RV College of Engineering®

Go, change the world

Unit 3-

UDP client/server and Name server

Contents

There are various ways to get and set the options that affect a socket:

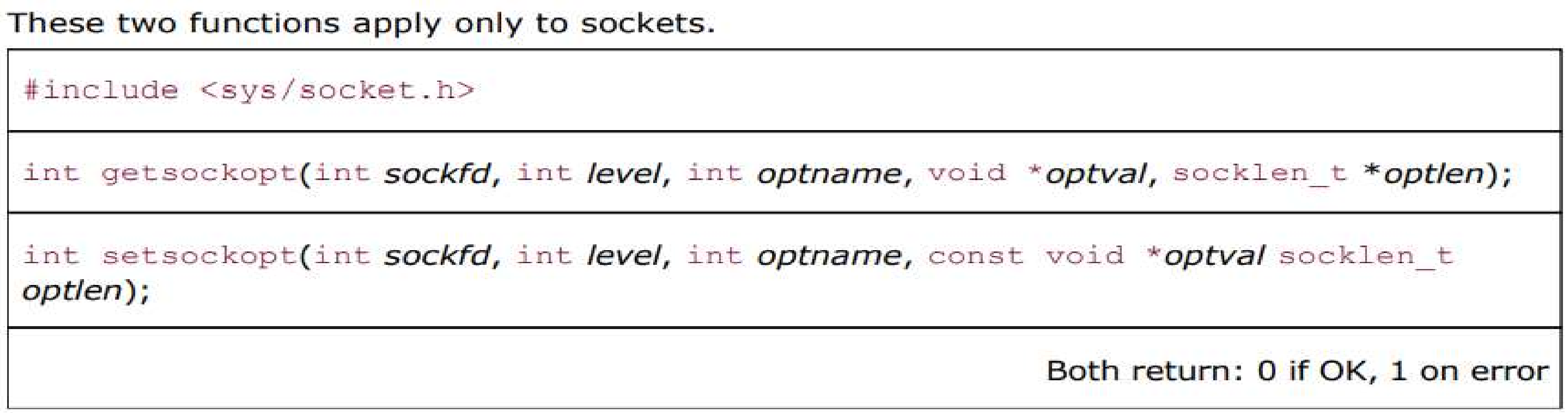
- The getsockopt and setsockopt functions
- The fcntl function
- The ioctl function

RV College of Engineering

getsockopt and setsockopt Functions:

Go, change the world

- sockfd must refer to an open socket descriptor.
- level specifies the code in the system that interprets the option: the general socket code or some protocol-specific code (e.g., IPv4, IPv6, TCP, or SCTP).
- optval is a pointer to a variable from which the new value of the option is fetched by getsockopt, or into which the current value of the option is stored by setsockopt.
- The size of this variable is specified by the final argument, as a value for setsockopt and as a value-result for getsockopt.



These two functions apply only to sockets.

```
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);

int setsockopt(int sockfd, int level, int optname, const void *optval socklen_t
optlen);

                                                     Both return: 0 if OK, 1 on error
```

- Some of the socket and IP-layer and transport layer socket options for getsockopt and setsockopt.

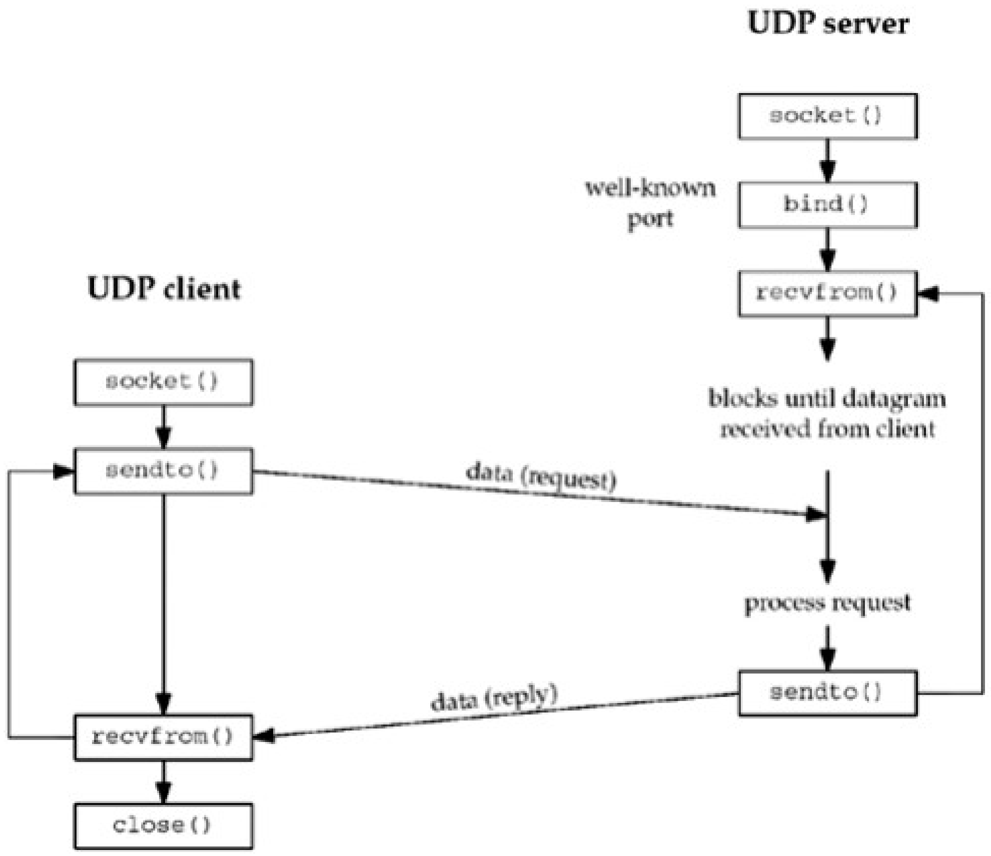
level	optname	get	set	Description	Flag	Datatype
IPPROTO_TCP	TCP_MAXSEG	•	•	Maximum segment size	•	int
IPPROTO_SCTCP	SCTCP_MAXSEG	•	•	Max fragmentation size		int
IPPROTO_IP or IPPROTO_IPV6	MCAST_JOIN_GROUP		•	Join multicast group		group_req{}
IPPROTO_IP	IP_TTL	•	•	TTL		int
SOL_SOCKET	SO_SNDBUF	•	•	Send buffer size		int

- There are two basic types of options:
    - binary options that enable or disable a certain feature (flags), and
    - options that fetch and return specific values that we can either set or examine (values).
- The column labelled "Flag" specifies if the option is a flag option.
- When calling getsockopt for these flag options, *optval is an integer.
- The value returned in *optval is zero if the option is disabled, or nonzero if the option is enabled. Similarly, setsockopt requires a nonzero *optval to turn the option on, and a zero value to turn the option off.
- If the "Flag" column does not contain a "•," then the option is used to pass a value of the specified datatype between the user process and the system.

Introduction to UDP

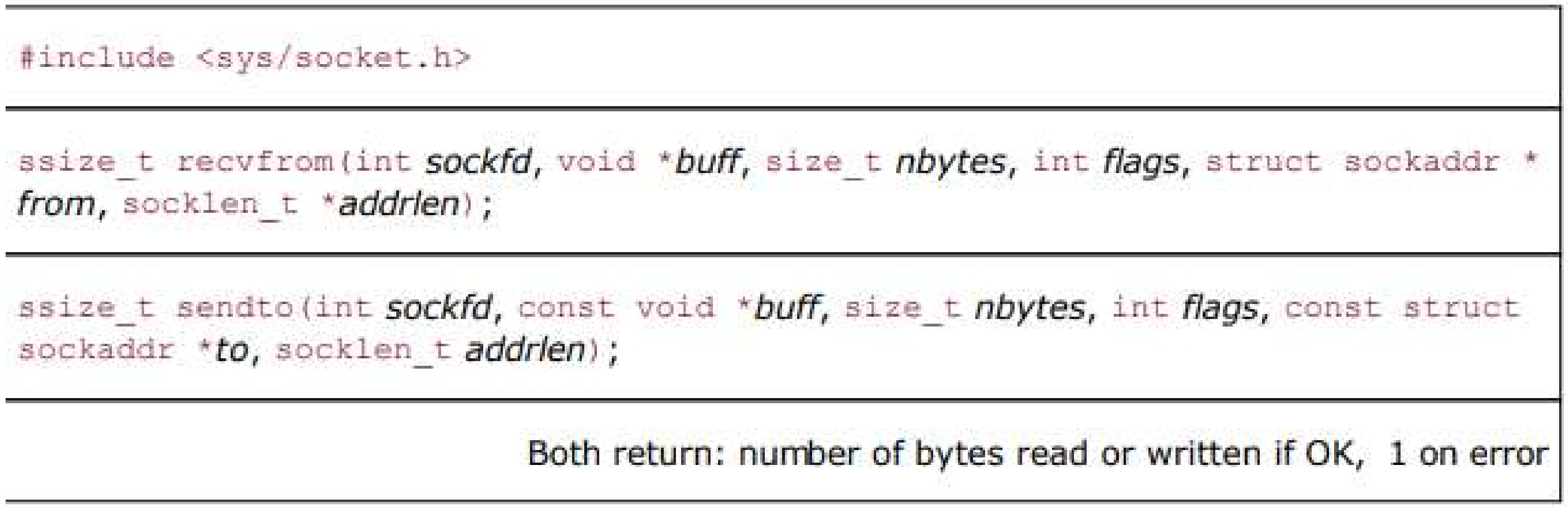
- There are fundamental differences between applications written using TCP versus those that use UDP.
- It is because differences in UDP and TCP protocols.
- UDP is a connectionless, unreliable, datagram protocol.
- TCP is connection-oriented, reliable byte stream.
- Applications built using UDP are: DNS, NFS, and SNMP

Socket functions for UDP client and server



- The client does not establish a connection with the server.
- Instead, the client just sends a datagram to the server using the sendto().
- Sendto() takes the address of the destination (the server) as a parameter.
- The server does not accept a connection from a client.
- Instead, the server calls the recvfrom(), which waits until data arrives from some client.
- Recvfrom returns the protocol address of the client, along with the datagram, so the server can send a response to the correct client.

sendto() and recvfrom()



```
#include <sys/socket.h>
```

```
ssize_t recvfrom(int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr *
from, socklen_t *addrlen);
```

```
ssize_t sendto(int sockfd, const void *buff, size_t nbytes, int flags, const struct
sockaddr *to, socklen_t addrlen);
```

Both return: number of bytes read or written if OK, 1 on error

- The first three arguments, sockfd, buff, and nbytes, are identical to the first three arguments of send() and recv() : descriptor, pointer to buffer to read into or write from, and number of bytes to read or write.
- The flag argument is set to 0.
- **to** argument for sendto is a socket address structure containing the protocol address (e.g., IP address and port number) of where the data is to be sent.
- The size of this socket address structure is specified by addrlen.

- The recvfrom function fills in the socket address structure pointed to by **<u>from</u>** with the protocol address of who sent the datagram.
- The number of bytes stored in this socket address structure is also returned to the caller in the integer pointed to by addrlen.
- *Note*: the final argument to sendto is an integer value, while the final argument to recvfrom is a pointer to an integer value (a value-result argument).

- The final two arguments to recvfrom are similar to the final two arguments to accept: The contents of the socket address structure upon return tell us *who sent the datagram (in the case of UDP) or who initiated the connection (in the case of TCP).*
- The final two arguments to sendto are similar to the final two arguments to connect: We fill in the socket address structure with the protocol address of *where to send the datagram (in the case of UDP) or with whom to establish a connection (in the case of TCP).*

- Both functions return the length of the data that was read or written as the value of the function.
- In the typical use of recvfrom, with a datagram protocol, the return value is the amount of user data in the datagram received.
- Writing a datagram of length 0 is acceptable.
- *In* UDP, this results in an IP datagram containing an IP header (normally 20 bytes for IPv4 and 40 bytes for IPv6), an 8-byte UDP header, and no data.
- This also means that a return value of 0 from recvfrom is acceptable for a datagram protocol.
- It does not mean that the peer has closed the connection, as does a return value of 0 from read on a TCP socket. Since UDP is connectionless, there is no such thing as closing a UDP connection.

- If the from argument to recvfrom is a null pointer, then the corresponding length argument (addrlen) must also be a null pointer, and this indicates that we are not interested in knowing the protocol address of who sent us data.

Lost Datagrams

- UDP client/server example is not reliable.
- If a client datagram is lost (say it is discarded by some router between the client and server), the client will block forever in its call to recvfrom.
- Similarly, if the client datagram arrives at the server but the server's reply is lost, the client will again block forever in its call to recvfrom.
- A typical way to prevent this is to place a timeout on the client's call to recvfrom

Lost Datagrams

There are three ways to place a timeout on an I/O operation involving a socket:

- Call alarm, which generates the SIGALRM signal when the specified time has expired.
- Block waiting for I/O in select, which has a time limit built-in, instead of blocking in a call to read or write.
- Use the newer SO_RCVTIMEO and SO_SNDTIMEO socket options.

Domain Name Systems(DNS).

- The DNS is used primarily to map between hostnames and IP addresses.
- A hostname can be either a simple name, such as solaris or freebsd or google.
- A fully qualified domain name (FQDN) such as solaris.unpbook.com.
- An FQDN is also called an absolute name and must end with a DOT, but users often omit the ending period.
- The trailing DOT tells the resolver that this name is fully qualified and it doesn't need to search its list of possible domains.

- Entries in the DNS are known as resource records (RRs).
- Few types of RRs that are of interest are:
- A- An A record maps a hostname into a 32-bit IPv4 address.
- AAAA- a AAAA record, called a "quad A" record, maps a hostname into a 128-bit IPv6 address.
  - The term "quad A" is used because a 128-bit address is four times larger than a 32-bit address.
- PTR - PTR records (called "pointer records") map IP addresses into hostnames. PTR records are used for the reverse DNS lookup

- A reverse DNS lookup is the opposite of this process: it is a query that starts with the IP address and looks up the domain name.
- While DNS A records are stored under the given domain name, DNS PTR records are stored under the IP address — reversed, and with ".in-addr.arpa" added.
- For example, the PTR record for the IP address 192.0.2.255 would be stored under "255.2.0.192.in-addr.arpa".
- "in-addr.arpa" has to be added because PTR records are stored within the .arpa top-level domain in the DNS.
- .arpa is a domain used mostly for managing network infrastructure, and it was the first top-level domain name defined for the Internet.

- IPv6 addresses are constructed differently from IPv4 addresses.
- IPv6 PTR records exist in a different namespace within .arpa.
- IPv6 PTR records are stored under the IPv6 address, reversed and converted into four-bit sections (as opposed to 8-bit sections, as in IPv4), plus ".ip6.arpa".
- Uses for reverse DNS include:
  - Anti-spam: Some email anti-spam filters use reverse DNS to check the domain names of email addresses and see if the associated IP addresses are likely to be used by legitimate email servers.
  - Troubleshooting email delivery issues: Because anti-spam filters perform these checks, email delivery problems can result from a misconfigured or missing PTR record.
  - Logging: System logs typically record only IP addresses; a reverse DNS lookup can convert these into domain names for logs that are more human-readable.

- MX - An MX record specifies a host to act as a "mail exchanger" for the specified host.
- The MX record indicates how email messages should be routed in accordance with the Simple Mail Transfer Protocol

Example of an MX record:

example.com	record type:	priority:	value:	TTL
@	MX	10	mailhost1.example.com	45000
@	MX	20	mailhost2.example.com	45000

- The 'priority' numbers before the domains for these MX records indicate preference; the lower 'priority' value is preferred.
- The server will always try mailhost1 first because 10 is lower than 20. In the result of a message send failure, the server will default to mailhost2.

What is the process of querying an MX record?

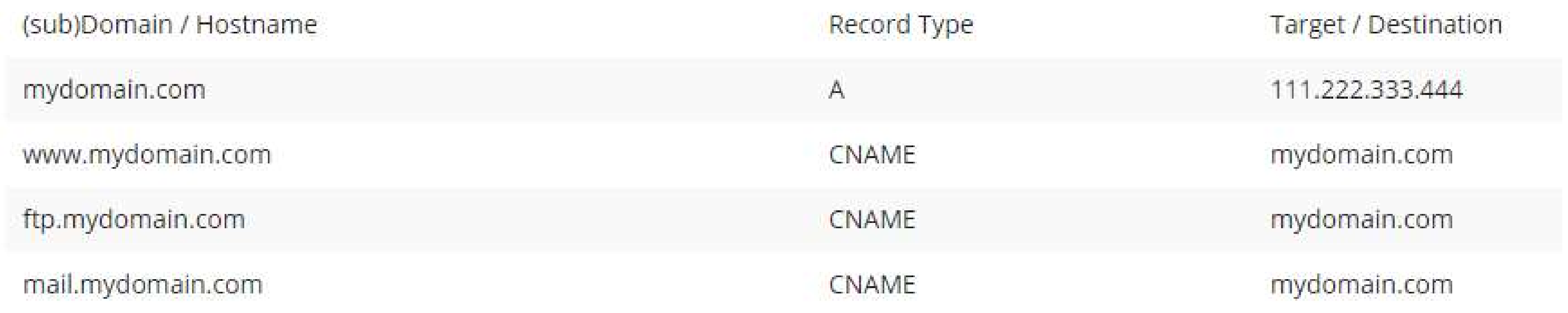
- Message transfer agent (MTA) software is responsible for querying MX records.
- When a user sends an email, the MTA sends a DNS query to identify the mail servers for the email recipients.
- The MTA establishes an SMTP connection with those mail servers, starting with the prioritized domains (in the first example above, mailhost1).

- CNAME The ‘canonical name’ (CNAME) record is used in lieu of an A record, when a domain or subdomain is an alias of another domain.
- All CNAME records must point to a domain, never to an IP address.
- A common use is to assign CNAME records for common services, such as ftp and www. If people use these service names instead of the actual hostnames, it is transparent when a service is moved to another host.

- For example, suppose blog.example.com has a CNAME record with a value of 'example.com' (without the 'blog').
- This means when a DNS server hits the DNS records for blog.example.com, it actually triggers another DNS lookup to example.com, returning example.com's IP address via its A record.
- In this case we would say that example.com is the canonical name (or true name) of blog.example.com.
- Oftentimes, when sites have subdomains such as blog.example.com or shop.example.com, those subdomains will have CNAME records that point to a root domain (example.com).
- This way if the IP address of the host changes, only the DNS A record for the root domain needs to be updated and all the CNAME records will follow along with whatever changes are made to the root.

For example, say you have several subdomains, like www.mydomain.com, ftp.mydomain.com, mail.mydomain.com etc and you want these sub domains to point to your main domain name mydomain.com. Instead of creating A records for each sub-domain and binding it to the IP address of your domain you can create CNAME records.

As you can see in the table below, in the case where the IP address of your server changes, you only need to update one A record and all the subdomains follow automatically because all the CNAMES point to the main domain with the A record:



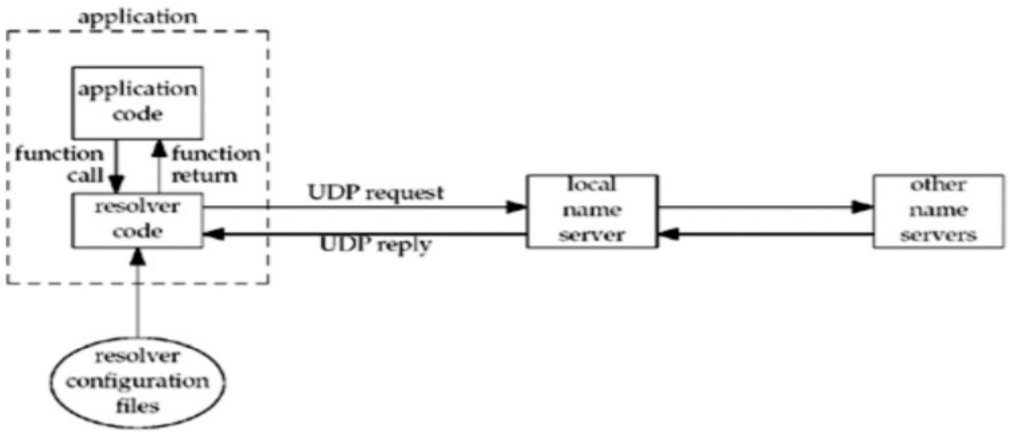
(sub)Domain / Hostname	Record Type	Target / Destination
mydomain.com	A	111.222.333.444
www.mydomain.com	CNAME	mydomain.com
ftp.mydomain.com	CNAME	mydomain.com
mail.mydomain.com	CNAME	mydomain.com

Limitations of CNAME records:

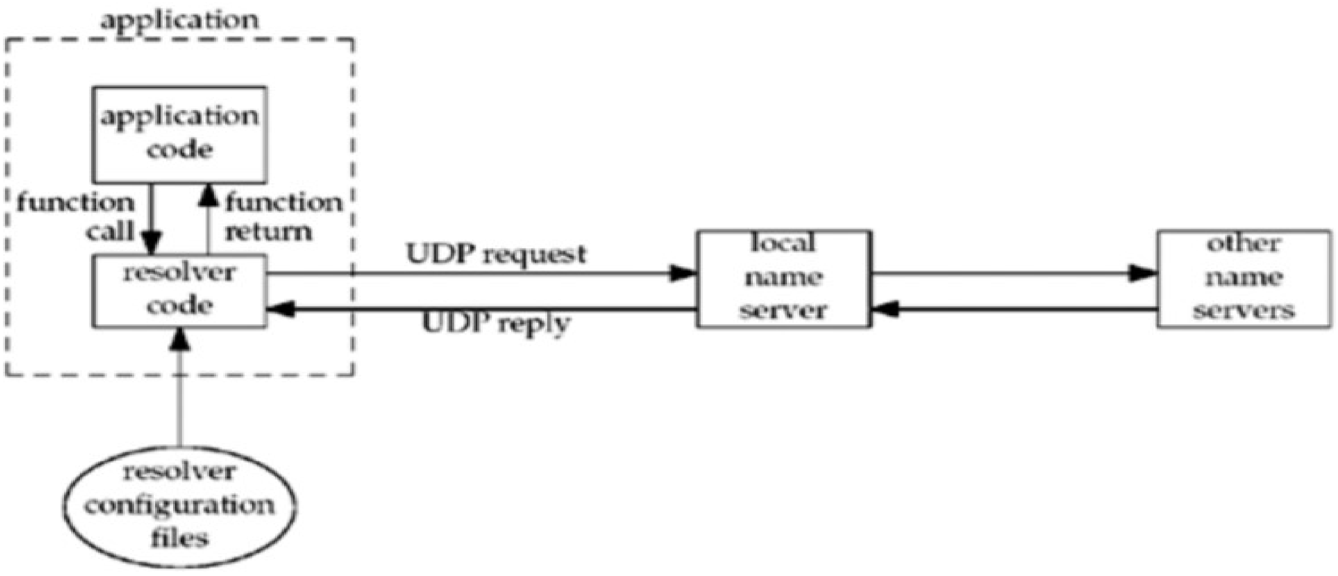
1. Usage of CNAME records means that there is an additional request sent to the DNS servers, which can cause a delay of a few milliseconds.
2. You cannot create a CNAME record for the main domain name (mydomain.com) itself, this must be an A record.
   For example, you cannot map mydomain.com to google.com, however, you can map google.mydomain.com to google.com.
3. MX or NS (nameserver) records may never point to a CNAME record, only A records.

Resolvers and Name Servers.

- Organizations run one or more name servers, often the program known as BIND (Berkeley Internet Name Domain).
- Applications such as the clients and servers that we are writing in this text contact a DNS server by calling functions in a library known as the resolver.
- The common resolver functions are gethostbyname and gethostbyaddr.
- The former maps a hostname into its IPv4 addresses, and the latter does the reverse mapping.



Arrangements of clients ,resolvers and name Servers.



- On some systems, the resolver code is contained in a system library and is link-edited into the application when the application is built.
- On others, there is a centralized resolver daemon that all applications share, and the system library code performs RPCs to this daemon.
- In either case, application code calls the resolver code using normal function calls, typically calling the functions gethostbyname and gethostbyaddr.
- The resolver code reads its system-dependent configuration files to determine the location of the organization's name servers.
- The file /etc/resolv.conf normally contains the IP addresses of the local name servers.

- The resolver sends the query to the local name server using UDP.
- If the local name server does not know the answer, it will normally query other name servers across the Internet, also using UDP.
- If the answers are too large to fit in a UDP packet, the resolver will automatically switch to TCP.

DNS Alternatives

- It is possible to obtain name and address information without using the DNS.
- Common alternatives are static host files (normally the file /etc/hosts), the Network Information System (NIS) or Lightweight Directory Access Protocol (LDAP).
- It is implementation-dependent how an administrator configures a host to use the different types of name services.
- If a name server is to be used for hostname lookups, then all these systems use the file /etc/resolv.conf to specify the IP addresses of the name servers.
- Fortunately, these differences are normally hidden to the application programmer, so we just call the resolver functions such as gethostbyname and gethostbyaddr.

Gethostbyname Function

- Host computers are normally known by human-readable names.
- But functions, connect, accept, bind…… these use IP addresses.
- Most applications should deal with names, not addresses. (especially IPv6).
- The most basic function that looks up a hostname is gethostbyname.
- If successful, it returns a pointer to a hostent structure that contains all the IPv4 addresses for the host.
- However, it is limited in that it can only return IPv4 addresses.(newer variant is getaddrinfo())

Gethostbyname Function

```
#includc <nctdb.h>
```

```
struct hostent *gethostbyname (const char *hostname);
```

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

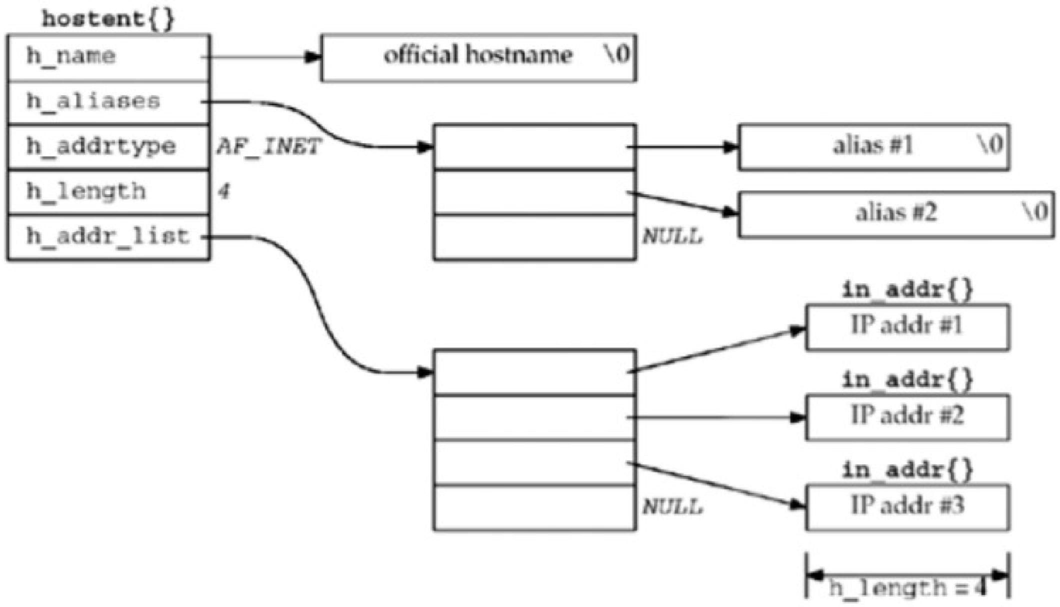
The non-null pointer returned by this function points to the following hostent structure:

```
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 */
   int    h_length;      /* length of address: 4 */
   char **h_addr_list;   /* ptr to array of ptrs with IPv4 addrs */
};
```

Gethostbyname Function

- In terms of the DNS, gethostbyname performs a query for an A record. This function can return only IPv4 addresses.

Arrangements of hostent structure

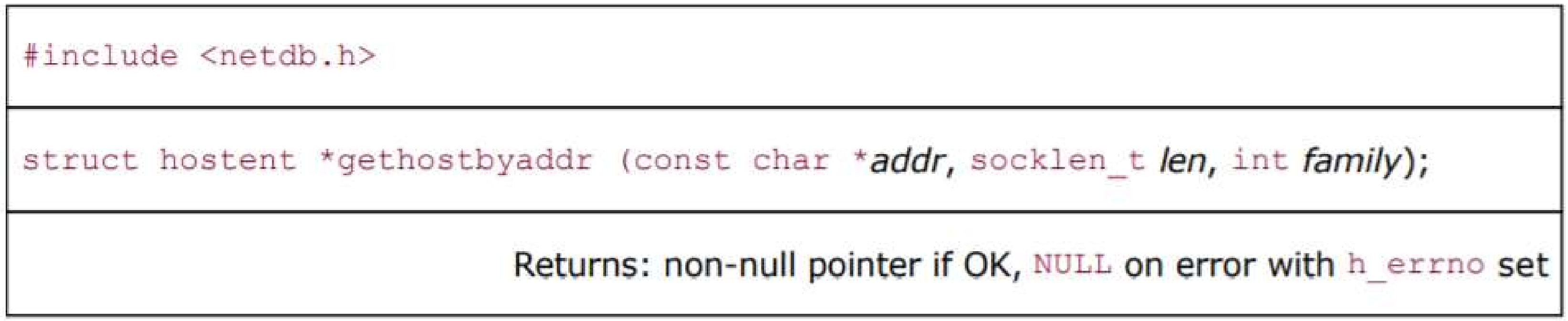


- gethostbyname differs from the other socket functions as it does not set errno when an error occurs.
- Instead, it sets the global integer h_errno to one of the following constants defined by including <netdb.h>:
    - HOST_NOT_FOUND
    - TRY_AGAIN
    - NO_RECOVERY
    - NO_DATA (identical to NO_ADDRESS)

The NO_DATA error means the specified name is valid, but it does not have an A record.

gethostbyaddr Function

- The function gethostbyaddr takes a binary IPv4 address and tries to find the hostname corresponding to that address.
- This is the reverse of gethostbyname.

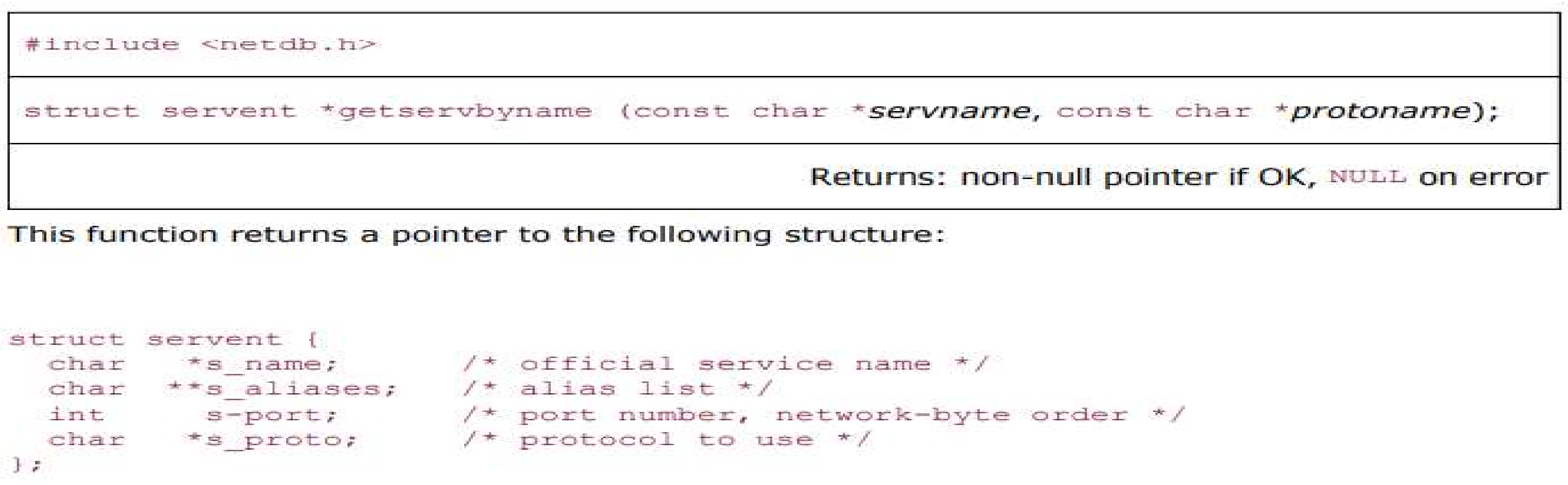


```
#include <netdb.h>

struct hostent *gethostbyaddr (const char *addr, socklen_t len, int family);

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

- This function returns a pointer to the same hostent structure
- The field of interest in this structure is normally h_name, the canonical hostname.
- The addr argument is not a char*, but is really a pointer to an in_addr structure containing the IPv4 address.
- len is the size of this structure: 4 for an IPv4 address.
- The family argument is AF_INET.
- In terms of the DNS, gethostbyaddr queries a name server for a PTR record in the in-addr.arpa domain

getservbyname Functions:

- If we refer to a service by its name in our code, instead of by its port number, and if the mapping from the name to port number is contained in a file (normally /etc/services), then if the port number changes, all we need to modify is one line in the /etc/services file instead of having to recompile the applications.
- The next function, getservbyname, looks up a service given its name.



```
#include <netdb.h>

struct servent *getservbyname (const char *servname, const char *protoname);
```

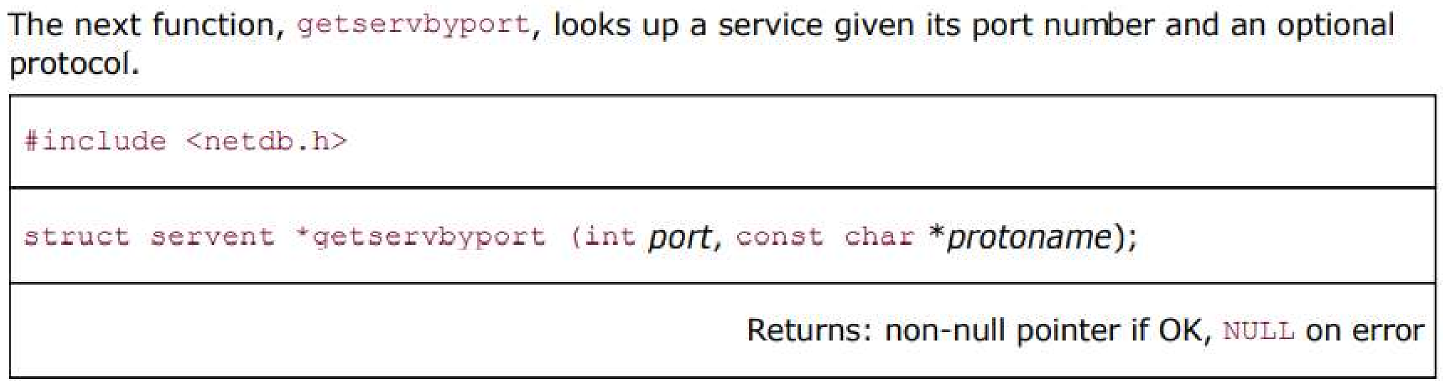
Returns: non-null pointer if OK, `NULL` on error

This function returns a pointer to the following structure:

```
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 to use */
};
```

getservbyport Functions:

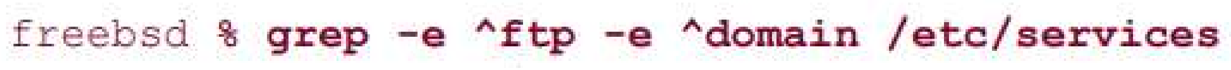
- getservbyport, looks up a service given its port number and an optional protocol.
- The port value must be network byte ordered.



The next function, `getservbyport`, looks up a service given its port number and an optional protocol.

```
#include <netdb.h>

struct servent *getservbyport (int port, const char *protoname);

Returns: non-null pointer if OK, NULL on error
```

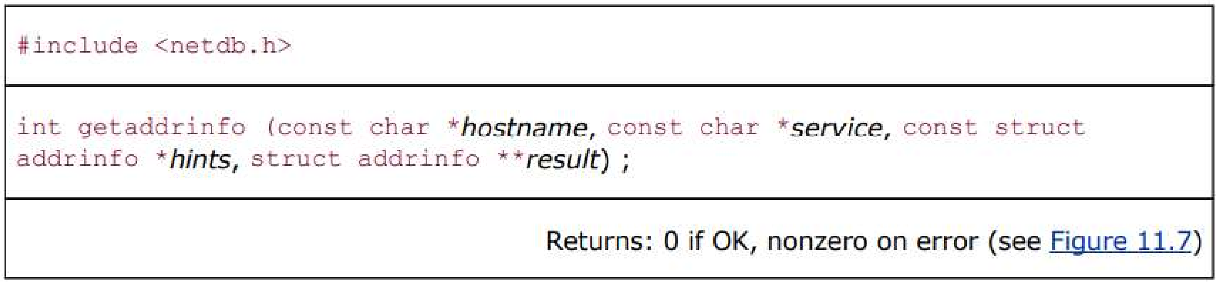
- Use the folloing command to know port and protocol combinations



```
freebsd % grep -e ^ftp -e ^domain /etc/services
```

getaddrinfo Function:

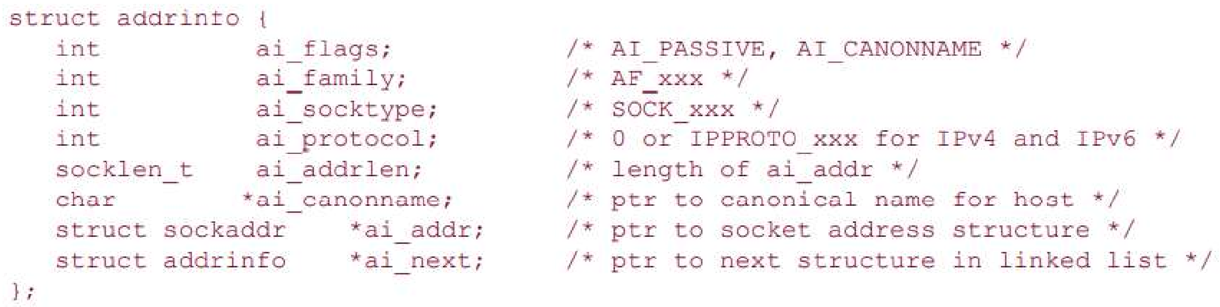
- getaddrinfo function is for IPv6 , where as gethostbyname and gethostbyaddr functions only support IPv4.
- The getaddrinfo function handles both name-to-address and service-to-port translation, and returns sockaddr structures instead of a list of addresses.
- These sockaddr structures can then be used by the socket functions directly
- This function is defined in the POSIX specification.
- The POSIX definition of this function comes from an earlier proposal by Keith Sklower for a function named getconninfo.



```
#include <netdb.h>

int getaddrinfo (const char *hostname, const char *service, const struct
addrinfo *hints, struct addrinfo **result) ;
```

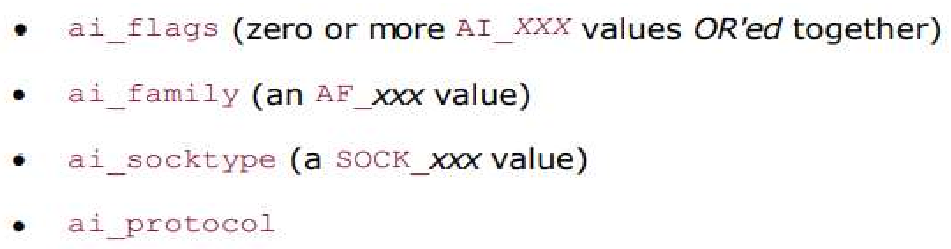
Returns: 0 if OK, nonzero on error (see Figure 11.7)

- This function returns through the **result** pointer a pointer to a linked list of addrinfo structures, which is defined by including <netdb.h>
- The **hostname** is either a hostname or an address string (dotted-decimal for IPv4 or a hex string for IPv6). The service is either a service name or a decimal port number string.
- **hints** is either a null pointer or a pointer to an addrinfo structure that the caller fills in with hints about the types of information the caller wants returned. For example, if the specified service is provided for both TCP and UDP (e.g., the domain service, which refers to a DNS server), the caller can set the ai_socktype member of the hints structure to SOCK_DGRAM. The only information returned will be for datagram sockets.



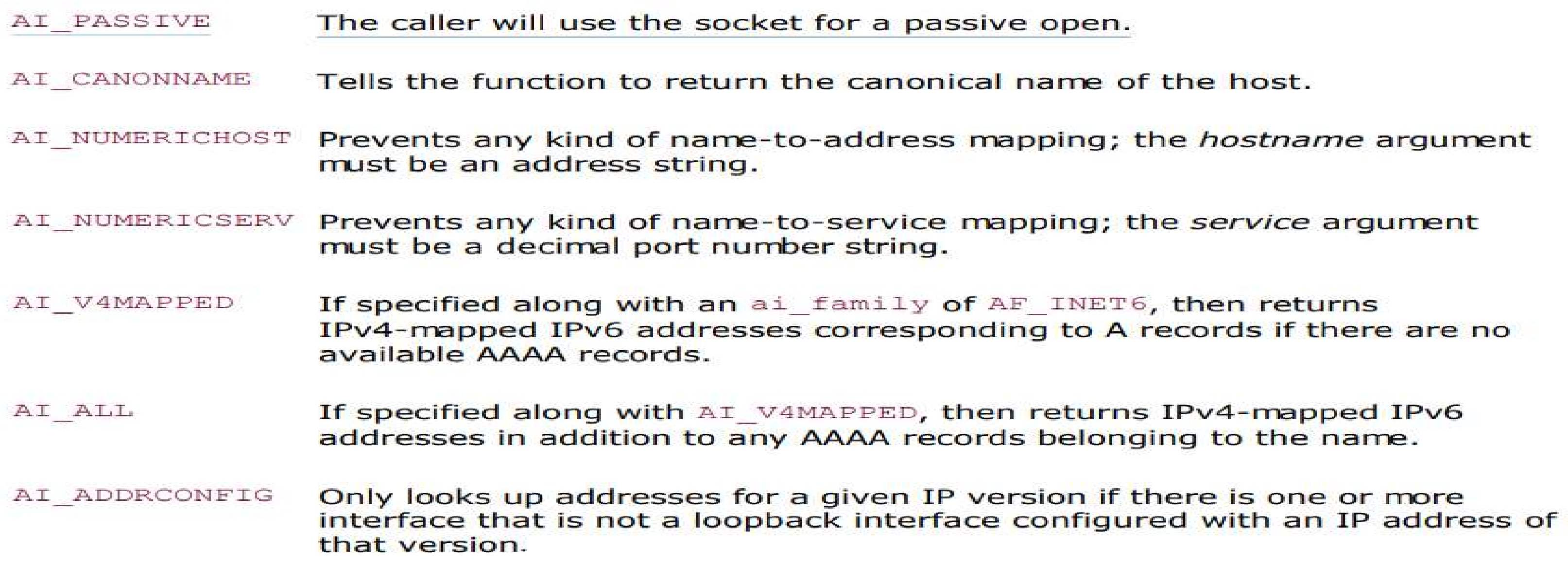
```
struct addrinfo {
   int          ai_flags;           /* AI_PASSIVE, AI_CANONNAME */
   int          ai_family;          /* AF_xxx */
   int          ai_socktype;        /* SOCK_xxx */
   int          ai_protocol;        /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
   socklen_t    ai_addrlen;         /* length of ai_addr */
   char        *ai_canonname;       /* ptr to canonical name for host */
   struct sockaddr    *ai_addr;     /* ptr to socket address structure */
   struct addrinfo    *ai_next;     /* ptr to next structure in linked list */
};
```

- The members of the hints structure that can be set by the caller are:



- `ai_flags` (zero or more `AI_XXX` values *OR'ed* together)
- `ai_family` (an `AF_xxx` value)
- `ai_socktype` (a `SOCK_xxx` value)
- `ai_protocol`

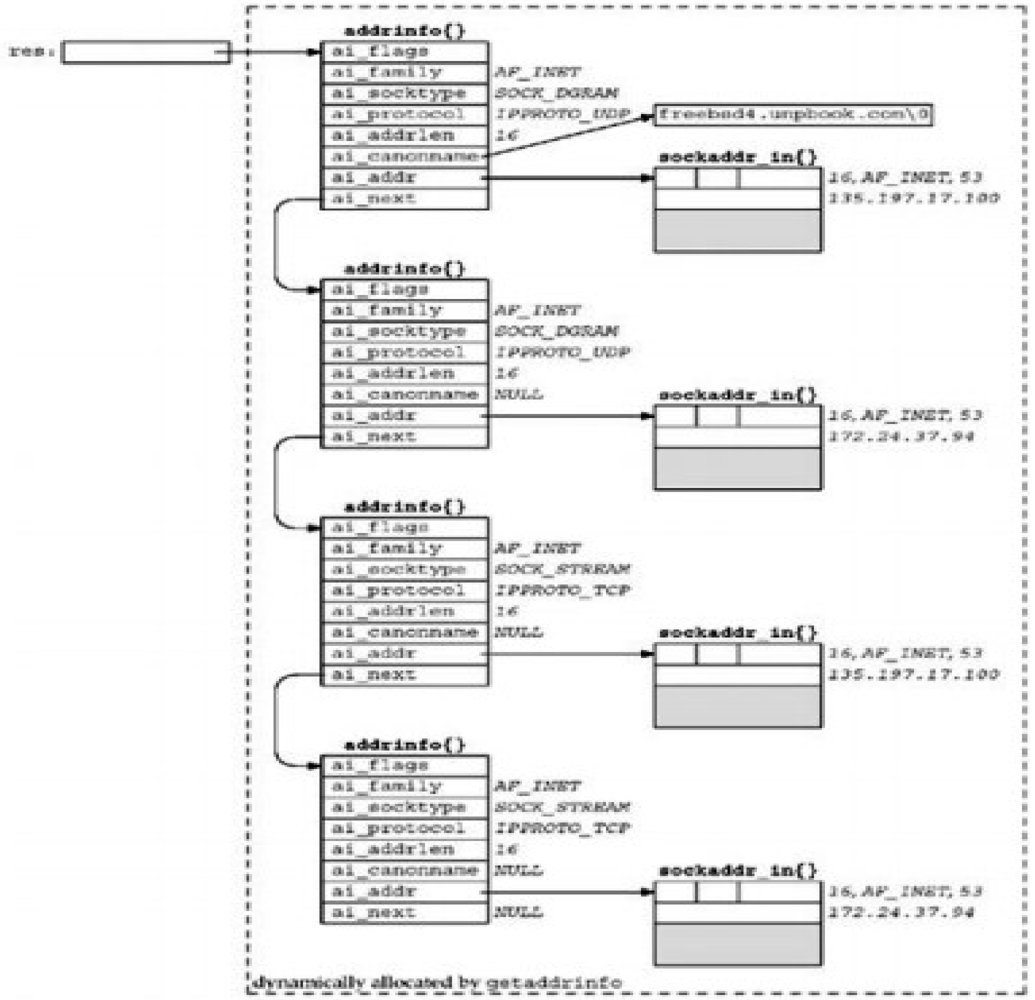
- The possible values for the ai_flags member and their meanings are:



AI_PASSIVE	The caller will use the socket for a passive open.
AI_CANONNAME	Tells the function to return the canonical name of the host.
AI_NUMERICHOST	Prevents any kind of name-to-address mapping; the *hostname* argument must be an address string.
AI_NUMERICSERV	Prevents any kind of name-to-service mapping; the *service* argument must be a decimal port number string.
AI_V4MAPPED	If specified along with an `ai_family` of `AF_INET6`, then returns IPv4-mapped IPv6 addresses corresponding to A records if there are no available AAAA records.
AI_ALL	If specified along with `AI_V4MAPPED`, then returns IPv4-mapped IPv6 addresses in addition to any AAAA records belonging to the name.
AI_ADDRCONFIG	Only looks up addresses for a given IP version if there is one or more interface that is not a loopback interface configured with an IP address of that version.

- If the **hints** argument is a null pointer, the function assumes a value of 0 for ai_flags, ai_socktype, and ai_protocol, and a value of AF_UNSPEC for ai_family.
- If the function returns success (0), the variable pointed to by the result argument is filled in with a pointer to a linked list of **addrinfo** structures, linked through the **ai_next** pointer.
- There are two ways that multiple structures can be returned:
  - If there are multiple addresses associated with the hostname, one structure is returned for each address that is usable with the requested address family (the ai_family hint, if specified).
  - If the service is provided for multiple socket types, one structure can be returned for each socket type, depending on the ai_socktype hint.

- For example, if no hints are provided and if the domain service is looked up for a host with two IP addresses, four addrinfo structures are returned:
    - One for the first IP address and a socket type of SOCK_STREAM
    - One for the first IP address and a socket type of SOCK_DGRAM
    - One for the second IP address and a socket type of SOCK_STREAM
    - One for the second IP address and a socket type of SOCK_DGRAM


res:
addrinfo{}
ai_flags
ai_family
AF_INET
ai_socktype
SOCK_DGRAM
ai_protocol
IPPROTO_UDP
ai_addrlen
16
ai_canonname
freebsd4.unpbook.com\0
ai_addr
sockaddr_in{}
16, AF_INET, 53
135.197.17.100
ai_next
addrinfo{}
ai_flags
ai_family
AF_INET
ai_socktype
SOCK_DGRAM
ai_protocol
IPPROTO_UDP
ai_addrlen
16
ai_canonname
NULL
ai_addr
sockaddr_in{}
16, AF_INET, 53
172.24.37.94
ai_next
addrinfo{}
ai_flags
ai_family
AF_INET
ai_socktype
SOCK_STREAM
ai_protocol
IPPROTO_TCP
ai_addrlen
16
ai_canonname
NULL
ai_addr
sockaddr_in{}
16, AF_INET, 53
135.197.17.100
ai_next
addrinfo{}
ai_flags
ai_family
AF_INET
ai_socktype
SOCK_STREAM
ai_protocol
IPPROTO_TCP
ai_addrlen
16
ai_canonname
NULL
ai_addr
sockaddr_in{}
16, AF_INET, 53
172.24.37.94
ai_next
NULL
dynamically allocated by getaddrinfo

- Not guaranteed that an implementation should return the IP addresses in the same order as they are returned by the DNS.
- Some resolvers allow the administrator to specify an address sorting order in the /etc/resolv.conf file- which could affect the order of addresses returned by getaddrinfo.
- The information returned in the addrinfo structures is ready for a call to socket and then either a call to connect or sendto (for a client), or bind (for a server).
- The arguments to socket are the members ai_family, ai_socktype, and ai_protocol.
- The second and third arguments to either connect or bind are ai_addr (a pointer to a socket address structure of the appropriate type, filled in by getaddrinfo) and ai_addrlen (the length of this socket address structure).

- If we were to enumerate all 64 possible inputs to getaddrinfo (there are six input variables), many would be invalid and some would make little sense.

1. Specify the hostname and service. This is normal for a TCP or UDP client. On return, a TCP client loops through all returned IP addresses, calling socket and connect for each one, until the connection succeeds or until all addresses have been tried.
2. For a UDP client, the socket address structure filled in by getaddrinfo would be used in a call to sendto or connect. If the client can tell that the first address doesn't appear to work (either by receiving an error on a connected UDP socket or by experiencing a timeout on an unconnected socket), additional addresses can be tried.
3. If the client knows it handles only one type of socket (e.g., Telnet and FTP clients handle only TCP; TFTP clients handle only UDP), then the ai_socktype member of the hints structure should be specified as either SOCK_STREAM or SOCK_DGRAM.

4. A typical server specifies the service but not the hostname, and specifies the AI_PASSIVE flag in the hints structure. The socket address structures returned should contain an IP address of INADDR_ANY (for IPv4) or IN6ADDR_ANY_INIT (for IPv6). A TCP server then calls socket, bind, and listen. If the server wants to malloc another socket address structure to obtain the client's address from accept, the returned ai_addrlen value specifies this size.
5. A UDP server would call socket, bind, and then recvfrom. If the server wants to malloc another socket address structure to obtain the client's address from recvfrom, the returned ai_addrlen value specifies this size.
6. As with the typical client code, if the server knows it only handles one type of socket, the ai_socktype member of the hints structure should be set to either SOCK_STREAM or SOCK_DGRAM. This avoids having multiple structures returned, possibly with the wrong ai_socktype value.

7. The TCP servers that we have shown so far create one listening socket, and the UDP servers create one datagram socket. That is what we assume in the previous item. An alternate server design is for the server to handle multiple sockets using select or poll. In this scenario, the server would go through the entire list of structures returned by getaddrinfo, create one socket per structure, and use select or poll.

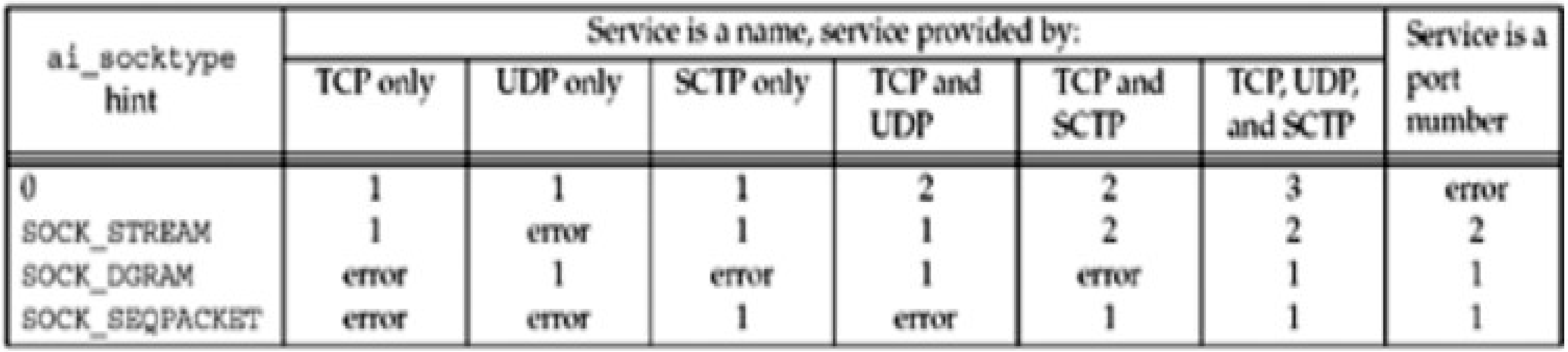
- The problem with this technique is that one reason for getaddrinfo returning multiple structures is when a service can be handled by IPv4 and IPv6.

- getaddrinfo is "better" than the gethostbyname and getservbynamefunctions
  - It makes it easier to write protocol-independent code;
  - One function handles both the hostname and the service and all the returned information is dynamically allocated, not statically allocated.

Disadvantages:

- Allocate a hints structure, initialize it to 0, fill in the desired fields, call getaddrinfo, and then traverse a linked list trying each one.

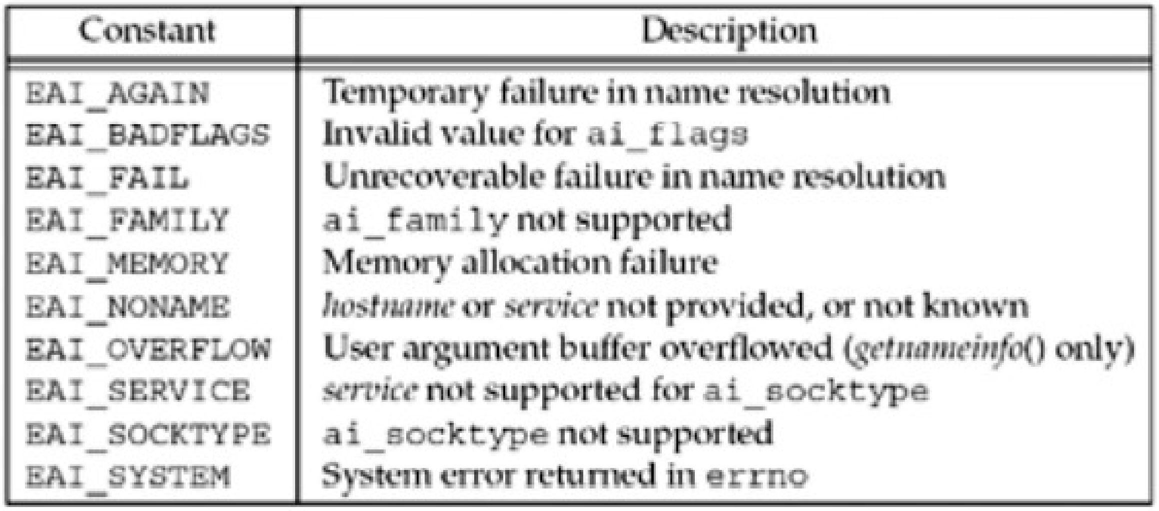
Number of addrinfo structures returned per IP address.



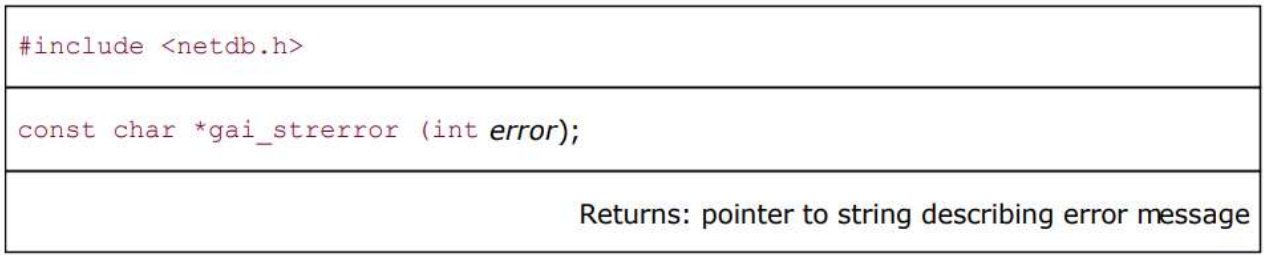
ai_socktype hint	Service is a name, service provided by:						Service is a port number
	TCP only	UDP only	SCTP only	TCP and UDP	TCP and SCTP	TCP, UDP, and SCTP	
0	1	1	1	2	2	3	error
SOCK_STREAM	1	error	1	1	2	2	2
SOCK_DGRAM	error	1	error	1	error	1	1
SOCK_SEQPACKET	error	error	1	error	1	1	1

gai_strerror Function

- The nonzero error return values from getaddrinfo have the names and meanings shown:



Constant	Description
EAI_AGAIN	Temporary failure in name resolution
EAI_BADFLAGS	Invalid value for ai_flags
EAI_FAIL	Unrecoverable failure in name resolution
EAI_FAMILY	ai_family not supported
EAI_MEMORY	Memory allocation failure
EAI_NONAME	*hostname* or *service* not provided, or not known
EAI_OVERFLOW	User argument buffer overflowed (*getnameinfo*() only)
EAI_SERVICE	*service* not supported for ai_socktype
EAI_SOCKTYPE	ai_socktype not supported
EAI_SYSTEM	System error returned in errno



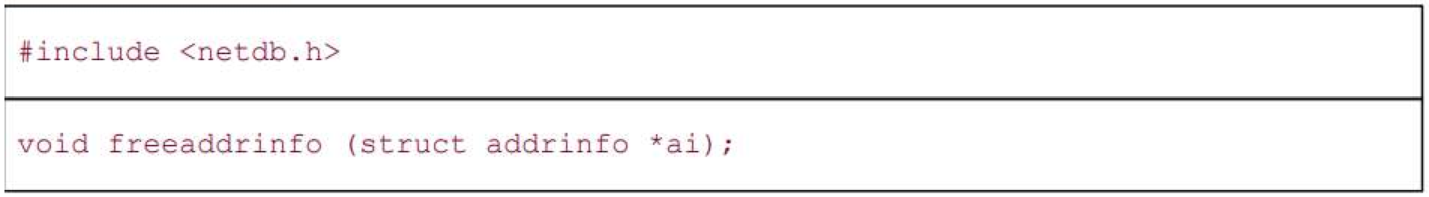
```
#include <netdb.h>
```

```
const char *gai_strerror (int error);
```

Returns: pointer to string describing error message

freeaddrinfo Function

- All the storage returned by getaddrinfo, the addrinfo structures, the ai_addr structures, and the ai_canonname string are obtained dynamically (e.g., from malloc).
- This storage is returned by calling freeaddrinfo.



```
#include <netdb.h>
```

```
void freeaddrinfo (struct addrinfo *ai);
```

- ai should point to the first addrinfo structure returned by getaddrinfo.
- All the structures in the linked list are freed, along with any dynamic storage pointed to by those structures

Note:

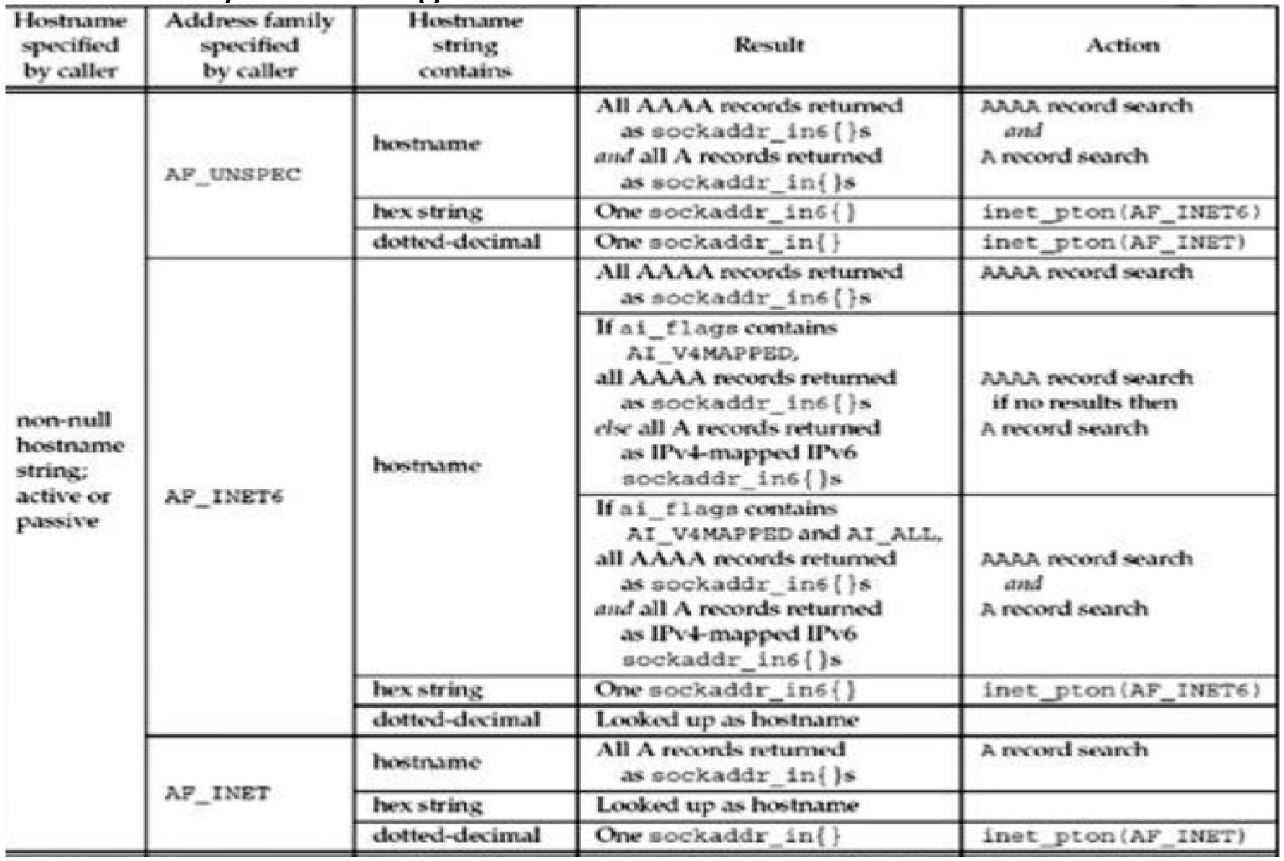
1. Making a copy of just the addrinfo structure and not the structures that it in turn points to is called a shallow copy.
2. Copying the addrinfo structure and all the structures that it points to is called a deep copy.

RV College of Engineering

Go, change the world

getaddrinfo Function: IPv6

- Summary of how getaddrinfo handles IPv4 and IPv6 addresses. .



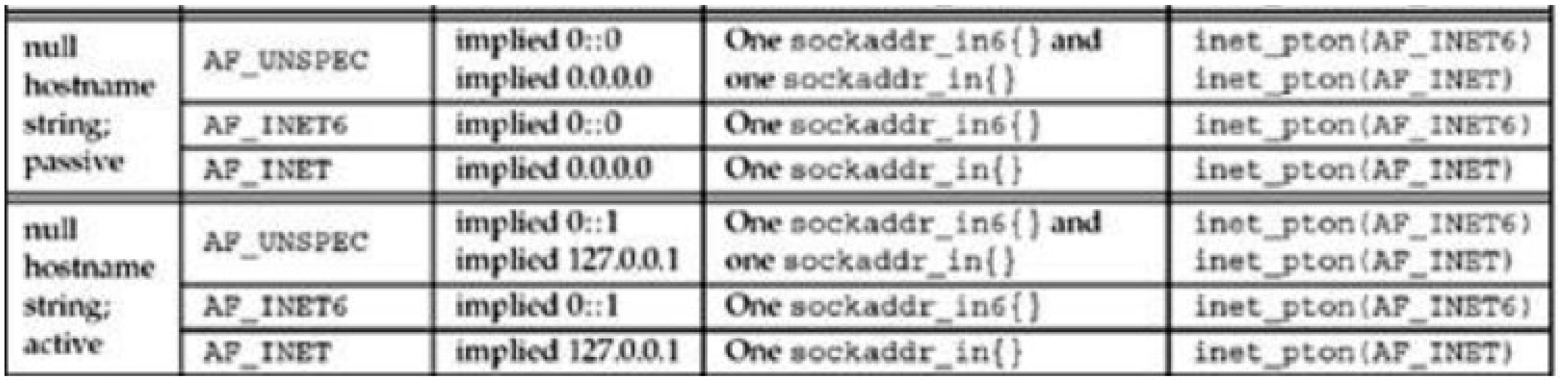
Hostname specified by caller	Address family specified by caller	Hostname string contains	Result	Action
non-null hostname string; active or passive	AF_UNSPEC	hostname	All AAAA records returned as sockaddr_in6{}s *and* all A records returned as sockaddr_in{}s	AAAA record search *and* A record search
		hex string	One sockaddr_in6{}	inet_pton(AF_INET6)
		dotted-decimal	One sockaddr_in{}	inet_pton(AF_INET)
	AF_INET6	hostname	All AAAA records returned as sockaddr_in6{}s	AAAA record search
			If ai_flags contains AI_V4MAPPED, all AAAA records returned as sockaddr_in6{}s *else* all A records returned as IPv4-mapped IPv6 sockaddr_in6{}s	AAAA record search if no results then A record search
			If ai_flags contains AI_V4MAPPED and AI_ALL, all AAAA records returned as sockaddr_in6{}s *and* all A records returned as IPv4-mapped IPv6 sockaddr_in6{}s	AAAA record search *and* A record search
		hex string	One sockaddr_in6{}	inet_pton(AF_INET6)
		dotted-decimal	Looked up as hostname	
	AF_INET	hostname	All A records returned as sockaddr_in{}s	A record search
		hex string	Looked up as hostname	
		dotted-decimal	One sockaddr_in{}	inet_pton(AF_INET)

- The "Result" column is what is to be returned to the caller, given the variables in the first three columns.
- The "Action" column is how the function obtains this result.

RV College of Engineering

Go, change the world

getaddrinfo Function: IPv6



null hostname string; passive	AF_UNSPEC	implied 0::0
		implied 0.0.0.0
			One sockaddr_in6{} and one sockaddr_in{}	inet_pton(AF_INET6)
				inet_pton(AF_INET)

	AF_INET6	implied 0::0	One sockaddr_in6{}	inet_pton(AF_INET6)
	AF_INET	implied 0.0.0.0	One sockaddr_in{}	inet_pton(AF_INET)

null hostname string; active	AF_UNSPEC	implied 0::1
		implied 127.0.0.1
			One sockaddr_in6{} and one sockaddr_in{}	inet_pton(AF_INET6)
				inet_pton(AF_INET)

	AF_INET6	implied 0::1	One sockaddr_in6{}	inet_pton(AF_INET6)
	AF_INET	implied 127.0.0.1	One sockaddr_in{}	inet_pton(AF_INET)

Remember:

- getaddrinfo is dealing with two different inputs
  1. The type of socket address structure the caller wants back.
  2. The type of records that should be searched for in the DNS or other database.
- The address family in the hints structure provided by the caller specifies the type of socket address structure that the caller expects to be returned.
- specifying AF_UNSPEC will return addresses that can be used with any protocol family that can be used with the hostname and service name.
  - This implies that if a host has both AAAA records and A records, the AAAA records are returned as sockaddr_in6 structures and the A records are returned as sockaddr_in structures.

Remember:

- If the AI_PASSIVE flag is specified without a hostname, then the IPv6 wildcard address (IN6ADDR_ANY_INIT or 0::0) should be returned as a sockaddr_in6 structure, along with the IPv4 wildcard address (INADDR_ANY or 0.0.0.0), which is returned as a sockaddr_in structure.
- The address family specified in the hint structure's ai_family member, along with the flags such as AI_V4MAPPED and AI_ALL specified in the ai_flags member, dictate the type of records that are searched for in the DNS (A and/or AAAA) and what type of addresses are returned (IPv4, IPv6, and/or IPv4-mapped IPv6).

Remember:

- The hostname can also be either an IPv6 hex string or an IPv4 dotted-decimal string. The validity of this string depends on the address family specified by the caller. (AF_INET for IPV4, AF_INET6 for IPV6 and AF_UNSPEC for both)