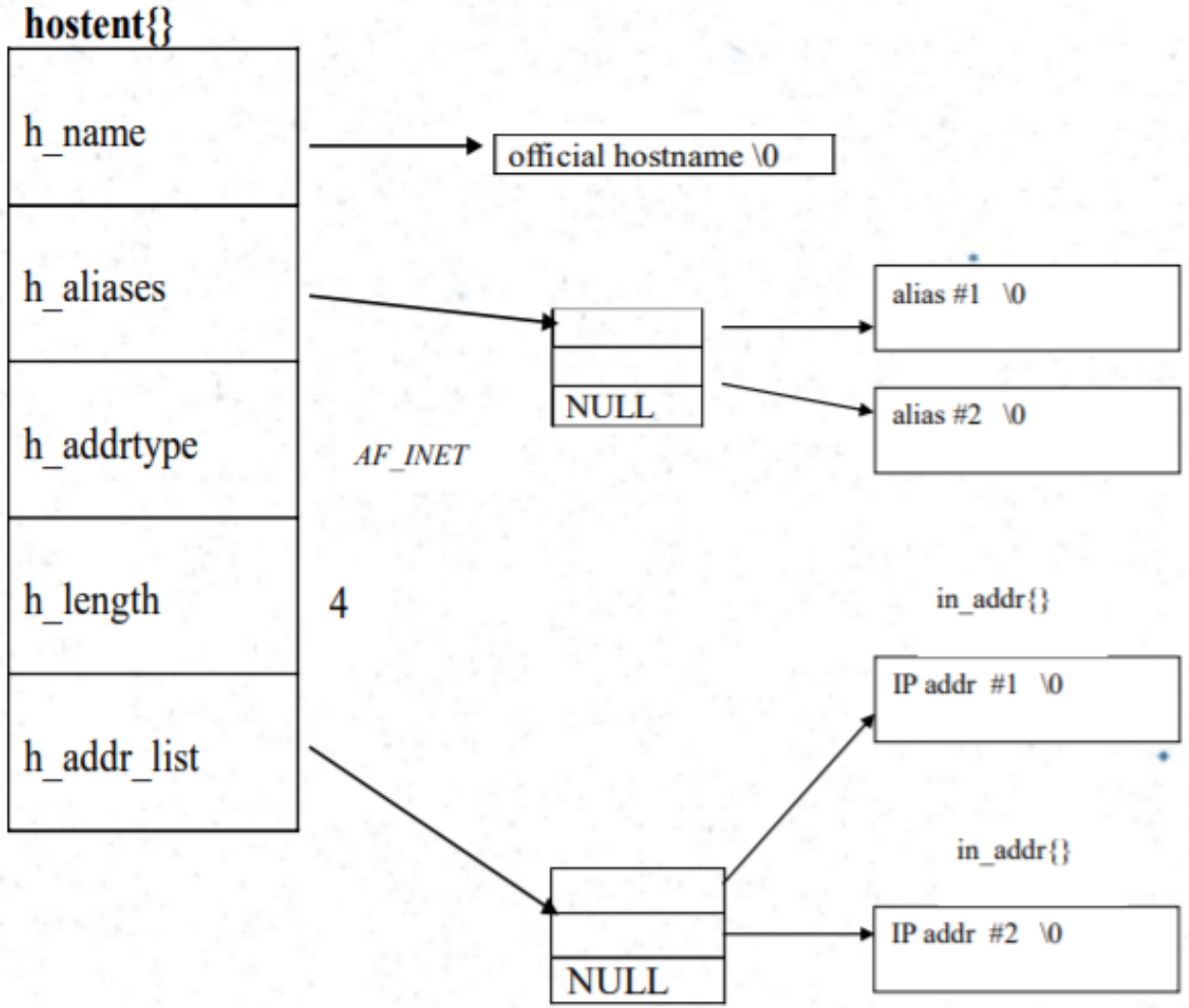
```
include <netdb.h>
/* returns nonnull pointer if OK, NULL on error with h_errno set */
struct hostent
*gethostbyname(const char *hostname);
```

- In terms of DNS, gethostbyname() performs a query for a record type **A**.
- The function returns a pointer to a **hostentstructure**.
- On error, it sets the global integer h_errno to one of the following constant:
  - HOST_NOT_FOUND
  - TRY_AGAIN
  - NO_RECOVERY
  - NO_DATA
- Use `hstrerror()` function with the above constant→`hstrerror(h_errno);`

hostent structure

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

Contents of hostent structure



Using gethostbyname() to print returned information

```
#include "unp.h"
int
main(int argc, char **argv)
{
        char  *ptr, **pptr, char  str[INET6_ADDRSTRLEN];
        struct hostent *hptr;
        while (--argc > 0) {
                ptr = *++argv;
                if ( (hptr = gethostbyname(ptr)) == NULL) {
                        err_msg("gethostbyname error for host: %s: %s", ptr,
hstrerror(h_errno)); continue;
                }
                printf("official hostname: %s\n", hptr->h_name);
                for (pptr = hptr->h_aliases; *pptr != NULL; pptr++)
                        printf("\talias: %s\n", *pptr);
                switch (hptr->h_addrtype)
                { case AF_INET:
#ifdef  AF_INET6
                case AF_INET6:
#endif pptr = hptr->h_addr_list; for ( ; *pptr != NULL; pptr++) printf("\taddress: %s\n",
                        inet_ntop(hptr->h_addrtype,
*pptr, str, sizeof(str))); break;
                default:
                        err_ret("unknown address type");
                        break;
                }
        }
        exit(0);
}
```

Fetch the domain name input as an command line argument

Call resolver function gethostbyname to obtain the IP information

Printing the hostname

Printing the aliases

Printing the addressing type

Printing the available IP address in the address list

Other functions

gethostbyname2() Function

```
include <netdb.h>
/* returns nonnull pointer if OK, NULL on error with h_errno set */
struct hostent *gethostbyname2(const char *hostname, int family);
```

- The return value is the same as gethostbyname().
- gethostbyname2() supports both IPv4 and IPv6;
  - IPv4: family = AF_INET.
  - IPv6: family = AF_INET6.

gethostbyaddr() Function

```
include <netdb.h>
/* returns nonnull pointer if OK, NULL on error with h_errno set */
struct hostent *gethostbyaddr(const char *addr, size_t len, int family);
```

- `gethostbyaddr()` takes a binary IP address and tries to find the hostname corresponding to that address.
- The function returns a pointer to the `hostent` structure ❼ interested in `h_name` field.
- `addr` is a pointer to `in_addr` or `in6_addr`❼`len` is the size of the corresponding structure.
- family argument is either `AF_INET` or `AF_INET6`.
- In terms of DNS, `gethostbyaddr()` queries a name server for a record type PTR in the *in-addr.arpa* domain for IPv4.

Cont.

uname() Function

- uname() function returns the name and some information of the current host.
- This function is **not part of resolver library**.
- The uname() function fills in utsname structure whose address is passed by the caller.

```
/* returns nonnegative value if OK, -1 on error */
#include <sys/utsname.h> int
 uname(struct   utsname*name);
#define _UTS_NAMESIZE 16 #define
_UTS_NODESIZE 256

struct utsname {
  char sysname[_UTS_NAMESIZE]; /* name of this 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*/
}
```

Determine Local Host's IP addresses

```
/* include my_addrs */
                #include    "unp.h"
char **         #include    <sys/utsname.h>
my_addrs(int *addrtype)
{
        struct hostent      *hptr;
        struct utsname      myname;

    if (uname(&myname) < 0) return(NULL);
```

Call uname(..) for utsname structor

```
if ( (hptr = gethostbyname(myname.nodename)) == NULL)
```

Call gethostbyname( ) and pass the name of the host which is returned from the uname( )

```
        return(NULL); *addrtype = hptr->h_addrtype;
    return(hptr->h_addr_list);
} /* end my_addrs */
char ** My_addrs(int
*pfamily)
{
```

my_addrs( ) rapper function

```
char      **pptr;
    if ( (pptr = my_addrs(pfamily)) == NULL)
            err_sys("my_addrs error");

    return(pptr);

}
```

Call my_addrs(…) with address family type

Other functions (cont.)

gethostname() Function

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

- gethostname function returns the name of the current host.
- name is a pointer to where the hostname is stored.
- namelen is the size of this array.

getservbyname() Function

```
#include <netdb.h>
struct     servent    *getservbyname(const char *servname, const char *protoname);
```

- This function returns a pointer to the servent structure which fills the service information.

```
struct servent {
            char          sname;        /*official service name*/
            char          **s_aliases;  /*alias list */
            int           s_port;       /*port number, network byte order*/
            char          s_proto;      /* protocol to use */
}
```

- Field of interest is the port number.
- Typical calls to this function:

```
            struct servent        *sptr;
            sptr = getservbyname("domain", "udp");    /* DNS using UDP */
            sptr = getservbyname("ftp", "tcp");         /* FTP using TCP */
            sptr = getservbyname("ftp", NULL);          /* FTP using TCP */
            sptr = getservbyname("ftp", "udp");         /* This call will fail */
```

Other functions (cont.)

getservbyport() Function

```
#include        <netdb.h>
/* returns nonnull pointer if OK, NULL on error */ struct
servent*getservbyport(int port, const char *protname);
```

- This function looks up a service given its port number and an optional protocol.

- Typical calls to this function:

```
struct servent        *sptr;

/* DNS using UDP */ sptr =
getservbyport(htons(53), “udp”);

/* FTP using TCP */ sptr =
getservbyport(htons(21), “tcp”);

/* FTP using TCP */ sptr =
getservbyport(htons(21), NULL);

/* This call will fail */ sptr =
getservbyport(htons(21),“udp”);
```