Programmation Réseau

API C

Jean-Baptiste.Yunes@univ-paris-diderot.fr

UFR Informatique

2012-2013

Les flux réseaux en C

- Pré-requis : utiliser correctement les flux d'entrées-sorties…
- En C, il est important de se rappeler que l'API est de bas-niveau, i.e. tout est à faire soi-même ou quasiment…
  - retrouver l'adresse à partir du nom (consultation de l'annuaire)
  - forger les bonnes structures (NBO…)
  - paramétrer différents mécanismes sous-jacent (file d'attente…)
  - …

Les adresses Internet en C

- Rappelons qu'aujourd'hui les adresses Internet peuvent être de deux types IPv4 ou IPv6, en C cela est représenté par deux types différents :
  - une adresse IPv4 peut être stockée dans une variable de type `struct in_addr` ou `in_addr_t`
  - une adresse IPv6 peut être stockée dans une variable de type `struct in6_addr`
- La structure interne de ces types peut être oubliée (i.e. aucune raison d'avoir à manipuler l'intérieur de la structure soi-même)
- Le fichier d'inclusion associé est :

```
#include <netinet/in.h>
```

- Il existe différentes fonctions de manipulation de ces structures et permettant de les convertir depuis ou vers une chaîne de caractères...

- Inclure le fichier `<arpa/inet.h>`

- Pour la pile IPv4 :

```
char *inet_ntoa(struct in_addr);

int inet_aton(const char *,struct in_addr *);

in_addr_t inet_addr(const char *);
```

- Pour la pile IPv6, il existe désormais des fonctions génériques (qui fonctionnent pour toutes les piles) :

```
const char *inet_ntop(int af, const void *src,
                      char *chaine,
                      socklen_t size);
int inet_pton(int af, const char *chaine,
              void *dst);
```

- où `af` représente la famille protocolaire `AF_INET` ou `AF_INET6`, et `src` et `dst` des pointeurs vers des adresses Internet conformes à la valeur de `af`

Ces fonctions ne font PAS appel à l'annuaire, elles n'effectuent que des transformations entre représentations!

Les adresses de socket en C

- On rappelle qu'une socket est un point de communication et est donc caractérisé par :
  - une adresse Internet,
  - un numéro de port,
  - et un type de communication

- la structure `sockaddr`

  - il s'agit d'une structure générique représentant l'adresse d'une socket :

```
#include <sys/socket.h>
struct sockaddr {
  socklen_t    sa_len;
  sa_family_t sa_family;
  char            sa_data[];
};
```

- Le champ `sa_family` permet de spécifier le type de la socket (et donc la structure implémentant le type `sockaddr`). POSIX définit :
  - `AF_LOCAL` ou `AF_UNIX`, pour une socket locale
  - `AF_INET`, pour une socket IPv4
  - `AF_INET6`, pour une socket IPv6
- Nous ne nous intéresserons qu'aux sockets du domaine Internet...

- Pour une socket du domaine IPv4, la structure correspondante est :

```
#include <netinet/in.h>
struct sockaddr_in {
  socklen_t      sin_len;
  sa_family_t    sin_family;
  in_port_t      sin_port;
  struct in_addr sin_addr;
};
```

- Pour une socket du domaine IPv6, la structure correspondante est :

```
#include <netinet/in.h>
struct sockaddr_in6 {
  socklen_t       sin6_len;
  sa_family_t     sin6_family;
  in_port_t       sin6_port;
  uint32_t        sin6_flowinfo;
  struct in6_addr sin6_addr;
  …
};
```

L’accès à l’annuaire en C

- On peut avoir besoin de déterminer une adresse associée à un nom Internet
  - Ceci nécessite d’obtenir la
  - Il existe différences fonctions d’accès à l’annuaire (DNS)
    - l’historique `gethostbyname`
    - la moderne `getaddrinfo`

- La consultation de l'annuaire peut s'effectuer simplement par un appel à :

```
#include <netdb.h>
struct hostent *gethostbyname(const char
*nom);
```

- où :

```
struct hostent {
  char  *h_name;
  char **h_aliases;
  int    h_addrtype;
  int    h_length;
  char **h_addr_list;
};
```

- les différents champs correspondent à :
  - `h_name` est le nom officiel
  - `h_aliases` est le tableau des surnoms
  - `h_addrtype` est le type de l'adresse (`AF_INET` en général ou `AF_INET6`)
  - `h_length` est la longueur de l'adresse (`sizeof(struct in_addr)` ou `sizeof(struct in6_addr)`)
  - `h_addr_list` est un tableau d'adresses (au format NBO et terminé par un pointeur NULL)
- Attention la fonction et la structure associée ne permet pas de choisir quelle résolution (IPv4 ou IPv6) sera obtenue… On préfèrera `getaddrinfo` (à venir)

- Un exemple de code permettant d'obtenir l'ensemble des adresses (IPv4) disponibles pour chaque nom Internet donné en argument :
  - on récupère le `hostent` associé à un nom
  - pour chaque adresse on la convertit pour affichage à l'aide de la fonction :

```
char *inet_ntoa(struct sockaddr_in);
```

qui permet d'obtenir la chaîne de caractères représentant une adresse IPv4 (avec traduction NBO si nécessaire)

API C - DNS

```c
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv[]) {
  struct hostent *he; int i;
  if (argc<2) {
    fprintf(stderr,"usage: %s name [name...]\n",argv[0]);
    exit(EXIT_FAILURE);
  }
  for (i=1; i<argc; i++) {
    char **aliases; struct in_addr **addresses;
    printf("--------%s\n",argv[i]);
    he = gethostbyname(argv[i]);
    if (he==NULL) { printf("unknown\n"); continue; }
    printf("Official name : %s\n",he->h_name);
    aliases = he->h_aliases;
    while ((*aliases)!=NULL) {
      printf("Alias         : %s\n",*(aliases++));
    }
    for (addresses=(struct in_addr **)he->h_addr_list;
         *addresses!=NULL; addresses++) {
      printf("Address       : %s\n",inet_ntoa(**addresses));
    }
  }
  return 0;
}
```

- Un autre exemple plus générique mais donc complexe consiste à utiliser l'API plus récente :

```
int getaddrinfo(const char *nom,
                const char *service,
                const struct addrinfo *paramètres,
                struct addrinfo *first);
```

- que l'on ne décrira que partiellement…

- et qui permet d'obtenir (donc entre autres) une liste d'adresses (au sens très large) associées dans l'annuaire à un nom Internet donné

- La structure `addrinfo` est :

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

- Les champs utiles lors d'une résolution sont :
  - `ai_family`, `ai_socktype`, `ai_protocol`
  - et pour lesquels une `sockaddr` est utilisable (`ai_addrlen`, `ai_addr`)

- La résolution fournit une collection d'informations chaînées via `ai_next`

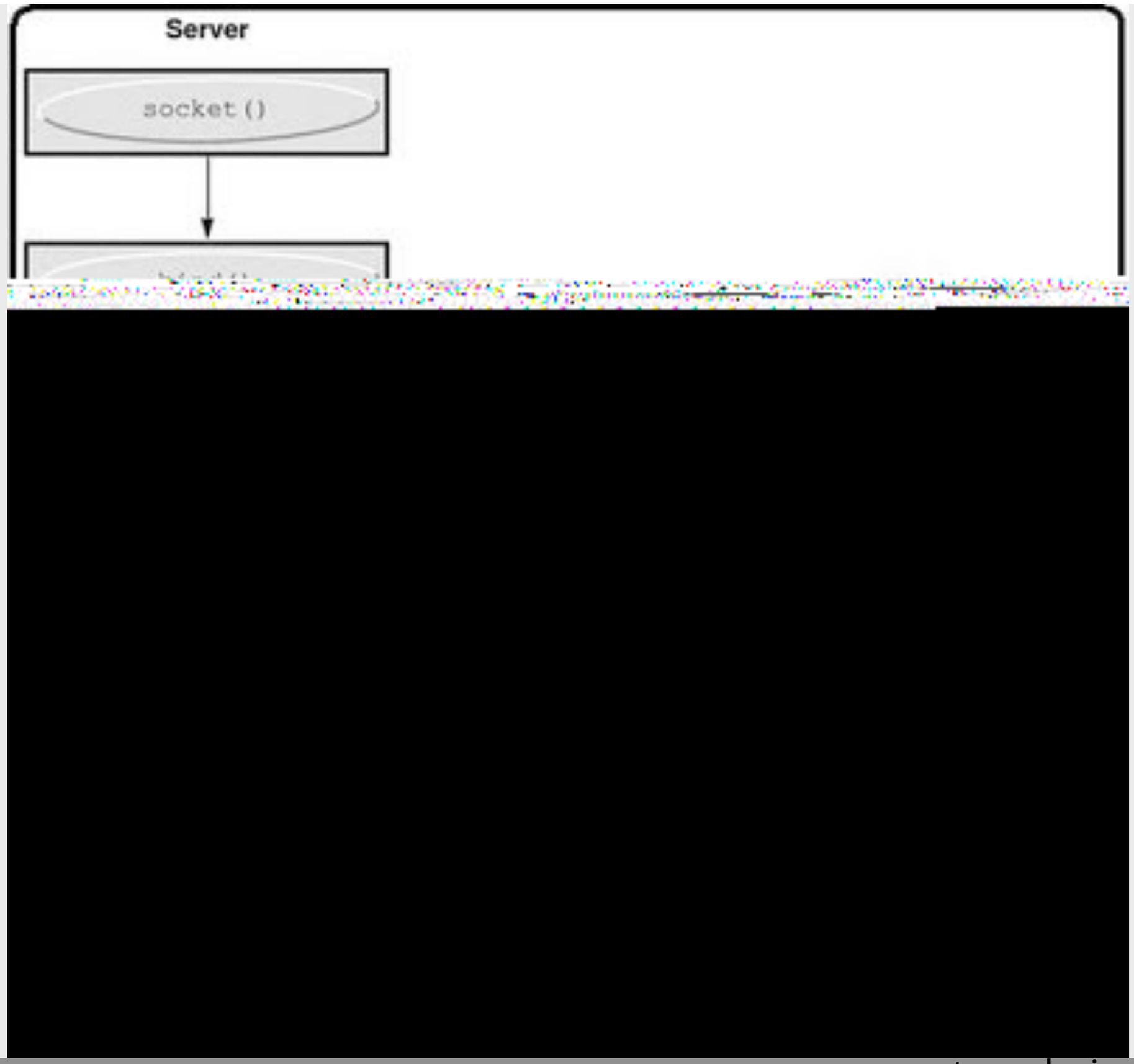
- Il est nécessaire de libérer la mémoire de cette chaîne par appel à :

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

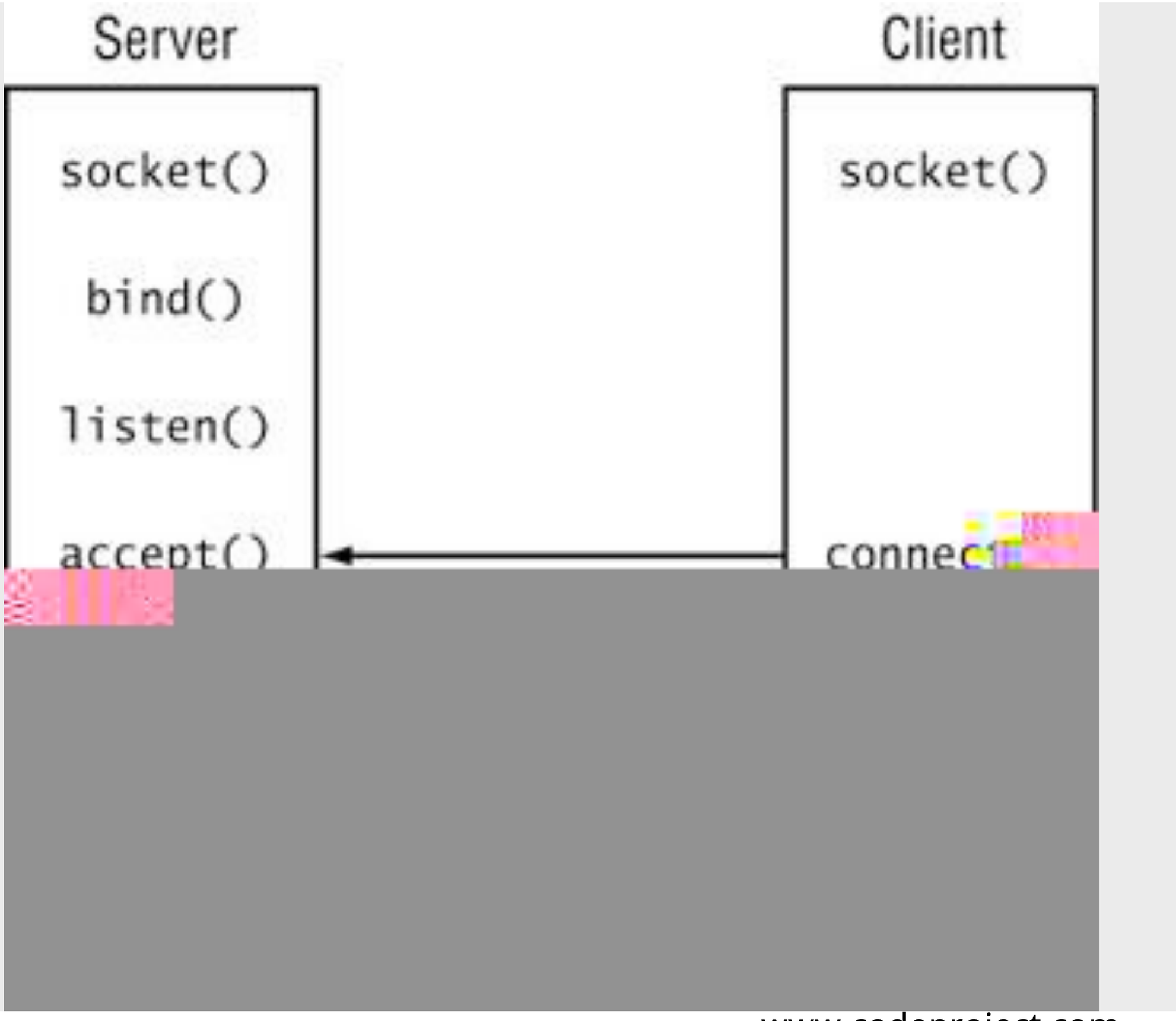
API C - DNS

```
int main(int argc,char *argv[]) {
  int i;
  if (argc<2) { fprintf(stderr,"usage: %s name [name...]\n",argv[0]);
exit(EXIT_FAILURE); }
  for (i=1; i<argc; i++) {
    struct addrinfo *first_info;
    printf("--------%s\n",argv[i]);
    if (getaddrinfo(argv[i],NULL,NULL,&first_info)) { printf("unknown\n"); } else {
      struct addrinfo *info;
      for (info=first_info; info!=NULL; info=info->ai_next) {
        char string[MAXLEN]; struct sockaddr *saddr = info->ai_addr;
        if (info->ai_socktype==SOCK_DGRAM) continue;
        switch (saddr->sa_family) {
          case AF_INET:
          { struct sockaddr_in *sin = (struct sockaddr_in *)saddr;
            inet_ntop(saddr->sa_family,&(sin->sin_addr),string,MAXLEN); }
            break;
          case AF_INET6:
          { struct sockaddr_in6 *sin6 = (struct sockaddr_in6 *)saddr;
            inet_ntop(saddr->sa_family,&(sin6->sin6_addr),string,MAXLEN); }
            break;
          default:
            strcpy(string,"???"); break;
        }
        printf("Address: %s\n",string);
      }
      freeaddrinfo(first_info);
    }
  }
}
```

API C

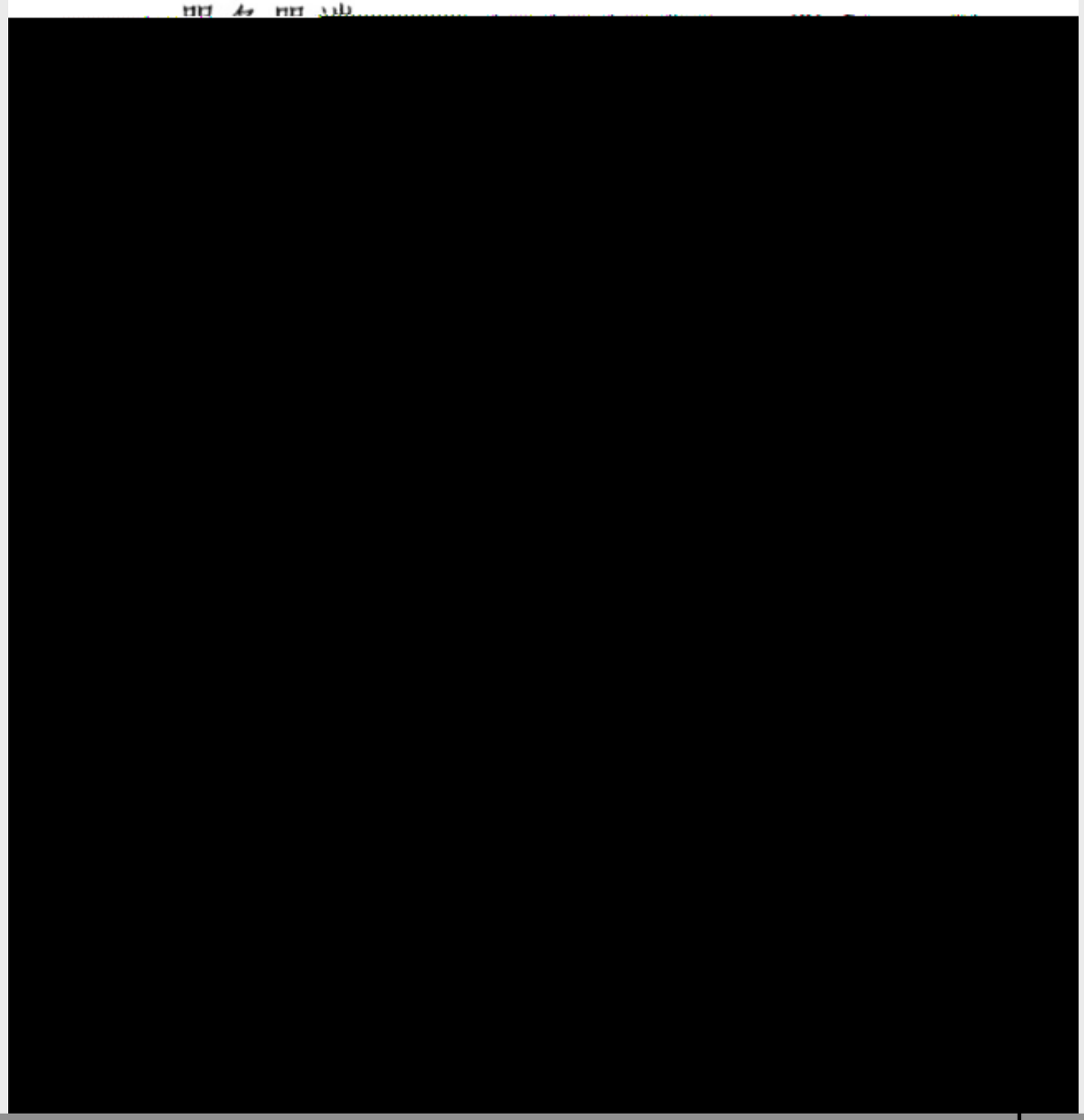


API C



www.codeproject.com

API C



- La création d'une socket s'effectue par appel à

```
#include <sys/socket.h>
int socket(int domaine,int type,int protocole);
```

- où :
  - `domaine` vaut `PF_INET` ou `PF_INET6` (pour ce qui nous intéresse)
  - `type` `SOCK_STREAM` (pour ce qui nous intéresse, mais il existe `SOCK_DGRAM`, `SOCK_RAW` ou `SOCK_SEQPACKET`)
  - `protocole` spécifie un protocole de la famille donnée (pour nous ce sera 0)

Les sockets doivent être associées à un point de communication

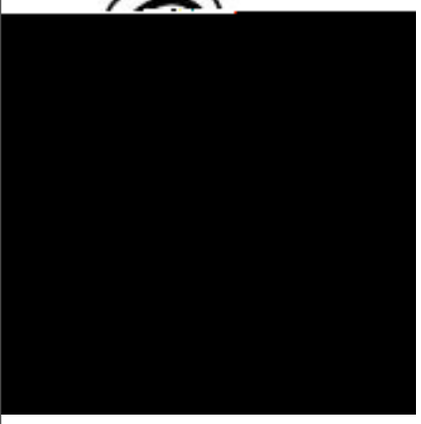
- Du côté serveur, il est nécessaire d'associer cette socket à un port donné sur l'une des (ou toutes les) adresses de la machine

- cette association s'effectue par appel à

```
#include <sys/socket.h>
int bind(int socket,
         const struct sockaddr *adresse,
         socklen_t longueur);
```

- où `longueur` est la longueur de la structure décrivant l'adresse

- en général le `bind` s'effectue sur `INADDR_ANY`



- une fois associée à un point de communication, il est nécessaire de faire de cette socket une socket serveur
  - en faisant en sorte que le système autorise la réception de demandes connexions entrantes
  - en contrôlant le nombre de demandes en attente possible
- La fonction à appeler est :

```
#include <sys/socket.h>
int listen(int socket,int attente);
```

- en général `attente` est égal à 0 (ce qui signifie qu'on laisse le système trouver une valeur appropriée)

- l'acceptation d'une demande de connexion entrante s'effectue par appel à

```
#include <sys/socket.h>
int accept(int socket,
           struct sockaddr *adresse,
           socklen_t *longueur);
```

- qui attend qu'une demande de connexion entrante arrive si la file d'attente est vide (ce comportement peut être modifié, O_NONBLOCK)

- qui extrait une demande si la file d'attente n'est pas vide

- la valeur de retour de la fonction `accept` est
  - un descripteur d'entrée/sortie correspondant à une socket de service
  - ce descripteur est manipulable comme n'importe quel autre (O_NONBLOCK, etc.)
  - on peut consulter ses caractéristiques via `getsockname()`…

- Du côté client, il faut demander l'établissement d'un connexion à l'aide de la fonction :

```
int connect(int socket,
            const struct sockaddr *adresse,
            socklen_t longueur);
```

- connexion tentée vers la socket serveur à l'`adresse` indiquée

- Nous savons que les communications via des socket en mode flux sont en duplex intégral (full-duplex), i.e. des informations peuvent circuler dans les deux sens

  - il est possible d'interdire certains opérations futures par appel à :

    ```
    #include <sys/socket.h>
    int shutdown(int socket,int quoi);
    ```

  - où quoi peut-être :

    - SHUT_RD
    - SHUT_WR
    - SHUT_RDWR

- Une fois la communication établie les descripteurs peuvent être utilisés pour lire et écrire à l'aide des fonctions ordinaires `read()` et `write()`

- la terminaison est normalement obtenue par `close()`

Exemple

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

int main(int argc,char *argv[]) {
  int s;
  struct sockaddr_in a;

  s = socket(PF_INET,SOCK_STREAM,0);
  if (s==-1) {
    fprintf(stderr,"socket problem\n");
    exit(EXIT_FAILURE);
  }
  bzero(&a,sizeof(a));
  a.sin_family = AF_INET;
  a.sin_port = htons(61234);
  a.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(s,(struct sockaddr *)&a,sizeof(a))==-1) {
    fprintf(stderr,"bind problem");
    close(s);
    exit(EXIT_FAILURE);
  }
```

Exemple

```
if (listen(s,0)==-1) {
  fprintf(stderr,"listen problem");
  close(s);
  exit(EXIT_FAILURE);
}
int d, l, i;
struct sockaddr_in c;
char buffer[256];
int lus;
do {
  if ((d=accept(s,(struct sockaddr *)&c,&l))==-1) {
    fprintf(stderr,"accept problem");
    close(s);
    exit(EXIT_FAILURE);
  }
  shutdown(s,SHUT_WR);
  while ((lus=read(d,buffer,256))>0) {
    for (i=0; i<lus; i++) printf("%2x(%c) ",buffer[i],buffer[i]);
    printf("\n");
  }
  close(d);
} while(lus==0);

  close(s);
}
```

- Pour les sockets, il est possible de lire via :

```
ssize_t recv(int socket,void *tampon,
                size_t longueur,int options);
```

- où les `options` possibles sont des combinaisons de :
  - `MSG_PEEK`, pour lire sans extraire les données
  - `MSG_OOB`, pour lire les messages hors-bande
  - `MSG_WAITALL`, pour demander une lecture de la quantité donnée (attente si nécessaire)

- Pour les sockets, il est possible d'écrire via :

```
ssize_t send(int socket,const void *tampon,
                    size_t longueur,int options);
```

- où options est une combinaison de :

  - MSG_EOR, pour terminer l'envoi par une marque de fin de message

  - MSG_OOB, pour envoyer des données hors-bande

- On notera qu'il est aussi possible de réaliser des transmissions via `recvfrom()` et `sendto()`
  - mais on réservera ces fonctions pour des communications en mode par paquet
- ou encore via `recvmsg()` et `sendmsg()`
  - on réservera l'emploi de celles-ci pour des écritures/lectures de messages

Le mode paquet en C

- pour le mode paquet, c'est le type de la socket qui doit être modifié
  - `SOCK_DGRAM`
- pour les envois, il n'est pas nécessaire de connecter la socket via `connect()`
  - il s'agit d'ailleurs d'une pseudo-connexion
    - aucun protocole sous-jacent n'est mis en œuvre...

Serveur

```
int main(int argc,char *argv[]) {
  struct sockaddr_in addr; int sock; char tampon[256];

  sock = socket(PF_INET,SOCK_DGRAM,0); // Protocol family
  if (sock==-1) {
    perror("socket: "); exit(1);
  }
  addr.sin_family = AF_INET; // Address family
  addr.sin_port = htons(PORT);
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sock,(struct sockaddr *)(&addr),sizeof(addr))==-1) {
    perror("bind: "); close(sock); exit(1);
  }
  while (1) {
    if (recv(sock,tampon,256,0)==-1) {
      perror("recv:"); close(sock); exit(1);
    }
    printf("Recu : %s\n",tampon);
    if (!strcmp(tampon,"quit")) break;
  }
  close(sock);
  return 0;
}
```

- L'implémentation d'un service d'écho nécessite
  - l'association, côté client, à une adresse et un port
  - l'utilisation de `recvfrom()`, côté serveur, afin de récupérer l'adresse du client

Client

```
int main(int argc,char *argv[]) {
  struct sockaddr_in addrto, addrfrom; int sock; char tampon[256]; struct hostent *hent;

  hent = gethostbyname(argv[1]);
  if (hent==NULL) { fprintf(stderr,"%s: host %s unknown\n",argv[0],argv[1]); exit(1); }
  sock = socket(PF_INET,SOCK_DGRAM,0); // Protocol family
  if (sock==-1) { perror("socket: "); exit(1); }
  addrfrom.sin_family = AF_INET;                // Address family
  addrfrom.sin_port = htons(0);                 // Any port
  addrfrom.sin_addr.s_addr = htonl(INADDR_ANY); // Any address
  if (bind(sock,(struct sockaddr *)(&addrfrom),sizeof(addrfrom))==-1) {
    perror("bind: "); close(sock); exit(1);
  }
  addrto.sin_family = hent->h_addrtype;         // destination from db
  memcpy(&(addrto.sin_addr.s_addr),hent->h_addr_list[0],hent->h_length);
  addrto.sin_port = htons(PORT);                // fixed port
  strcpy(tampon,argv[2]);
  if (sendto(sock,tampon,256,0,(struct sockaddr *)(&addrto),sizeof(addrto))==-1) {
    perror("sendto:"); close(sock); exit(1);
  }
  if (strcmp(tampon,"quit")) {
    if (recv(sock,tampon,256,0)==-1) {
      perror("recv:"); close(sock); exit(1);
    }
    printf("Recu : %s\n",tampon);
  }
  close(sock);
  return 0;
}
```

Serveur

```
int main(int argc,char *argv[]) {
  struct sockaddr_in addr, from; int sock; char tampon[256]; socklen_t lg;

  sock = socket(PF_INET,SOCK_DGRAM,0); // Protocol family
  if (sock==-1) { perror("socket: "); exit(1); }
  addr.sin_family = AF_INET;            // Address family
  addr.sin_port = htons(PORT);
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sock,(struct sockaddr *)(&addr),sizeof(addr))==-1) {
    perror("bind: "); close(sock); exit(1);
  }
  while (1) {
    if (recvfrom(sock,tampon,256,0,(struct sockaddr *)(&from),&lg)==-1) {
      perror("recv:"); close(sock); exit(1);
    }
    printf("Recu : %s\n",tampon);
    if (!strcmp(tampon,"quit")) break;
    if (sendto(sock,tampon,256,0,(struct sockaddr *)(&from),lg)==-1) {
      perror("sendto:"); close(sock); exit(1);
    }
    printf("Renvoye : %s\n",tampon);
  }
  close(sock);
  return 0;
}
```