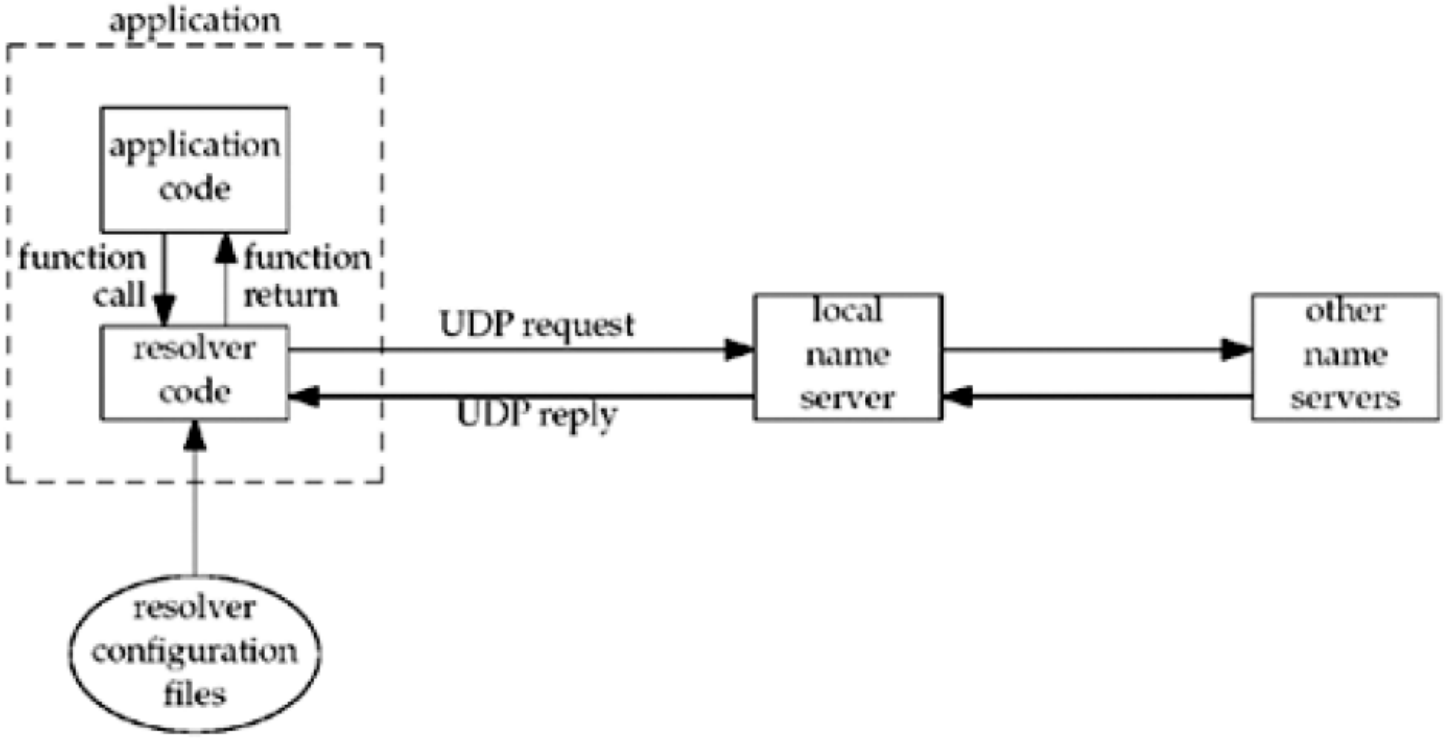
Advanced Name and Address Conversions

- *getaddrinfo, getnameinfo, gai_strerror, freeaddrinfo*
- *host_serv, tcp_connect, tcp_listen, udp_client, udp_connect, udp_server*
- Reentrant functions: *gethostbyname_r, gethostbyaddr_r*

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



CNAME CNAME stands for "canonical name." 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, the following could be CNAMEs for our host `linux`:

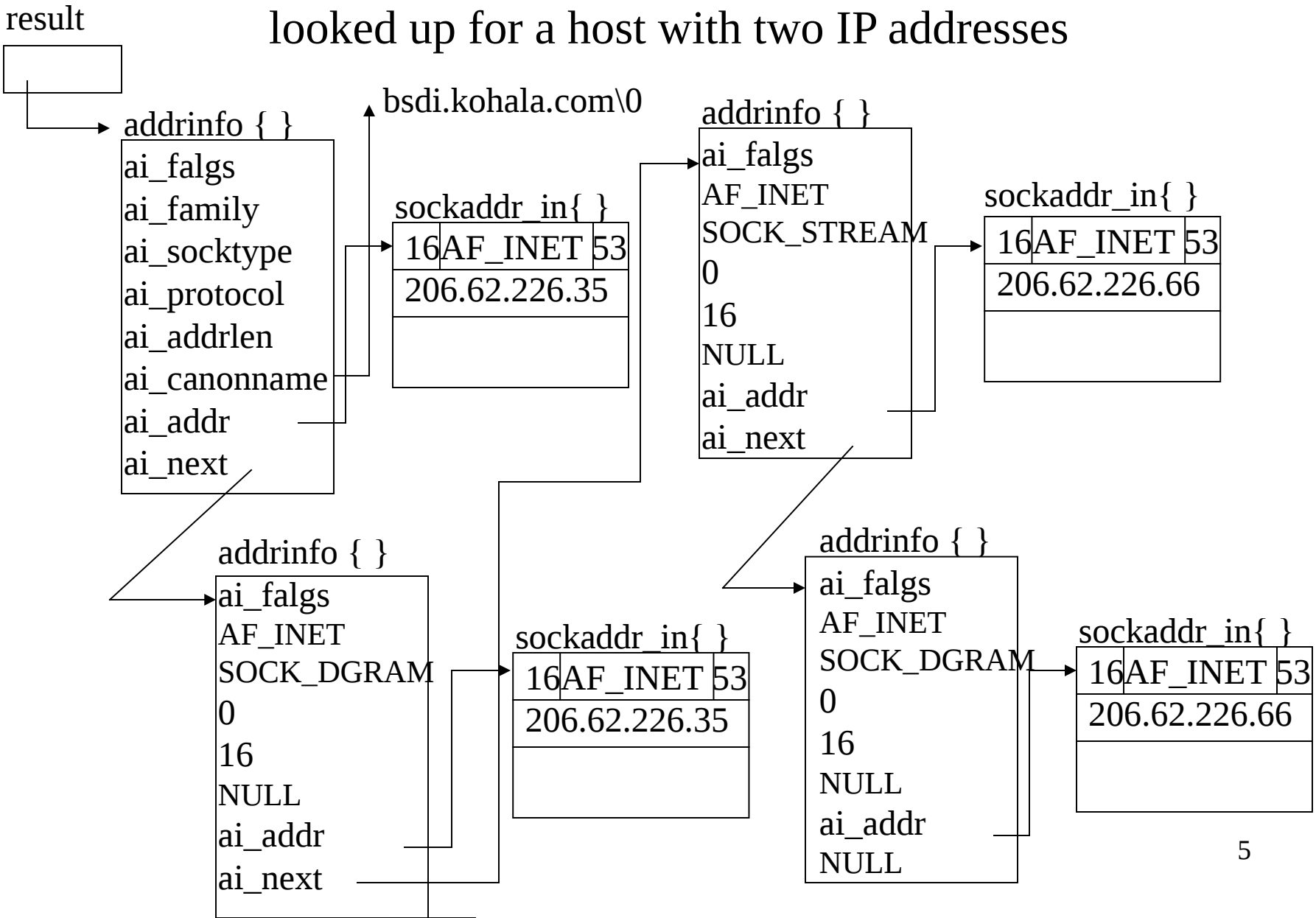
```
ftp             IN      CNAME     linux.unpbook.com.
www             IN      CNAME     linux.unpbook.com.
```

getaddrinfo Function

```
#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
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
*/
        size_t      ai_addrlen;     /* length of ai_addr */
        char        *ai_canonname;  /* ptr to cononical name for host */
        struct sockaddr *ai_addr;   /* ptr to socket address structure */
        struct addrinfo *ai_next;   /* ptr to next structure in linked list */   };
```

Example *addrinfo* Returned by *getaddrinfo*

no hints are provided and the domain service is looked up for a host with two IP addresses



getaddrinfo Example

```
main(int argc, char *argv[]) {

    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    int rv;

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if ((rv = getaddrinfo("www.apple.com", "http", &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        exit(1);
    }

    for(p = servinfo; p != NULL; p = p->ai_next) {
       printf("IP address is %s\n", inet_ntoa(((struct sockaddr_in *) p->ai_addr)->sin_addr));

    }
}
```

getnameinfo Example

```c
#include <netinet/in.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]) {

            struct sockaddr_in ssaa, *sain;    /* input */
            struct sockaddr *sa;    /* input */
            char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];

        sain = &ssaa;
        sain->sin_family = AF_INET;
        sain->sin_port = htons(80);
        inet_aton("140.113.1.1", &sain->sin_addr);
        sa = (struct sockaddr *) sain;
            if (getnameinfo(sa, sizeof(struct sockaddr), hbuf, sizeof(hbuf), sbuf,
                sizeof(sbuf), 0)) {
                    printf("could not get hostname\n");
            }
            printf("host=%s, serv=%s\n", hbuf, sbuf);
}
```

gai_strerror, freeaddrinfo, host_serv, getnameinfo

#include <netdb.h>
char *gai_strerror (int *error*);
returns: pointer to string describing error message
error: returned from *getaddrinfo*

#include <netdb.h>
void freeaddrinfo (struct addrinfo **ai*);

#include <unp.h>
struct addrinfo *host_serv (const char **hostname*, const char **service*,
int *family*, int *socktype*);
returns: pointer to addrinfo structure if OK, NULL on error
(initializes a hints structure, calls getaddrinfo, returns a pointer)

#include <netdb.h>
int getnameinfo (const struct sockaddr **sockaddr*, socklen_t *addrlen*,
char **host*, size_t *hostlen*, char **serv*, size_t *servlen*, int *flags*);
returns: 0 if OK, -1 on error

tcp_connect, tcp_listen, udp_client, udp_connect, udp_server

calls *getaddrinfo* and provides protocol independency

#include <unp.h>

int tcp_connect (const char **hostname*, const char **service*);

returns: connected socket descriptor if OK, no return on error

(calls getaddrinfo, tries each addrinfo structure by calling socket, connect)

int tcp_listen (const char **hostname*, const char **service*, socklen_t **lenptr*);

returns: connected socket descriptor if OK, no return on error

(calls getaddrinfo, creates socket, binds address, returns sockfd and size of socket address structure)

int udp_client (const char **hostname*, const char **service*,

void ***saptr*, socklen_t **lenp*);

returns: unconnected socket descriptor if OK, no return on error

int udp_connect (const char **hostname*, const char **service*);

returns: connected socket descriptor if OK, no return on error

int udp_server (const char **hostname*, const char **service*, socklen_t **lenptr*);

returns: unconnected socket descriptor if OK, no return on error

Reentrant Functions
gethostbyname_r, gethostbyaddr_r

Reentrant problem: gethostbyname/gethostbyaddr called from main flow of control and from a signal handler. *hostent* structure may be overwritten.

Two solutions:

1. Caller allocates the structure and reentrant function fills in the caller's structure --- *gethostbyanem_r*
2. Reentrant function calls malloc and dynamically allocates memory --- *getaddrinfo*

```
#include <netdb.h>
struct hostent *gethostbyname_r (const char *hostname, struct hostent *result,
                    char *buf, int buflen, int *h_errnop);
struct hostent *gethostbyaddr_r (const char *addr, int len, int type,
                    struct hostent *result, char *buf, int buflen, int *h_errnop);
```

both return: nonnull pointer if OK, NULL on error