PROGRAMMATION RÉSEAU

Arnaud Sangnier
sangnier@irif.fr

API TCP C (2)

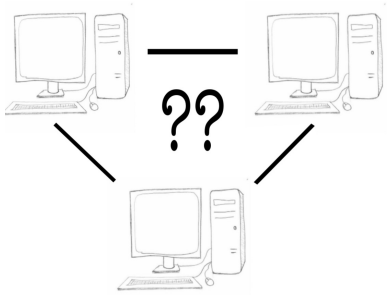
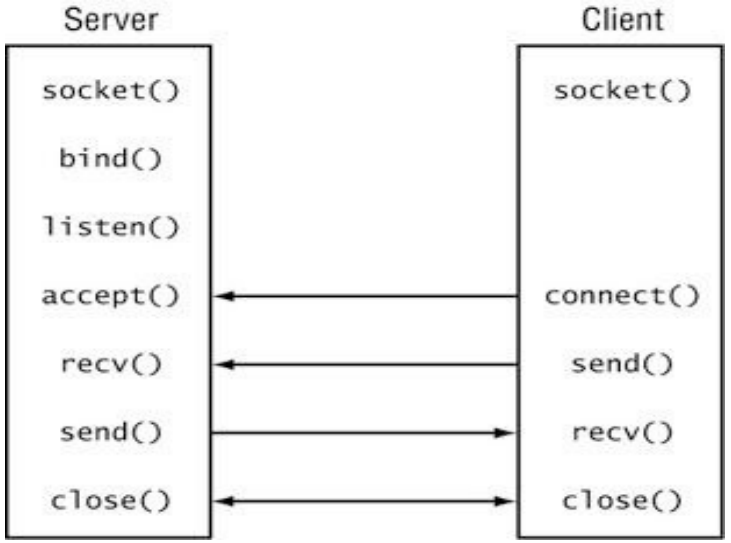


Schéma Client-Serveur en C



Création d'une socket

- La création d'une socket se fait grâce à :
  - **#include <sys/socket.h >**

    int socket(int domaine, int type, int protocol)
- Pour nous :
  - **domaine** vaudra **PF_INET** (pour IPv4) ou **PF_INET6** (pour IPv6)
  - **type** vaudra **SOCK_STREAM** (pour les sockets TCP)
  - **protocol** spécifie le procole de communication (mais pour TCP, on peut mettre 0 et le protcole est chosi de façon automatique)
- L'entier renvoyé sera le descripteur utilisé pour comuniquer

Côté client

- Il faut demander l'établissement d'une connexion à l'aide de la fonction suivante :

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

- On connecte la socket correspondante
- Pour rappel dans les objets de type struct sockaddr_in, on met une adresse et un port
- Pour le dernier argument, si on est en IPv4 et que adresse est de type **struct sockaddr_in**, on pourra mettre **sizeof(struct sockaddr_in)**
- Quand on a fini la communication, on peut fermer le descripteur de socket avec la commande
  - **int close(int fildes);**

Pour communiquer

- On va envoyer et recevoir des caractères sur le descripteur de socket
- Pour recevoir on va utiliser
- **ssize_t read(int filedes, void *buf, size_t nbyte);**
  - Remplit le buffer **buf**
  - **nbyte** est la taille maximale de **buf**
  - renvoie le nombre de données reçu (-1 si erreur et 0 si la connexion est fermée)
- Pour envoyer on va utiliser
- **ssize_t write(int filedes, void *buf, size_t nbyte);**
  - Même principe que readl en est la taille en octet de buf

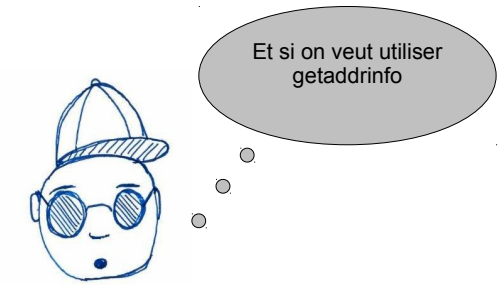
Exemple

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

int main() {
  struct sockaddr_in adress_sock;
  adress_sock.sin_family = AF_INET;
  adress_sock.sin_port = htons(4242);
  inet_aton("127.0.0.1",&adress_sock.sin_addr);

  int descr=socket(PF_INET,SOCK_STREAM,0);
  int r=connect(descr,(struct sockaddr *)&adress_sock,
                sizeof(struct sockaddr_in));
  if(r!=-1){
    char buff[100];
    int size_rec=read(descr,buff,99*sizeof(char));
    buff[size_rec]='\0';
    printf("Caracteres recus : %d\n",size_rec);
    printf("Message : %s\n",buff);
    char *mess="SALUT!\n";
    write(descr,mess,strlen(mess));
  }
  return 0;
}
```


Accès à une machine



int getaddrinfo(const char *node, //. "www.example.com" or IP
const char *service, // "http" or port number
const struct addrinfo *hints,
struct addrinfo **res);

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 *);

La fonction getaddrinfo

```
int getaddrinfo(const char *node,    //. "www.example.com" or IP
                const char *service,  //  "http" or port number
                const struct addrinfo *hints,
                struct addrinfo **res);
```

- En pratique elle remplit une structure de type **struct addrinfo** qui est stockée dans la variable **res**
- **On remarque que l'on peut donner le port**
- Dans le champ, **ai_addr** on trouvera un objet de type **struct addr**
- On peut filtrer les données remplies dans la structure **res** grâce à la structure **hints**
  - En pratique on mettra tous les champs de **hints** a 0 et on utilisera **ai_family** et **ai_socktype**

Utilisation

```
struct addrinfo hints;
bzero(&hints, sizeof(struct addrinfo));
hints.ai_family = AF_INET;
hints.ai_socktype=SOCK_STREAM;
```

- La deuxième ligne met tout à 0
- La troisième ligne servira pour filtrer uniquement des adresses IPV4
- La quatrième ligne précise que l'on veut des adresses sockets TCP
- Et ensuite, il suffit de faire :

```
struct addrinfo *first_info;
int r=getaddrinfo("lucien","4242",&hints,&first_info);
```


Exemple 1

```
int main() {
  int sock=socket(PF_INET,SOCK_STREAM,0);
  struct sockaddr_in *addressin;
  struct addrinfo *first_info;
  struct addrinfo hints;
  bzero(&hints,sizeof(struct addrinfo));
  hints.ai_family = AF_INET;
  hints.ai_socktype=SOCK_STREAM;
  int r=getaddrinfo("localhost","4242",&hints,&first_info);
  if(r==0){
    if(first_info!=NULL){
      addressin=(struct sockaddr_in *)first_info->ai_addr;
      int ret=connect(sock,(struct sockaddr *)addressin,(socklen_t)sizeof(struct
sockaddr_in));
      if(ret==0){
        char buff[100];
        int recu=read(sock,buff,99*sizeof(char));
        buff[recu]='\0';
        printf("Message recu : %s\n",buff);
        char *mess="Salut!\n";
        write(sock,mess,strlen(mess)*sizeof(char));
      }
      else{
        printf("Probleme de connexion!\n");
      }
      close(sock);
    }
  }
  return 0;
}
```


Exemple 2

```
int main() {
  int sock=socket(PF_INET,SOCK_STREAM,0);
  struct sockaddr_in *addressin;
  struct addrinfo *first_info;
  struct addrinfo hints;
  bzero(&hints,sizeof(struct addrinfo));
  hints.ai_family = AF_INET;
  hints.ai_socktype=SOCK_STREAM;
  int r=getaddrinfo("lucien","13",&hints,&first_info);
  if(r==0){
    if(first_info!=NULL){
      addressin=(struct sockaddr_in *)first_info->ai_addr;
      int ret=connect(sock,(struct sockaddr *)addressin,(socklen_t)sizeof(struct
sockaddr_in));
      if(ret==0){
        char buff[100];
        int recu=read(sock,buff,99*sizeof(char));
        buff[recu]='\0';
        printf("Message recu : %s\n",buff);
      }
      else{
        printf("Probleme de connexion!\n");
      }
      close(sock);
    }
  }
  return 0;
}
```

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

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**

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 - Accepter une connexion

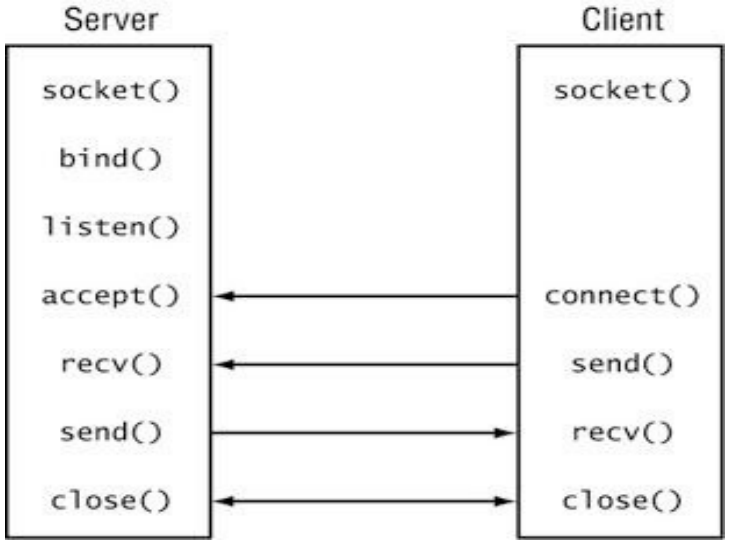
- L'acceptation d'une demande de connexion se fait grâce :
  - **int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);**
- Attend qu'une demande de connexion arrive si la file d'attente est vide
- On verra plus tard comment passer en mode non-bloquant
- Extrait une demande de la file d'attente
- **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 !!!!!!!**

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

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;
}
```