Information Technology | **IPv6 PMO**

Overview: Application Development for IPv6

Application Development – Design Impact

Multiple Address Families

- By deployment of Internet Protocol Version 6 (IPv6), the application programmer has to cope with socket connection with multiple address families, i.e. AF_INET and AF_INET6
- How does a programmer handle multiple address families with ease?

Address Family Independ. Application

- Allows for maximum portability
- All (most) address family dependent stuff is removed from the application by using these functions:
  - getnameinfo
  - getaddrinfo
- Your code doesn’t need to care if the calling host is IPv4 or IPv6

IRS Considerations for IP Independent Apps

- In order to allow applications to communicate with other IPv6 nodes, the first priority is to convert the applications supporting both IPv4 and IPv6
  - IP version-independent structures & APIs
  - Must iterate over all addresses returned to choose best address
- The applications have to work properly in dual-stacked nodes (where both IPv4 and IPv6 addresses are configured)
- The applications have to work properly in IPv4-only nodes (where no IPv6 address is configured)
- The applications have to work properly in IPv6-only nodes (where no IPv4 address is configured)

Coding Areas Requiring Changes

- Socket structures
  - Use “generic” socket structure
  - Avoid IP specific structures
- Name to Address Translation functions

sockaddr_storage Structure

New Socket structure

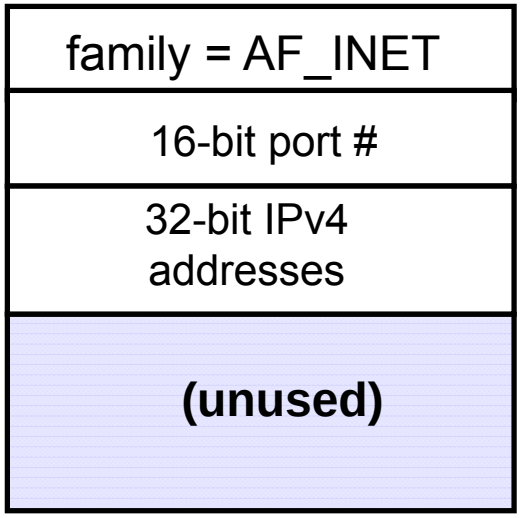
```
struct sockaddr_storage
{
    sa_family_t ss_family;          /*Address family */
   __ss_aligntype __ss_align;  /*Force desired alignment*/
    char __ss_padding[__ss_PADSIZE];
};
```

- Use when you need to allocate space for a socket structure
  - Address family independent: can accommodate IPv4, IPv6, or other AF

Comparison of Structures

IPv4

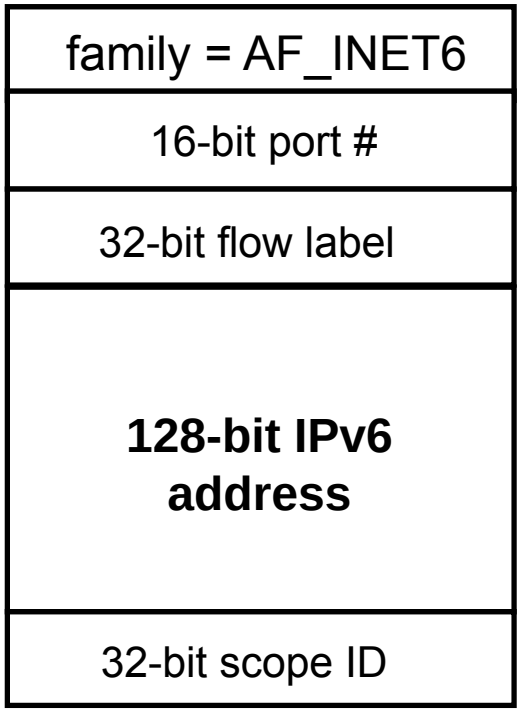
sockaddr_in{}



Fixed length – 16 bytes

IPv6

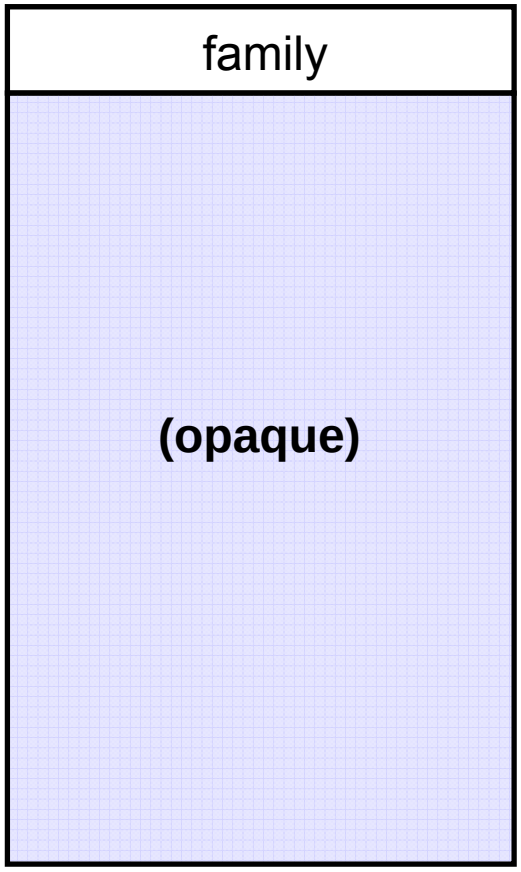
sockaddr_in6{}



Fixed length – 28 bytes

storage

sockaddr_storage{}



Longest on system

Examples of sockaddr_storage Usage

```
/* IPv4/IPv6 Support code */
struct sockaddr_storage addr;
socklen_t addrlen;
bind(sockfd, (struct sockaddr *)&addr, addrlen)
```

```
/* IPv4/IPv6 Support code */
struct sockaddr_storage addr;
socklen_t addrlen;
accept(sockfd, (struct sockaddr *)&addr, &addrlen)
```

sockaddr_storage to Store Arbitrary Addresses

Usage example 1

```
struct sockaddr_storage ss;
 ...
if (logging) {
     sval = sizeof(sockaddr_storage);
     if (getpeername(0, (struct sockaddr *)&ss, &sval) < 0)
          err("getpeername: %s", strerror(errno));
```

Usage example 2

```
struct sockaddr_storage  ss;
struct sockaddr_in6 *sin6;
…
     sin6 = (struct sockaddr_in6 *) &ss;
```

Structure Allocation & Casting

Structure Allocation

```
struct sockaddr_in svrAddr4;      /* IPv4 */
struct sockaddr_in6 svrAddr6;     /* IPv6 */
struct sockaddr_storage svrAddr; /* both */
```

Socket Functions Require (struct sockaddr *)

```
/* IPv4 */
bind (serverfd, (struct sockaddr *)&svrAddr4, length);
/* IPv6 */
bind (serverfd, (struct sockaddr *)&svrAddr6, length);
/* Both*/
bind (serverfd, (struct sockaddr *)&svrAddr, length);
```

Coding Areas Requiring Changes

- Socket structures
- Name to Address Translation functions

Protocol-Independent Name to Address Functions

IPv4 only or Protocol specific	Protocol Independent

`gethostbyname()`
`getservbyname()`
`inet_aton()`
`inet_pton()`
	`getaddrinfo()`

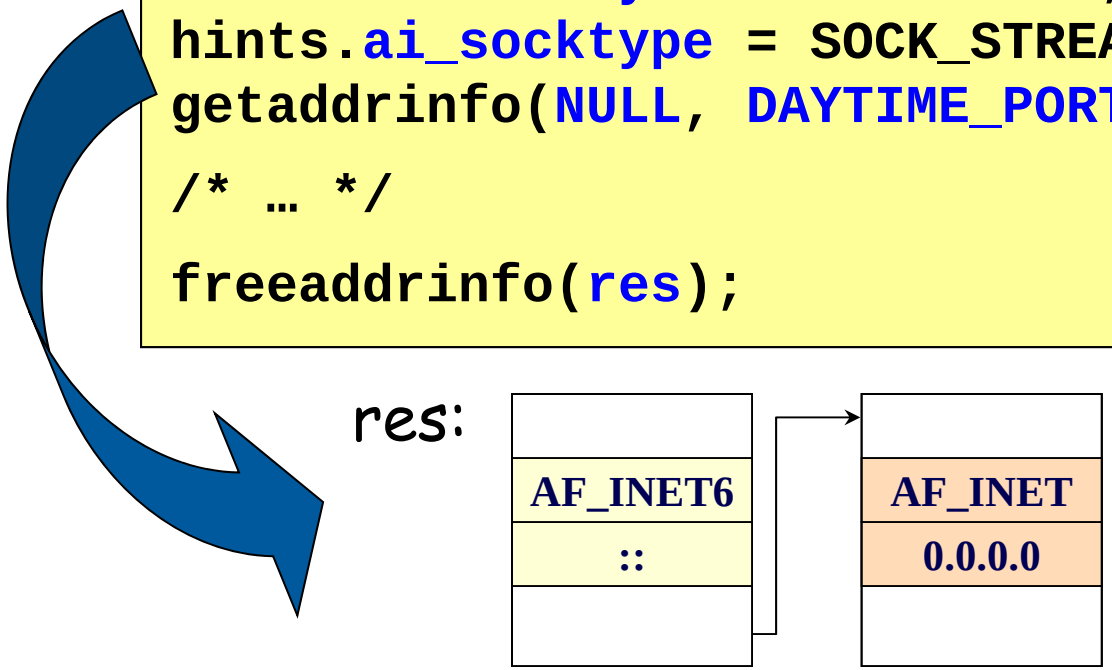
`gethostbyaddr()`
`getservbyport()`
`inet_ntoa()`
`inet_ntop()`
	`getnameinfo()`

Name and Address Translation

- getaddrinfo() returns a linked list of address info structures
  - Returns structure containing a socket structure
- Some hints about the desired kinds of addresses may be given in the hints argument
- getaddrinfo dynamically allocates memory unlike its predecessor functions
- The corresponding freeaddrinfo() function is called to free the buffers that are allocated by getaddrinfo().

getaddrinfo()

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags    = AI_PASSIVE;
hints.ai_family   = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM; /* SOCK_DGRAM */
getaddrinfo(NULL, DAYTIME_PORT, &hints, &res);
/* … */
freeaddrinfo(res);
```



getnameinfo()

```
struct sockaddr_storage clientAddr;
char clientHost[ADDRLEN];
char clientPort[PORTLEN];
/* … */
connectedfd = accept(serverfd,
                     (struct sockaddr *)&clientAddr,
                     &alen);
getnameinfo((struct sockaddr *)&clientAddr, addrLen,
            clientHost, sizeof(clientHost),
             clientPort, sizeof(clientPort),
             NI_NUMERICHOST);

printf("Request from host=[%s] port=[%s]\n",
       clienthost, clientservice);
```

Rules of Thumb

- Do not hardcode knowledge about particular AF
- Use getaddrinfo() and getnameinfo() everywhere
- Avoid struct in_addr and struct in6_addr

IRS Do Not Hardcode Knowledge About AF

The construct like the one below should be avoided:

```
/* BAD EXAMPLE */

switch (sa->sa_family) {

    case AF_INET:
        salen = sizeof(struct sockaddr_in);
        break;
…
}
```

Instead, use res->ai_addrlen returned by getaddrinfo

Example of Protocol Independent Code

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

main()  {
  struct addrinfo hints, *res, *res0;
  int     error;
  int     s;
  const char *cause =    NULL;


  memset(&hints, 0, sizeof(hints));
  hints.ai_family =         AF_UNSPEC;
  hints.ai_socktype =    SOCK_STREAM;
  error = getaddrinfo("www.company.example", "ftp",
           &hints, &res0);
  if(error) {
     fprintf(stderr, "%s", gai_strerror(error));
     exit(1);
  }
```

Ex. of Protocol Independent Code (2)

```
s =  -1;
for    (res = res0; res; res =    res->ai_next) {
   s = socket(res->ai_family, res->ai_socktype,
        res->ai_protocol);
   if (s < 0) {
    cause = "Error: socket";
    continue;
   }
   if (connect(s, res->ai_addr, res->ai_addrlen) < 0) {
    cause = "Error: connect";
    close(s);
    s =       -1;
    continue;
   }
   cause = "Success";
   break; /* okay we got one */                    …
```

getaddrinfo: struct addrinfo

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

```
struct addrinfo {
   int         ai_flags;        /* AI_PASSIVE, … */
   int         ai_family;       /* AF_xxx */
   int         ai_socktype;     /* SOCK_xxx */
   int         ai_protocol;     /* 0 or IPPROTO_xxx */
   socklen_t   ai_addrlen;      /* length of ai_addr */
   char       *ai_canonname;    /* canonical name */
   struct sockaddr  *ai_addr;   /* binary address */
   struct addrinfo  *ai_next;   /* next struct in list */
};
```

getaddrinfo: hints

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

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

FLAGS

AI_PASSIVE
AI_CANONNAME
AI_NUMERICHOST
AI_NUMERICSERV
AI_V4MAPPED
AI_ALL
AI_ADDRCONFIG

Hints for getaddrinfo()

- ai_flags (zero or more AI_xxx values OR'ed)
  - AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, AI_ADDRCONFIG
- ai_family (an AF_xxx value)
  - AF_INET, AF_INET6, AF_UNSPEC
- ai_socktype (a SOCK_xxx value)
  - SOCK_STREAM, SOCK_DGRAM
- ai_protocol

Flags for getaddrinfo

- **AI_PASSIVE** – Used for socket passive open (i.e., server socket)
- **AI_CANONNAME** – Return canonical name of the host
- **AI_NUMERICHOST** – Prevents any kind of *name-to-address mapping*; the hostname 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

Flags for getaddrinfo (2)

- **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

Multiple addrinfo Returned

- There are two ways that multiple addrinfo structures can be returned:

    1. If there are multiple addresses associated with the hostname, one structure is returned for each address that is usable with the requested address family (*ai_family* hint, if specified)

    2. If the service is provided for multiple socket types, one structure can be returned for each socket type, depending on the *ai_socktype* hint

Error Return for getaddrinfo

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 support
EAI_MEMORY	Memory Allocation failure
EAI_NONAME	*hostname* or *service* not provided, or not known
EAI_OVERFLOW	User argument buffer overflow
EAI_SERVICE	Service not supported for ai_socktype
EAI_SOCKTYPE	ai_socktype not supported
EAI_SYSTEM	System error returned in errno

Nonzero error return constants from getaddrinfo

Ex 1: *addrinfo* Returned by *getaddrinfo*

DNS

```
bsdi  IN  A    206.62.226.35
      IN  A    206.62.226.66
```

/etc/services

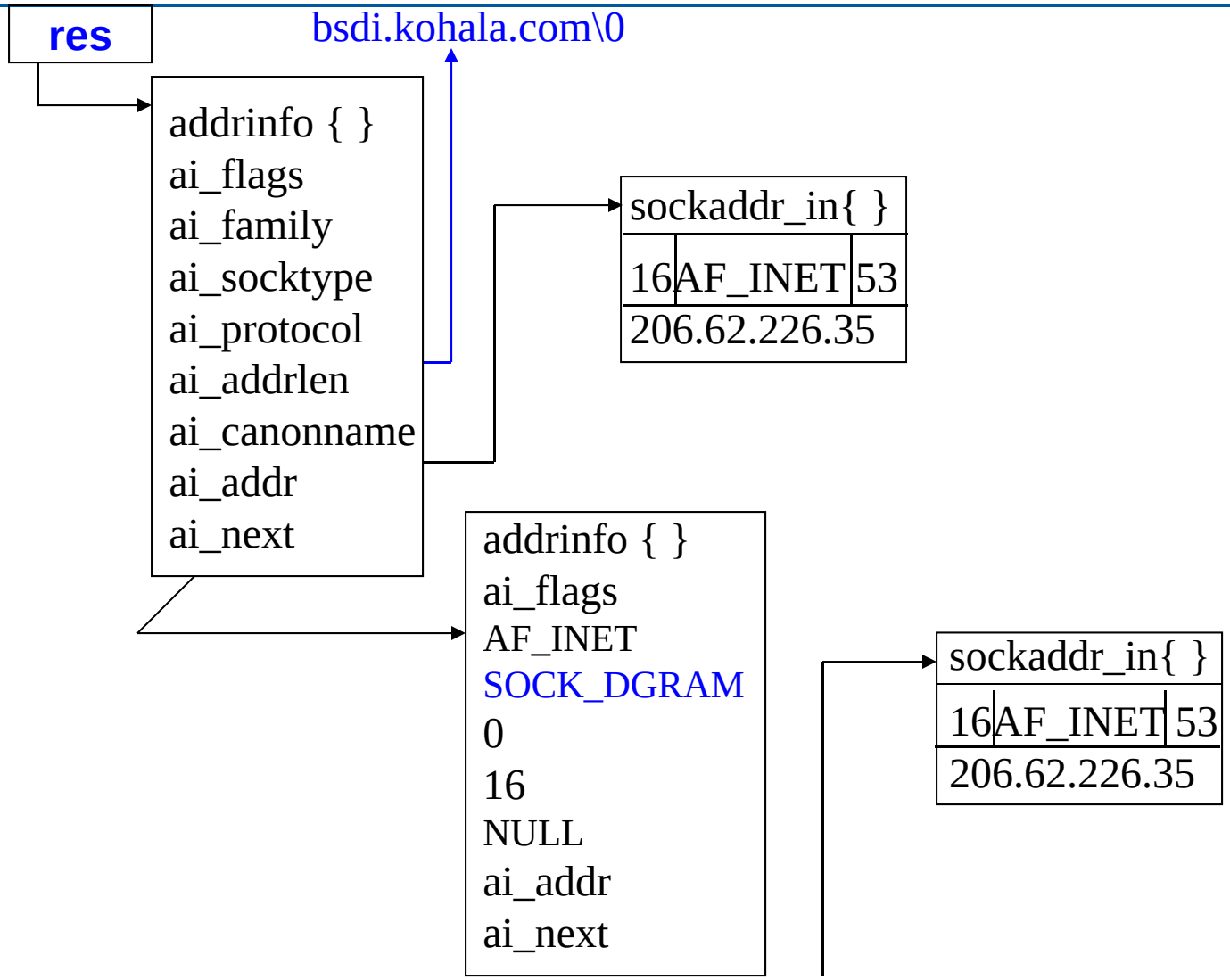
```
domain 53/tcp
domain 53/udp
```

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);

getaddrinfo(“bsdi”, “domain”, &hints, &res);

/* … */

freeaddrinfo(res);
```

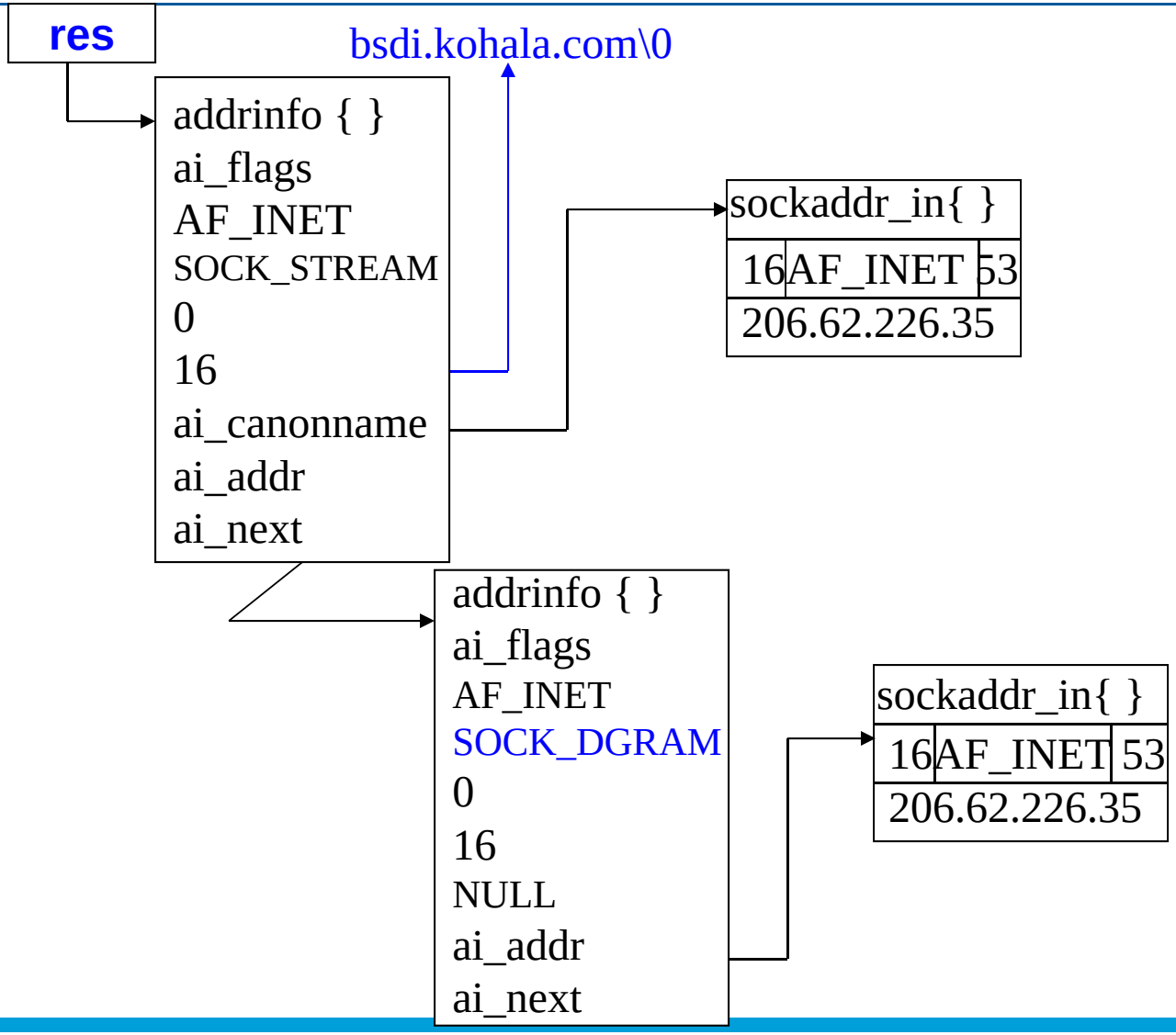
Ex 1: *addrinfo* Returned by *getaddrinfo* (2)

- 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 SOCKET_STREAM
- One for the second IP address and a socket type of SOCKET_DGRAM

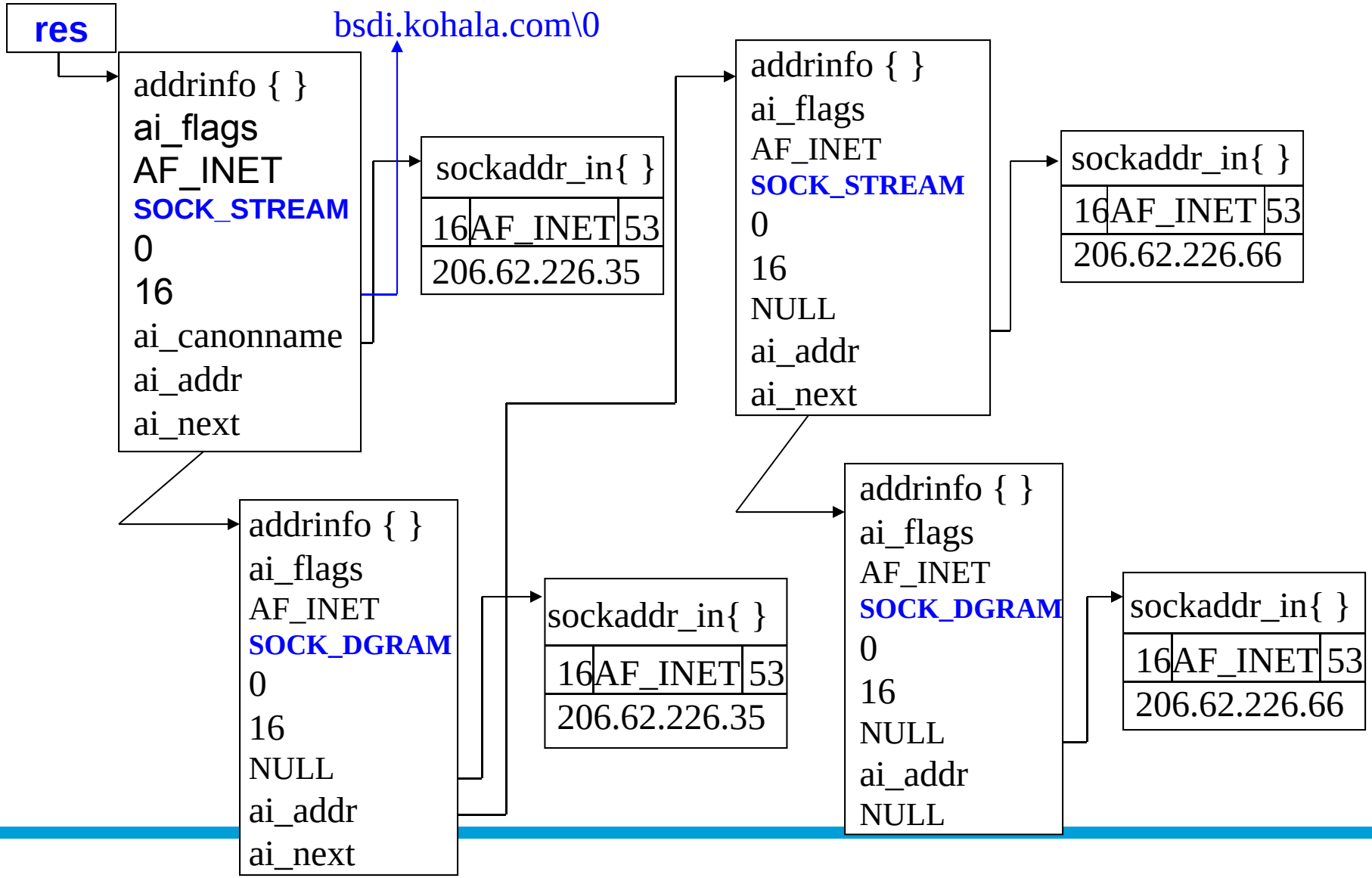
Ex 1: *addrinfo* Returned by *getaddrinfo* (3)



Ex 1: *addrinfo* Returned by *getaddrinfo* (4)



Ex 1: *addrinfo* Returned by *getaddrinfo* (5)



IRS *getaddrinfo* Ex 2: IPv4 and IPv6 Addresses

DNS

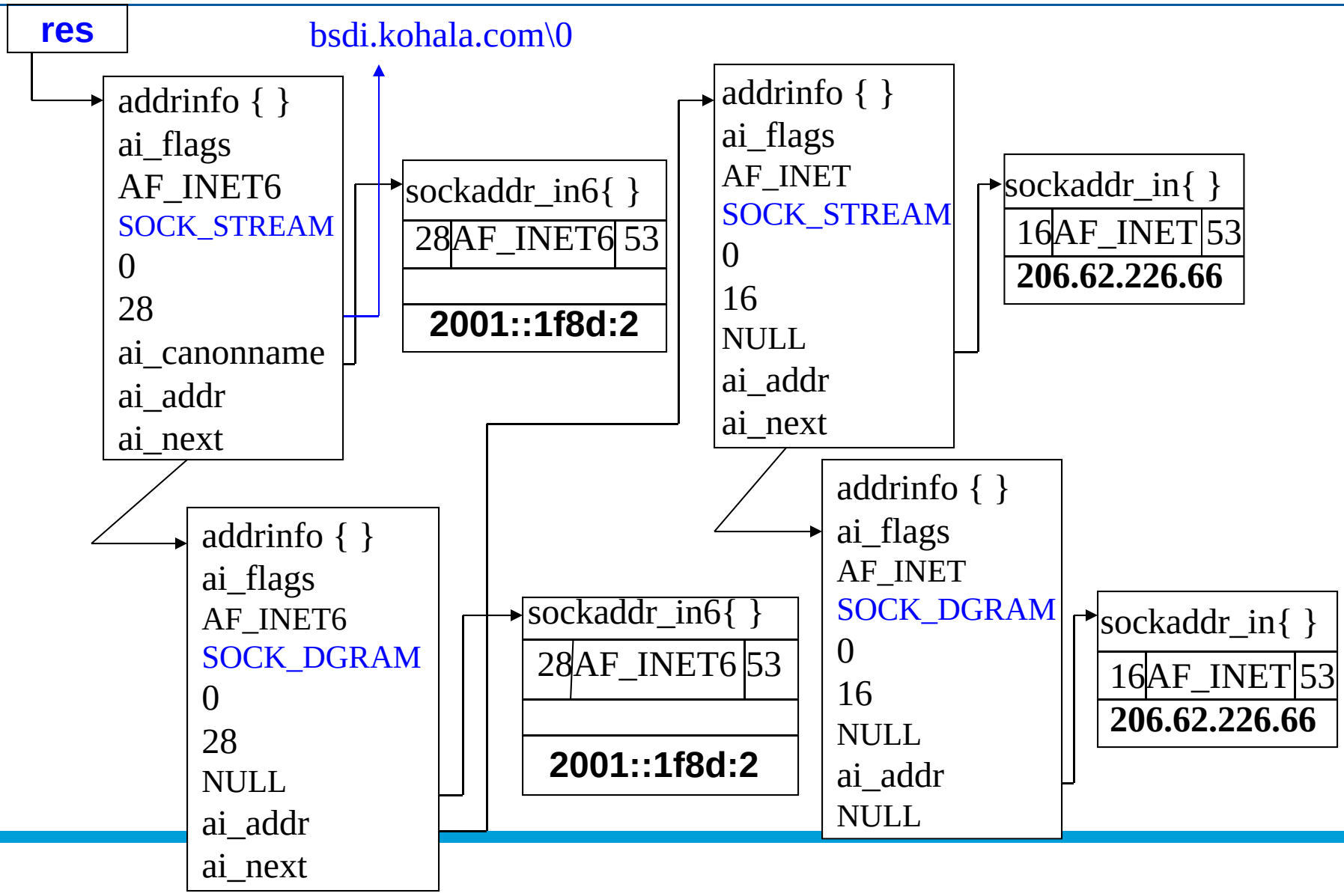
```
bsdi    IN  A       206.62.226.66
        IN  AAAA    2001::1f8d:2
```

/etc/services

```
domain 53/tcp
domain 53/udp
```

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);

getaddrinfo("bsdi", "domain", &hints, &res);

/* … */

freeaddrinfo(res);
```

Ex 2: IPv4 and IPv6 Addresses *(2)*



getaddrinfo Ex 3: IPv4-mapped

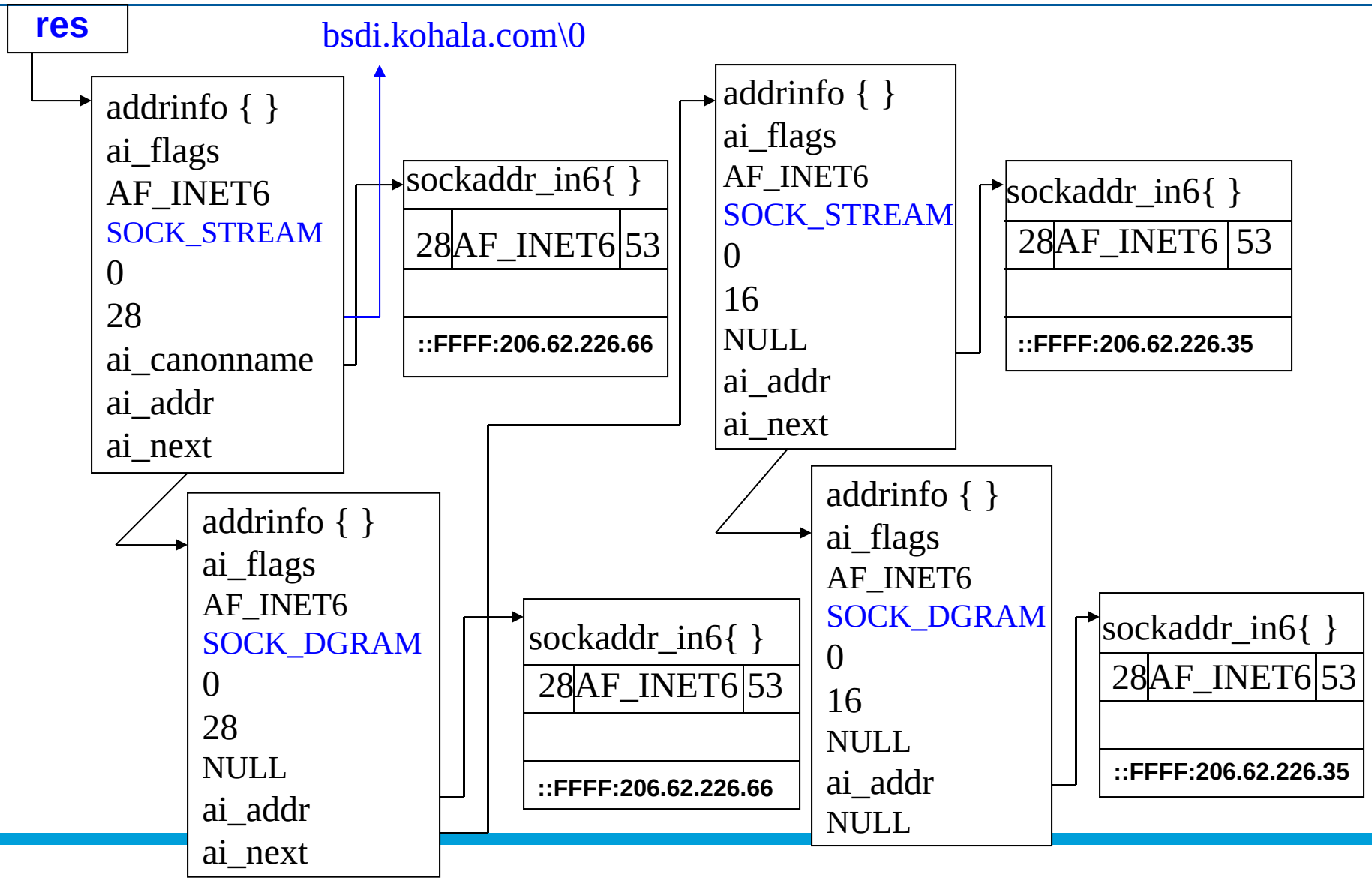
DNS

/etc/services

```
bsdi  IN   A          206.62.226.66
      IN   A     206.62.226.35      domain 53/tcp
                                    domain 53/udp
```

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags = AI_V4MAPPED;
hints.ai_family = AF_INET6;
getaddrinfo("bsdi", "domain", &hints, &res);

/* … */

freeaddrinfo(res);
```

Ex 3: IPv4-mapped (2)



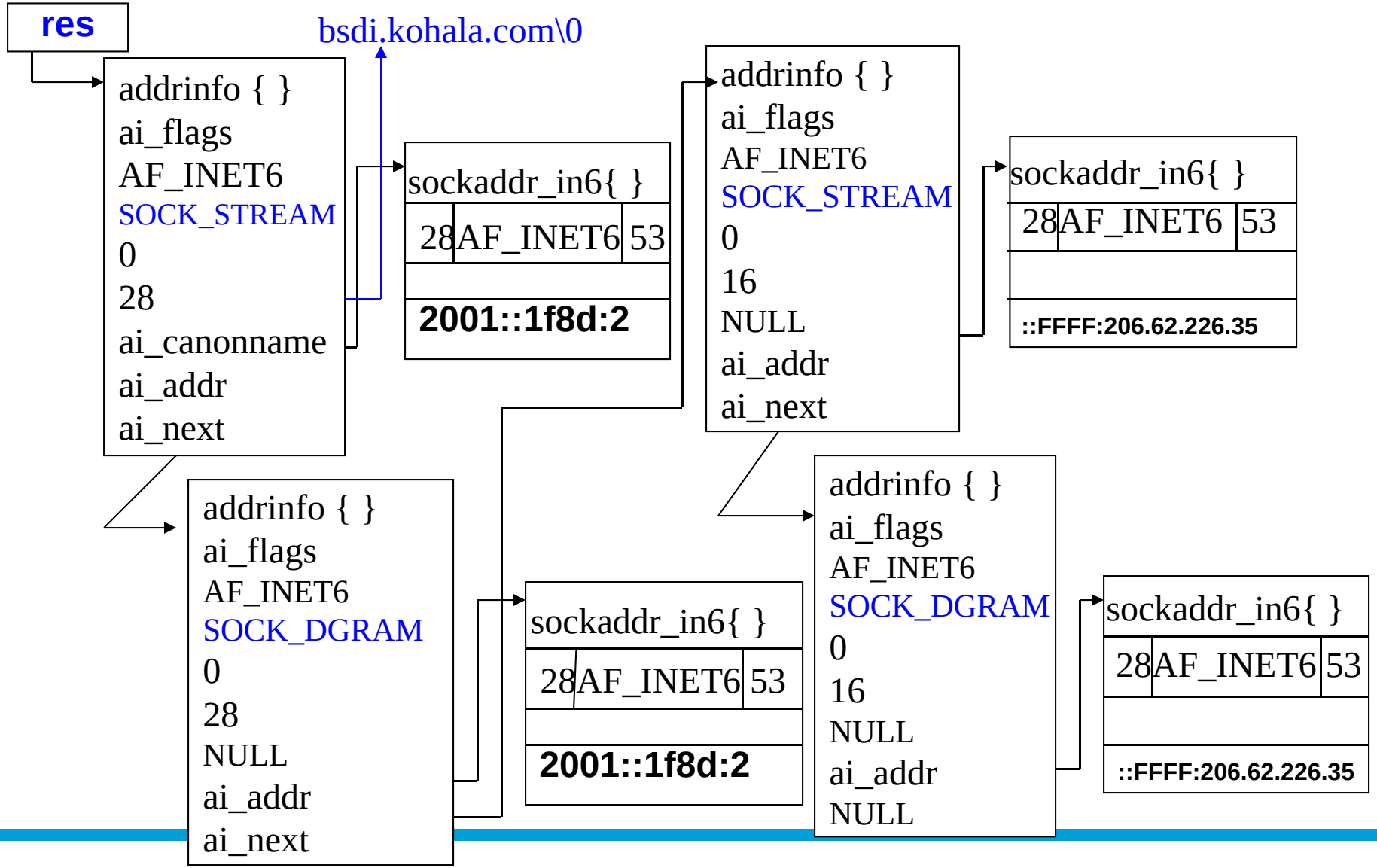
DNS

/etc/services

```
bsdi IN   A       206.62.226.35
     IN   AAAA    2001::1f8d:2      domain 53/tcp
                                    domain 53/udp
```

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags = AI_ALL | AI_V4MAPPED;
hints.ai_family = AF_INET6;
getaddrinfo("bsdi", "domain", &hints, &res);

/* … */

freeaddrinfo(res);
```

Ex 4: IPv4-mapped with IPv6 Addresses (2)



getaddrinfo Ex 5: AI_ADDRCONFIG

DNS

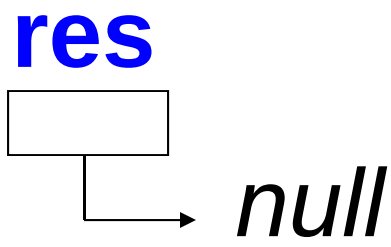
				/etc/services
bsdi	IN	A	206.62.226.35	domain 53/tcp
	IN	AAAA	2001::1f8d:2	domain 53/udp

Client making getaddrinfo() call has no interface configured with IPv6

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags = AI_ADDRCONFIG;
hints.ai_family = AF_INET6;
getaddrinfo("bsdi", "domain", &hints, &res);

/* … */

freeaddrinfo(res);
```

Ex 5: AI_ADDRCONFIG (2)

- If the node has no IPv6 source addresses configured, and *af* "hint" equals AF_INET6 , and the node name being looked up has both AAAA and A records, then
  - if only the AI_ADDRCONFIG "hint" is specified, the function returns a null pointer



IRS Ex 6: *addrinfo* Returned by *getaddrinfo*

DNS

bsdi	**IN**	**A**	**206.62.226.35**
	IN	**AAAA**	**2001::1f8d:2**

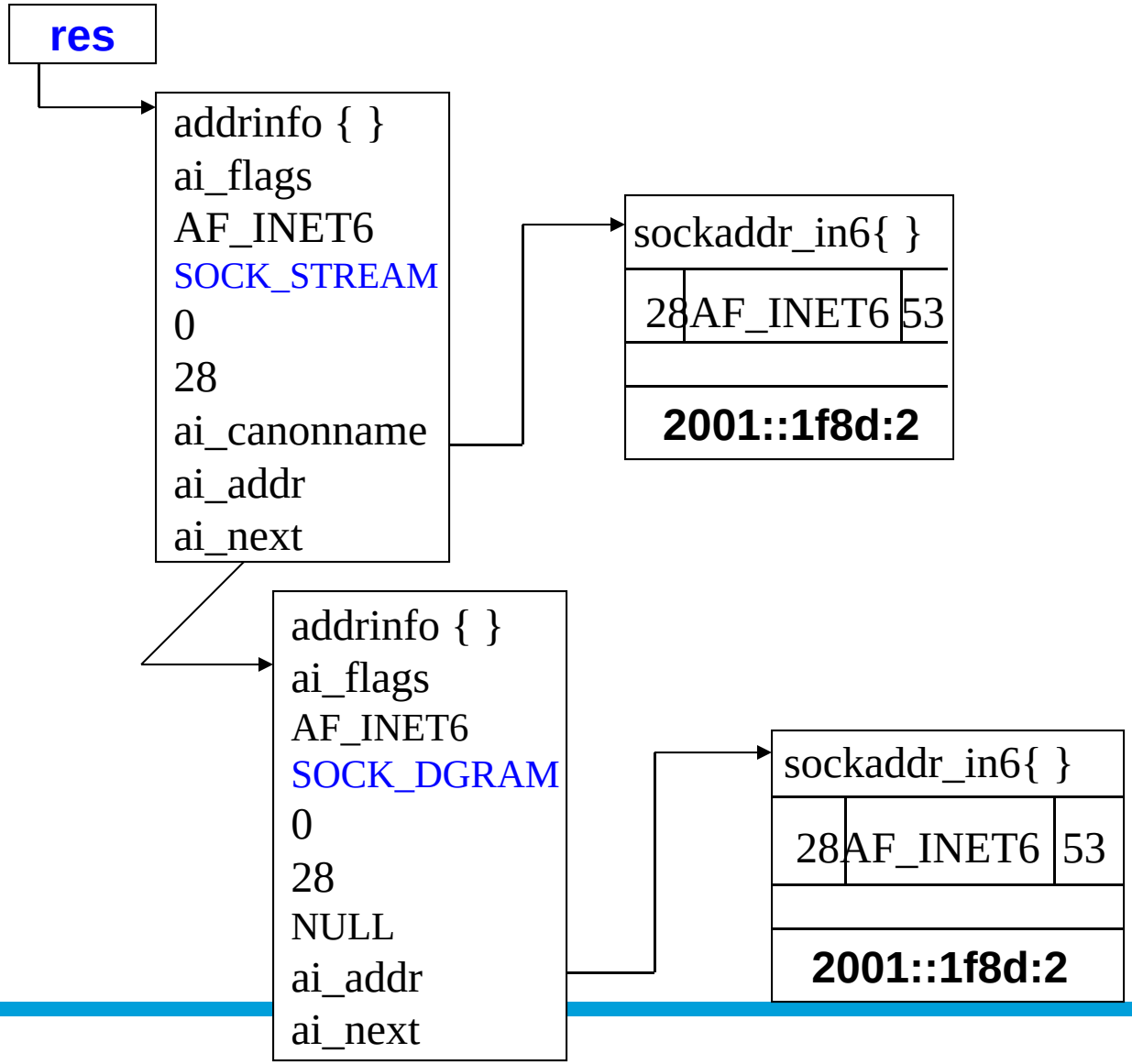
/etc/services

domain 53/tcp
domain 53/udp

Client making getaddrinfo() call has no interface configured with IPv6

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags = 0;
hints.ai_family = AF_INET6;
getaddrinfo("bsdi", "domain", &hints, &res);

/* … */

freeaddrinfo(res);
```

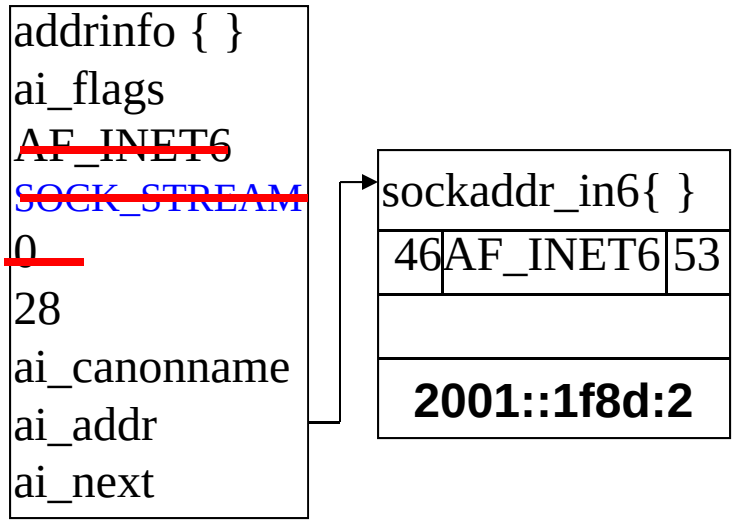
Ex 6: *addrinfo* Returned by *getaddrinfo* (2)



Ex 6: *addrinfo* Returned by *getaddrinfo* (3)

```
    . . .
s =  -1;
 for (res = res0; res; res = res->ai_next) {
   s = socket(res->ai_family, res->ai_socktype,
                  res->ai_protocol);
   if (s < 0) {
         cause = "Error: socket";
         continue;
   }
   if (connect(s, res->ai_addr, res->ai_addrlen) < 0) {
         cause = "Error: connect";
         close(s);
         s =       -1;
         continue;
   }
   cause = "Success";
   break; /* okay we got one */
   …
```

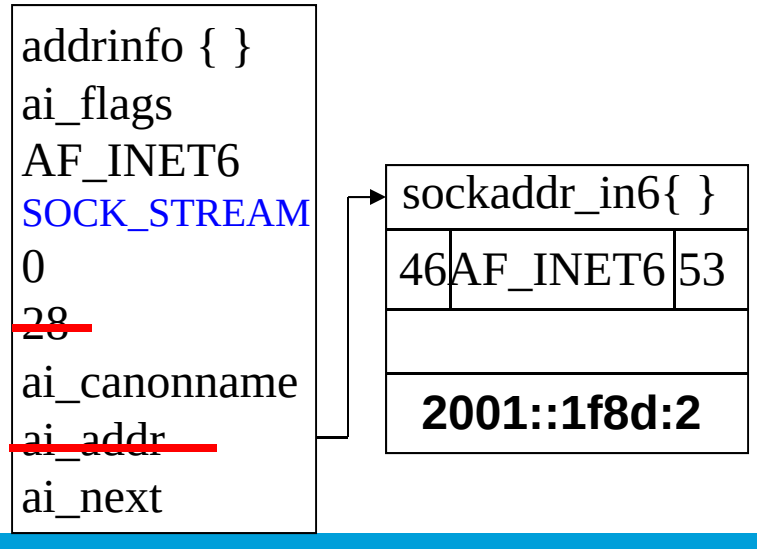
SUCCESS



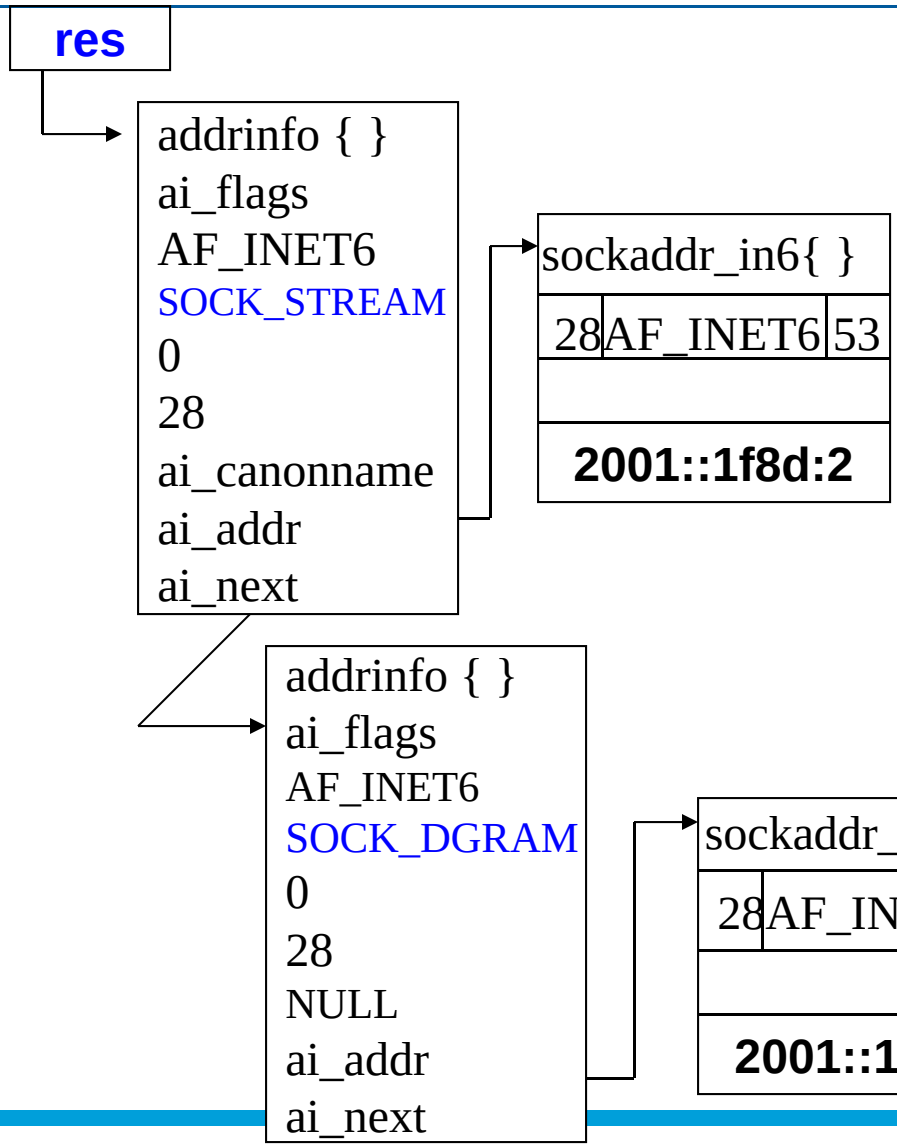
Ex 6: addrinfo Returned by getaddrinfo (4)

```
    . . .
s =  -1;
 for  (res = res0; res; res = res->ai_next) {
   s = socket(res->ai_family, res->ai_socktype,
                    res->ai_protocol);
   if (s < 0) {
         cause = "Error: socket";
         continue;
   }
   if (connect(s, res->ai_addr, res->ai_addrlen) < 0) {
         cause = "Error: connect";
         close(s);
         s =       -1;
         continue;
   }
   cause = "Success";
   break; /* okay we got one */
   …
```

Error: connect



Ex 6: *addrinfo* Returned by *getaddrinfo* (5)




```
%  nslookup
> set q=AAAA
name:  bsdi.kohala.com
address: 2001::1f8d:2
```

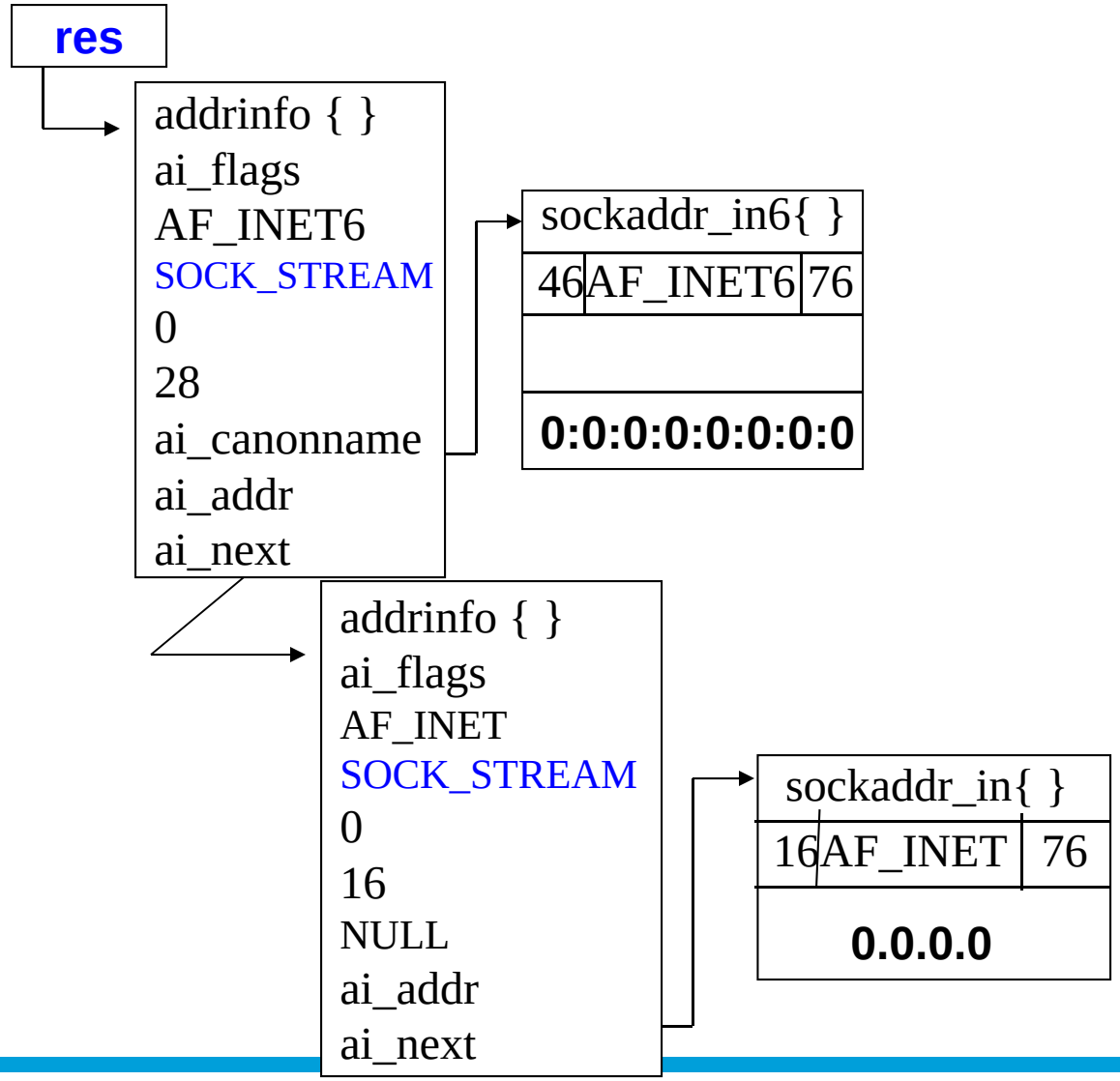
getaddrinfo Ex 7: Server Socket

/etc/services

myservice 76/tcp
myservice 76/udp

```
struct addrinfo hints;
struct addrinfo *res;
char *myservice;
…
memset(&hints, 0, sizeof(hints)); /* set-up hints
structure */
hints.ai_family = AF_UNSPEC;
hints.ai_flags = AI_PASSIVE;
hints.ai_socktype = SOCK_STREAM;
error = getaddrinfo(NULL, myservice, &hints, &res);
```

Ex 7: Server Socket (2)



getnameinfo()

```
int getnameinfo (const struct sockaddr *sockaddr,
                 socklen_t addrlen,
                 char *host, size_t hostlen,
                 char *serv, size_t servlen, int flags);
```

- Input parameters
  - sockaddr, addrlen
  - hostlen, servlen
  - flags
- Output
  - host, serv
  - error code

getnameinfo()

```
int getnameinfo (const struct sockaddr *sockaddr,
                 socklen_t addrlen,
                 char *host, size_t hostlen,
                 char *serv, size_t servlen, int flags);
```

FLAGS

NI_NOFQDN

NI_NUMERICHOST

NI_NAMEREQD

NI_NUMERICSERV

AI_DGRAM

Flags for getnameinfo()

Constant	Description
NI_DGRAM	Datagram service
NI_NAMEREQD	Return an error if name cannot be resolved from addr
NI_NOFQDN	Return only hostname portion of FQDN
NI_NUMERICHOST	Return numeric string for hostname
NI_NUMERICSERV	*Return numeric string for service name*

Flags for **getnameinfo()**

Socket Address to Name: Example

Getting IP address as string for the accepted connection:

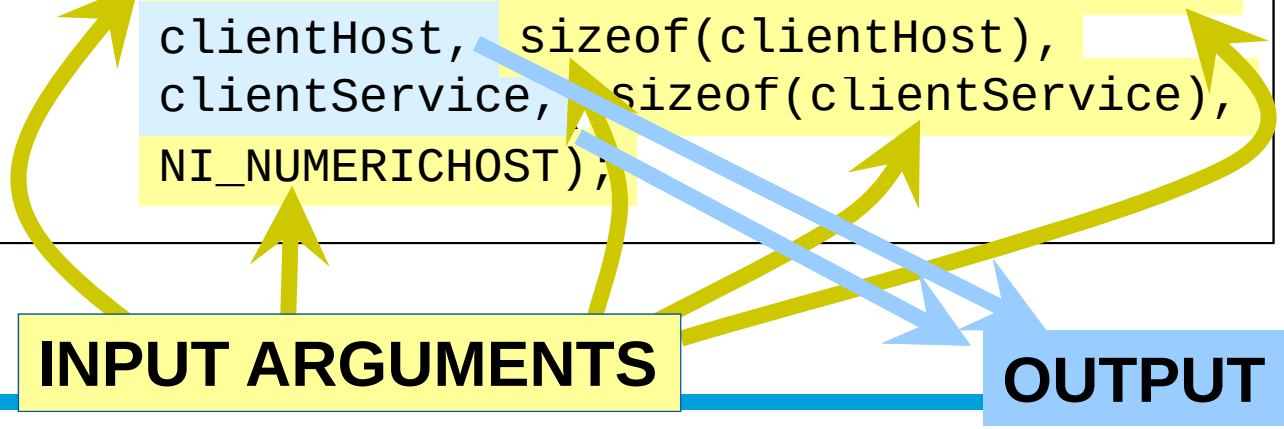
```
struct sockaddr_storage clientAddr;
char   clientHost[NI_MAXHOST];
char   clientService[NI_MAXSERV];
size_t alen = sizeof(struct sockaddr_storage);

/*… */

connectedfd = accept(serverfd,
                     (struct sockaddr *)&clientAddr,
                      &alen);
err =getnameinfo((struct sockaddr *)&clientAddr, alen,
                 clientHost, sizeof(clientHost),
                 clientService, sizeof(clientService),
                 NI_NUMERICHOST);
```

INPUT ARGUMENTS

Socket Address to Name: Example (2)

Getting IP address as string for the accepted connection:

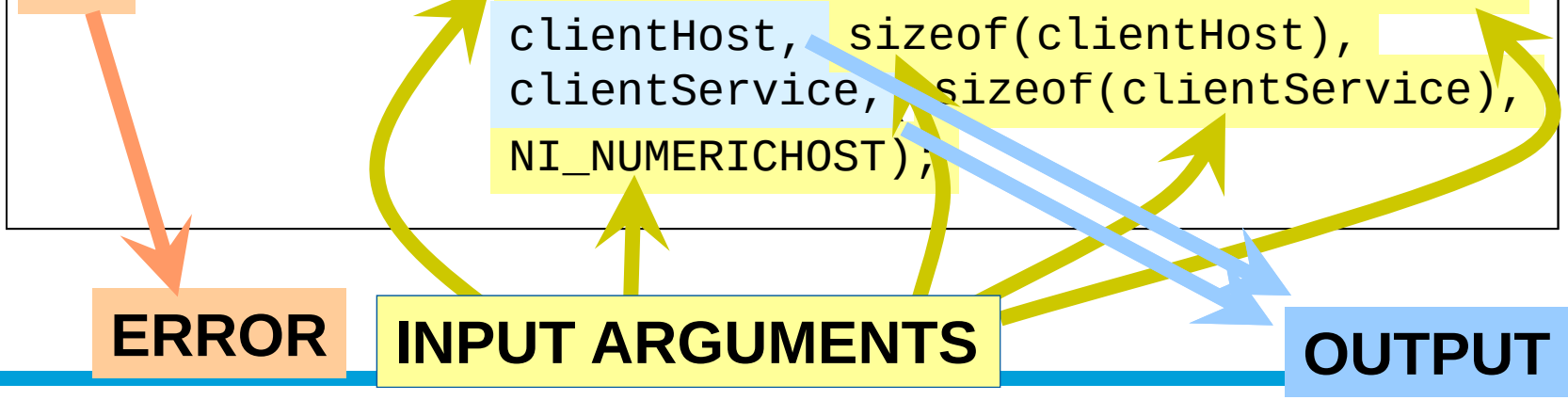
```
struct sockaddr_storage clientAddr;
char   clientHost[NI_MAXHOST];
char   clientService[NI_MAXSERV];
size_t alen = sizeof(struct sockaddr_storage);

/*… */

connectedfd = accept(serverfd,
                     (struct sockaddr *)&clientAddr,
                      &alen);
err =getnameinfo((struct sockaddr *)&clientAddr, alen,
                 clientHost, sizeof(clientHost),
                 clientService, sizeof(clientService),
                 NI_NUMERICHOST);
```



Socket Address to Name: Example (3)

Getting IP address as string for the accepted connection:

```
struct sockaddr_storage clientAddr;
char   clientHost[NI_MAXHOST];
char   clientService[NI_MAXSERV];
size_t alen = sizeof(struct sockaddr_storage);

/*… */

connectedfd = accept(serverfd,
                     (struct sockaddr *)&clientAddr,
                      &alen);
err =getnameinfo((struct sockaddr *)&clientAddr, alen,
                 clientHost, sizeof(clientHost),
                 clientService, sizeof(clientService),
                 NI_NUMERICHOST);
```



Use of getnameinfo() for Numeric Hostname

The following code returns the numeric hostname, and service name, for given socket address. Observe that there is no hardcoded reference to particular address family.

```
struct sockaddr *cliaddr;
 /* input */
char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];
. . .
 if (getnameinfo(cliaddr, cliaddr->sa_len, hbuf, sizeof(hbuf), sbuf,
       sizeof(sbuf),  NI_NUMERICHOST | NI_NUMERICSERV) != 0) {
             error(1, "could not get numeric hostname");
             /*NOTREACHED*/
   }
   printf("Connection From host=%s, serv=%s\n", hbuf, sbuf);
```

getnameinfo() for Reverse Address Lookup

The following code looks up the hostname corresponding to the socket address

```
struct sockaddr *sa;
/* input */
char hbuf[NI_MAXHOST];
. . .
if (getnameinfo(res->ai_addr, res->ai_addrlen,
  hbuf,
      sizeof(hbuf), NULL, 0, NI_NAMEREQD) != 0)
  {
        error(1, "could not resolve
  hostname");
       /*NOTREACHED*/
}
printf("host=%s\n", hbuf);
```

Presentation Format / Address Structure

Presentation Format <-> Address Structure

- Change inet_aton()/inet_addr() by:
  - **inet_pton()** (protocol dependent function)
  - **getaddrinfo()** (protocol independent function), remember to use freeaddrinfo

Address Structure -> Presentation Format

- Change inet_ntoa() by:
  - **inet_ntop()** (protocol dependent function)
  - **getnameinfo()** (protocol independent function) with flag NI_NUMERICHOST

Summary

- Use IP version independent structures:
  - sockaddr_storage
- Use IP version independent functions:
  - getaddrinfo()/getnameinfo()
- Not use inet_ntop()/inet_pton()
- Not use gethostbyname()/gethostbyaddr()
- Iterated jobs for finding the working address:
  - Server:
    - listening packets addressed to a specific port.
  - Clients:
    - connecting to one of the server addresses.

References

- "**Basic Socket Interface Extensions for IPv6**", http://www.ietf.org/rfc/rfc3493.txt
- "**Advanced Sockets Application Program Interface (API) for IPv6**", http://www.ietf.org/rfc/rfc3542.txt
- **"Default Address Selection for Internet Protocol Version 6 (IPv6)"**, http://www.ietf.org/rfc/rfc6724.txt
- **"Application Aspects of IPv6 Transition"**, http://www.ietf.org/rfc/rfc4038.txt
- **"Happy Eyeballs: Success with Dual-Stack Hosts"**, http://www.ietf.org/rfc/rfc6555.txt
- "**UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking API**", W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff, ISBN: 0-13141-155-1
- "**IPv6 Network Programming**", Jun-ichiro itojun Hagino, ISBN: 1-55558-318-0

Questions???

timothy.s.morizot@irs.gov