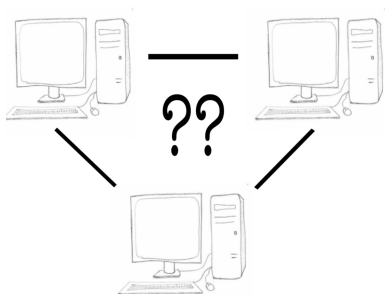


# PROGRAMMATION RÉSEAU

Arnaud Sangnier  
sangnier@irif.fr

## API TCP C - II



## Interroger l'annuaire en C

- On peut désirer récupérer l'adresse Internet associée à un nom Internet
  - Ceci nécessite d'obtenir **la résolution de nom**
  - Il existe différentes fonction d'accès à l'annuaire DNS
    - La fonction historique
      - **struct hostent \*gethostbyname(const char \*name);**
    - La fonction moderne
      - int getaddrinfo(const char \*node, // "www.example.com" or IP**  
**const char \*service, // "http" or port number**  
**const struct addrinfo \*hints,**  
**struct addrinfo \*\*res);**

## Accès à une machine

```
inet_aton("10.0.0.1", &address_sock.sin_addr);
```



- Il faut interroger l'annuaire DNS

## La fonction gethostbyname

- **#include <netdb.h>**  
**struct hostent \*gethostbyname(const char \*name);**
- L'appel à cette fonction renvoie une structure de la forme suivante :  
**struct hostent {**  
**char \*h\_name;** // Le nom canonique  
**char \*\*h\_aliases;** // Une liste d'alias - le dernier élément est NULL  
**int h\_addrtype;** // Le type de l'adresse, qui devrait être AF\_INET en général  
**int h\_length;** // La longueur des adresses en octet  
**char \*\*h\_addr\_list;** // Une liste d'adresses IP pour cet host  
**};**
- En fait la dernière est un tableau de **struct in\_addr \***, le dernier élément est NULL aussi

## Exemple

- On va faire un code qui pour un nom de machine va récupérer toutes les adresses IPv4 correspondantes et les afficher. On affichera également les alias associés à un nom
- Pour cela :
  - On récupère le hostent correspondant
  - On parcourt les tableaux d'alias et d'adresses
  - Pour chaque adresse, on la traduit en chaîne de caractères grâce à la fonction :

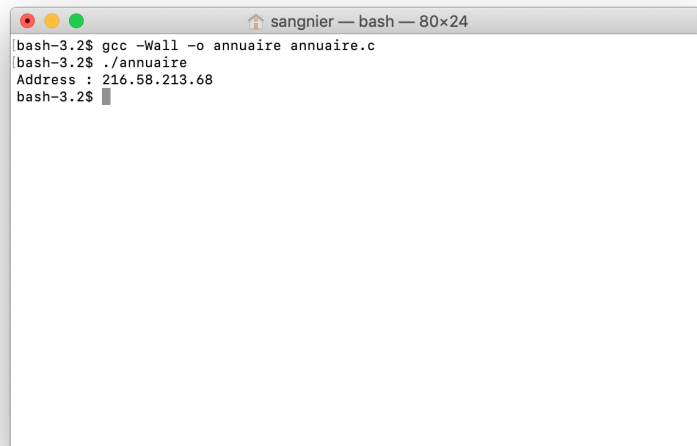
**char \* inet\_ntoa(struct in\_addr)**

## Récupération d'IP

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

int main() {
    struct hostent* host;
    host=gethostbyname("www.google.com");
    if(host==NULL) {
        printf("Unknown\n");
    }
    char **aliases=host->h_aliases;
    while (*aliases!=NULL) {
        printf("Alias : %s\n",*aliases);
        aliases++;
    }
    struct in_addr **addresses=(struct in_addr**)host->h_addr_list;
    while (*addresses!=NULL) {
        printf("Address : %s\n",inet_ntoa(**addresses));
        addresses++;
    }
    return 0;
}
```

## Résultat



```
bash-3.2$ gcc -Wall -o annuaire annuaire.c
bash-3.2$ ./annuaire
Address : 216.58.213.68
bash-3.2$
```

## La fonction getaddrinfo

- Cette fonction est plus générique mais donc plus complexe à utiliser !!!
- C'est la fonction que l'on recommande d'utiliser
  - int getaddrinfo(const char \*node, // "www.example.com" or IP**
  - const char \*service, // "http" or port number**
  - const struct addrinfo \*hints,**
  - struct addrinfo \*\*res);**
- On ne décrira que partiellement son utilisation
- Cette fonction permet d'obtenir entre autres choses une liste d'adresses (au sens très large) associées à un nom Internet dans l'annuaire
- En pratique elle remplit une structure de type **struct addrinfo** qui est stockée dans la variable **res**
- On remarque qu'on peut donner aussi un numéro de port (mais on peut mettre NULL, si on veut juste une adresse)
- Cette fonction renvoie 0 si tout se passe bien

## La structure struct addrinfo

```
struct addrinfo {
    int ai_flags;
    int ai_family; // la famille du protocole AF_xxxx
    int ai_socktype; // le type de la socket SOCK_xxx
    int ai_protocol;
    socklen_t ai_addrlen; // la longueur de ai_addr
    struct sockaddr *ai_addr; // l'adresse binaire
    char *ai_canonname; // le nom canonique
    struct addrinfo *ai_next; // le pointeur vers la structure suivante
};
```

- Il s'agit d'une liste chaînée, **ai\_next** est le successeur
- Il faut libérer la mémoire de la liste après utilisation grâce à

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

PR - API TCP C

9

## Récupération d'IP (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

int main() {
    struct addrinfo *first_info;
    struct addrinfo hints;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = PF_UNSPEC;
    int r=getaddrinfo("www.google.com", NULL, &hints, &first_info);
    if(r==0) {
        struct addrinfo *info=first_info;
        while(info!=NULL) {
            struct sockaddr *saddr=info->ai_addr;
            if(saddr->sa_family==AF_INET) {
                struct sockaddr_in *addressin=(struct sockaddr_in *)saddr;
                struct in_addr address=(struct in_addr) (addressin->sin_addr);
                printf("Address : %s\n", inet_ntoa(address));
            }
        }
    }
}
```

PR - API TCP C

10

## Récupération d'IP (2)

```
if(saddr->sa_family==AF_INET6) {
    struct sockaddr_in6 *addressin=(struct sockaddr_in6 *)saddr;
    struct in6_addr address=(struct in6_addr)
        (addressin->sin6_addr);
    char *string_address=(char *)malloc(
        sizeof(char)*INET6_ADDRSTRLEN);
    inet_ntop(AF_INET6, &address, string_address,
        INET6_ADDRSTRLEN);
    printf("Address IPv6 : %s\n", string_address);
}
info=info->ai_next;
}
}
freeaddrinfo(first_info);
return 0;
}
```

PR - API TCP C

11

## Récupération d'IP (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

int main() {
    struct addrinfo *first_info;
    struct addrinfo hints;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;
    int r=getaddrinfo("www.google.com", NULL, &hints, &first_info);
    if(r==0) {
        struct addrinfo *info=first_info;
        while(info!=NULL) {
            struct sockaddr *saddr=info->ai_addr;
            struct sockaddr_in *addressin=(struct sockaddr_in *)saddr;
            struct in_addr address=(struct in_addr) (addressin->sin_addr);
            printf("Address : %s\n", inet_ntoa(address));
            info=info->ai_next;
        }
    }
    freeaddrinfo(first_info);
    return 0;
}
```

12

## Un dernier exemple sans connaître l'Ip

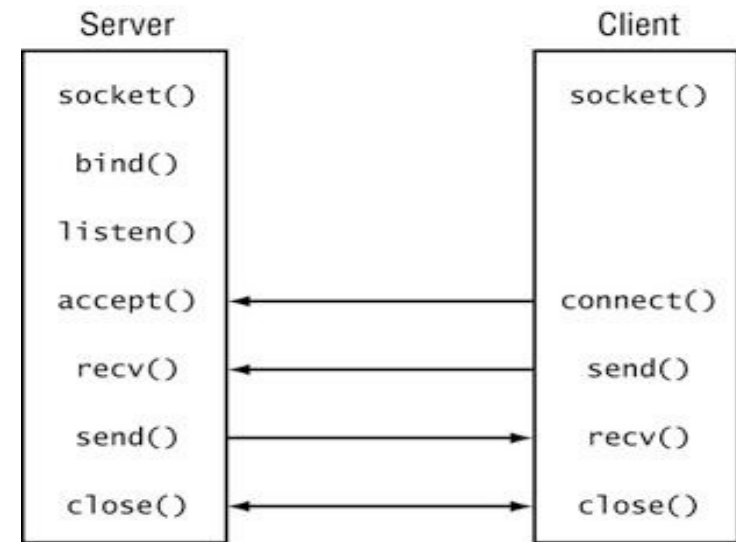
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>

//Code pour se connecter sur le serveur TCP sur le port 7 de lulu

int main() {
    struct addrinfo *first_info;
    struct addrinfo hints;
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    int r=getaddrinfo("lulu.informatique.univ-paris-diderot.fr", "7", &hints, &first_info);
    if(r==0){
        struct addrinfo *info=first_info;
        int found=0;
        struct sockaddr *saddr;
        struct sockaddr_in *addressin;
        if(info!=NULL){
            saddr=info->ai_addr;
            addressin=(struct sockaddr_in *)saddr;
            Found=1;
        }
        if(found==1){
            int descr=socket(PF_INET,SOCK_STREAM,0);
            int r2=connect(descr,(struct sockaddr *)addressin,
                sizeof(struct sockaddr_in));
            ///To be continued
        }
    }
}
```

3

## Schéma Client-Serveur en C



PR - API TCP C

14

## Côté Serveur - Lier la socket à un port

- Il faut associer la socket à un port donné
- **int bind(int sockfd, struct sockaddr \*my\_addr, int addrlen);**
- Comme pour connect, en IPv4, le deuxième argument sera souvent de type **struct sockaddr\_in** et le troisième sera **sizeof(struct sockaddr\_in)**
- Comme on est sur le serveur, on n'a pas besoin de spécifier l'adresse de la machine dans la plupart des cas, donc on mettra comme adresse en remplissant la valeur **htonl(INADDR\_ANY)**
- Le numéro de port est fourni en remplissant la structure du deuxième argument

PR - API TCP C

15

## Côté Serveur - exemple pour bind

```
int sock=socket(PF_INET,SOCK_STREAM,0);
struct sockaddr_in address_sock;
address_sock.sin_family=AF_INET;
address_sock.sin_port=htons(4242);
address_sock.sin_addr.s_addr=htonl(INADDR_ANY);
int r=bind(sock,(struct sockaddr *)&address_sock,sizeof(struct
sockaddr_in));
```

- La ligne **address\_sock.sin\_addr.s\_addr=htonl(INADDR\_ANY);** sert à préciser que l'on peut prendre n'importe quelle adresse dans la structure
  - On remplit le champ **s\_addr** de la structure **struct in\_addr**

PR - API TCP C

16

## Côté Serveur - Écouter sur le port

- Une fois associée à un port, il faut faire de la socket et une socket serveur
  - On fait en sorte que le système autorise les demandes de connexion entrantes
  - On peut aussi préciser le nombre de demandes en attente possibles
- La fonction qui fait cela
  - **int listen(int sockfd, int backlog);**
- **backlog** précise le nombre de demandes en attente autorisé
- En général, on le met à 0 pour laisser le système décidé

```
r=listen(sock,0) ;
```

## Côté Serveur - Utilisation d'accept

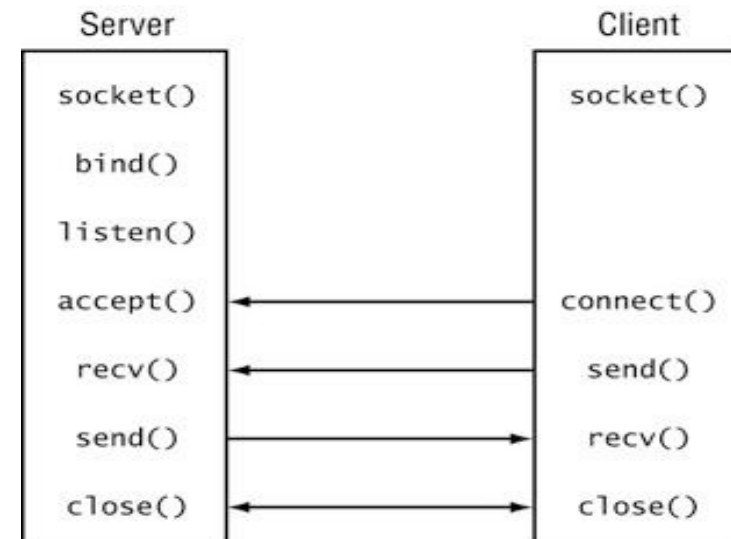
```
struct sockaddr_in caller;  
socklen_t size=sizeof(caller);  
int sock2=accept(sock, (struct sockaddr *)&caller, &size);
```

- Quand une connexion est acceptée, le programme remplit la structure caller avec les informations sur qui se connecte
- On communique ensuite sur **sock2**
- Ne pas oublier de fermer cette socket (et pas **sock**) avant d'accepter une nouvelle communication

## Côté Serveur - Accepter une connexion

- L'acceptation d'une demande de connexion se fait grâce :
  - **int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);**
- Attend qu'une demande de connexion arrive si la file d'attente est vide
- On verra plus tard comment passer en mode non-bloquant
- Extrait une demande de la file d'attente
- **Renvoie un descripteur de la socket créé pour communiquer**
- De plus, cette fonction remplit les champs addr et addrlen avec des infos sur qui s'est connecté
- On pourra en particulier savoir quel hôte s'est connecté sur quel port
- **Erreur classique : communiquer sur sockfd !!!!!!!**

## Schéma Client-Serveur en C



## Exemple

```
int main() {
    int sock=socket(PF_INET,SOCK_STREAM,0);
    struct sockaddr_in address_sock;
    address_sock.sin_family=AF_INET;
    address_sock.sin_port=htons(4242);
    address_sock.sin_addr.s_addr=htonl(INADDR_ANY);
    int r=bind(sock,(struct sockaddr *)&address_sock,sizeof(struct sockaddr_in));
    if(r==0){
        r=listen(sock,0);
        while(1){
            struct sockaddr_in caller;
            socklen_t size=sizeof(caller);
            int sock2=accept(sock,(struct sockaddr *)&caller,&size);
            if(sock2>=0){
                char *mess="Yeah!\n";
                send(sock2,mess,strlen(mess)*sizeof(char),0);
                char buff[100];
                int recu=recv(sock2,buff,99*sizeof(char),0);
                buff[recu]='\0';
                printf("Message recu : %s\n",buff);
            }
            close(sock2);
        }
    }
    return 0;
}
```

## Récupération d'informations

```
int main() {
    int sock=socket(PF_INET,SOCK_STREAM,0);
    struct sockaddr_in address_sock;
    address_sock.sin_family=AF_INET;
    address_sock.sin_port=htons(4242);
    address_sock.sin_addr.s_addr=htonl(INADDR_ANY);
    int r=bind(sock,(struct sockaddr *)&address_sock,sizeof(struct
sockaddr_in));
    if(r==0){
        r=listen(sock,0);
        while(1){
            struct sockaddr_in caller;
            socklen_t size=sizeof(caller);
            int sock2=accept(sock,(struct sockaddr *)&caller,&size);
            if(sock2>=0){
                printf("Port de l'appelant: %d\n",ntohs(caller.sin_port));
                printf("Adresse de l'appelant: %s\n",inet_ntoa(caller.sin_addr));
            }
            close(sock2);
        }
    }
    return 0;
}
```