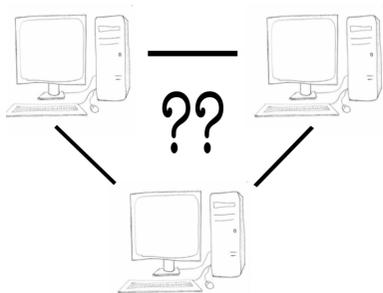


PROGRAMMATION RÉSEAU

Arnaud Sangnier
sangnier@irif.fr

API TCP C



Communication avec un service

- Un service tourne sur une **machine** et écoute sur un **port**
- Il existe au moins deux grands mode de communication
 - 1) Par **flux** (*stream*)
 - Dans ce mode, on est connecté
 - On se connecte, on échange des messages et on se déconnecte
 - Pensez au téléphone !
 - Exemple : **TCP** (Transmission Control Protocol) **RFC 793**
 - 2) Par **paquet** (*datagram*)
 - Dans ce mode, pas de connexion
 - On envoie des messages, peu importe si quelqu'un les reçoit
 - Pensez à la poste !
 - Exemple : **UDP** (User Datagram Protocol) **RFC 768**

Différence flux et paquet

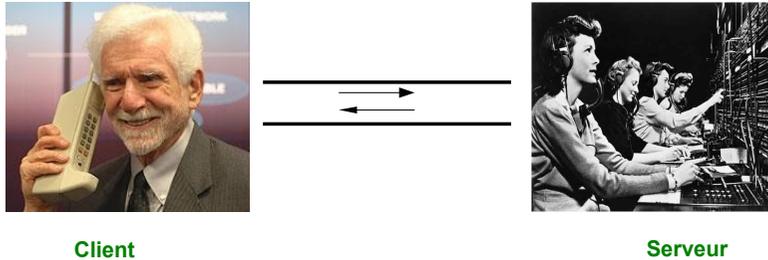
- Dans la communication **par flux** (comme **TCP**)
 - Les informations sont reçues dans l'ordre de leur émission
 - Il n'y a pas de perte
 - Inconvénient :
 - Établissement d'une connexion
 - Nécessité de ressources supplémentaire pour la gestion
- Dans la communication **par paquet** (comme **UDP**)
 - Pas d'ordre dans la délivrance des paquets
 - Un paquet posté en premier peut arrivé en dernier
 - Pas de fiabilité
 - Un paquet envoyé peut être perdu

Remarques

- Quand on communique avec un service, il faut connaître le mode de communication (TCP ou UDP dans notre cas)
- **telnet** sert à communiquer par **TCP**
- L'outil **netcat** (**nc**) permet lui de communiquer en **UDP** et TCP
- On verra plus tard comment utiliser **netcat**
- Pour certains services, il existe à la fois un mode **TCP** et un mode **UDP**
 - Allez voir le fichier **/etc/services**
 - On peut voir ces services comme deux services différents
 - Il se peut qu'un service disposant des deux modes soit actif en mode **TCP** et pas en mode **UDP**

Comment voir TCP

TCP -> communication avec flux, i.e. communication en mode connecté



PR - API TCP C

5

Aujourd'hui

- Comment communiquer en **C** en TCP
 - Comment se connecter à un service (présent sur une **machine** et écoutant sur un **port**)
 - Comment lire les messages envoyés par ce service
 - Comment envoyer des messages à ce service
 - En d'autres termes, comment créer un **client TCP**
- Comment créer un service qui écoute sur un port donné de la machine où l'on se trouve
 - En d'autres termes, comment créer un **serveur TCP**

PR - API TCP C

6

Les adresses internet en C

- **Rappel** : aujourd'hui les adresses internet peuvent être de deux types
 - les adresses **IPv4** sur 4 octets (donc 32 bits)
 - par exemple : **173.194.66.106**
 - les adresses **IPv6** sur 16 octets (donc 128 bits) ; 8 groupes de 2 octets
 - par exemple : **2a00:1450:400c:c02:0:0:93**
- Types des variables stockant les adresses IPv4 :
 - **struct in_addr** ou **in_addr_t**
- Types des variables stockant les adresses IPv6 :
 - **struct in6_addr**
- Il n'est pas nécessaire de connaître la structure interne de ces types (i.e. aucune raison d'avoir à manipuler l'intérieur de la structure soi-même)
- Le fichier à inclure pour manipuler ces types :
 - **#include <netinet/in.h>**

PR - API TCP C

7

Manipulation des structures d'adresse

- Différentes fonctions permettent de manipuler les structures précédentes
- En particulier pour les traduire vers une chaîne de caractères et vice-versa
- Inclure le fichier **<arpa/inet.h>**
 - **char * inet_ntoa(struct in_addr)**
 - Traduit une adresse IPv4 en chaîne caractères
 - **int inet_aton(const char *, struct in_addr *)**
 - Met l'adresse donnée par la chaîne de caractères dans le deuxième argument
 - Renvoie 0 si l'adresse n'est pas valide
 - **Penser à tester les erreurs en C**
 - **in_addr_t inet_addr(const char*)**
 - Similaire à inet_aton

PR - API TCP C

8

Exemple

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

int main() {
    struct in_addr address;
    char *string_address;
    inet_aton("127.0.0.1", &address);
    string_address=inet_ntoa(address);
    printf("L'adresse vaut : %s\n", string_address);
    return 0;
}
```

Manipulation des structures d'adresse

- Pour les adresses IPv6, il existe des fonctions génériques :
 - **const char *inet_ntop(int af, const void *src, char * chaine, socklen_t size)**
 - **int inet_pton(int af, char * chaine, void *dest)**
- « presentation to network » and « network to presentation »
- L'entier **af** représente la famille protocolaire (**AF_INET** ou **AF_INET6**)
- **src** et **dst** sont des pointeurs vers des adresses internet conformes à la valeur de **af**
- **socklen_t** est la taille maximale que l'on peut mettre dans chaine
 - Valeur utiles **INET_ADDRSTRLEN** et **INET6_ADDRSTRLEN**
- **ATTENTION** : Ces fonctions ne font PAS appel à l'annuaire. Elles ne font que des transformations entre représentations !!!!

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 in_addr address;
    char *string_address=
        (char *)malloc(sizeof(char)*INET_ADDRSTRLEN);
    inet_pton(AF_INET, "127.0.0.1", &address);
    inet_ntop(AF_INET, &address, string_address,
        INET_ADDRSTRLEN);
    return 0;
}
```

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 in6_addr address;
    char*string_address=(char*)malloc(
        sizeof(char)*INET6_ADDRSTRLEN);
    inet_pton(AF_INET6,
        "2a00:1450:400c:c02:0:0:0:93", &address);
    inet_ntop(AF_INET6, &address, string_address,
        INET6_ADDRSTRLEN);
    return 0;
}
```

Les sockets en C

- **Une socket est un point de communication**
- Une socket est caractérisée par :
 - une adresse Internet
 - un numéro de port
 - un type de communication (UDP ou TCP) - bien entendu le type de communication est le même aux deux extrémités de la socket
- En C, une socket est représentée par le type **sockaddr**

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

```
struct sockaddr {  
    sa_family_t sa_family;  
    char sa_data[];  
};
```

Les sockets en C (2)

- Le champ **sa_family** permet de spécifier le type de la socket (et donc la structure implémentant **sockaddr**). On a les constantes suivantes :
 - **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 (AF_INET ou AF_INET6)
- En fait **struct sockaddr** est une structure générale qui est implémentée par des structures plus spécifiques (en particulier le tableau **char sa_data**)

Structures des adresse de sockets

- Type pour les sockets IPv4

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

```
struct sockaddr_in {  
    short    sin_family; // famille de socket (ex : AF_INET)  
    unsigned short sin_port; // numéro de port : htons(3490)  
    struct in_addr sin_addr; // adresse internet IPv4  
    char    sin_zero[8]; // souvent rempli avec des zéros  
};
```

Structures des adresses de sockets

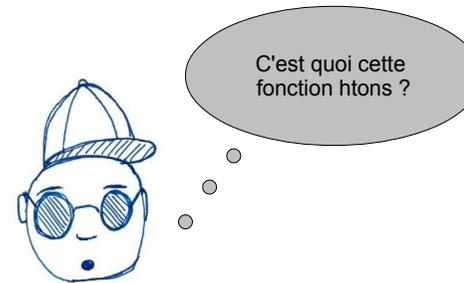
- Type pour les sockets IPv6

```
struct sockaddr_in6 {  
    u_int16_t  sin6_family; // famille de socket AF_INET6  
    u_int16_t  sin6_port; // numéro de port  
    u_int32_t  sin6_flowinfo;  
    struct in6_addr sin6_addr; // adresse IPv6  
    u_int32_t  sin6_scope_id;  
};
```

Création d'adresses de socket

```
struct sockaddr_in adress_sock;  
adress_sock.sin_family = AF_INET;  
adress_sock.sin_port = htons(3490);  
inet_aton("10.0.0.1",&adress_sock.sin_addr);
```

Codage des entiers ?



- La représentation des nombres peut être variée, or comme on communique entre machines, il est nécessaire de se mettre d'accord
- Deux grands types de codage des entiers :
 - **petit-boutiste** ou **petit-boutien (little-endian)**
 - **grand-boutiste** ou **grand-boutien (big-endian)**

Codage des entiers

- Le codage d'un entier n en base b s'écrit de la façon suivante :
 - $n = \sum c_i b^i$
- Comme les entiers utilisent des octets on peut considérer la base b comme valant 256 (c'est à dire 2^8)
- Pour un entier de 32 bits, il faut donc 4 octets et un entier n s'écrit de la façon suivante :
$$n=c3 \times 256^3 + c2 \times 256^2 + c1 \times 256 + c0$$
- En pratique on stocke dans un tableau de 4 octets les chiffres $c3, c1, c2$ et $c0$
- Ce qui change c'est l'ordre dans lesquels sont stockés dans le tableau ces 4 valeurs

Little-endian vs Big-endian

- Stockage de $n=c3 \times 256^3 + c2 \times 256^2 + c1 \times 256 + c0$
- En **big-endian**

0	1	2	3
c3	c2	c1	c0

- En **little-endian**

0	1	2	3
c0	c1	c2	c3

- **IMPORTANT** : C'est l'ordre des entiers qui changent pas l'ordre des bits dans un entier

Passer d'une machine au réseau

- Sur une machine, l'entier peut-être codé en big-endian ou little-endian (dépend du système)
- Sur le réseau, pour le protocole IP les entiers sont codés en Network Big Order (NBO) qui correspond au **big-endian**
- Il faut donc convertir les représentations des entiers

```
#include <arpa/inet.h>
```

```
uint32_t htonl(uint32_t hostlong);
```

```
uint16_t htons(uint16_t hostshort);
```

```
uint32_t ntohl(uint32_t netlong);
```

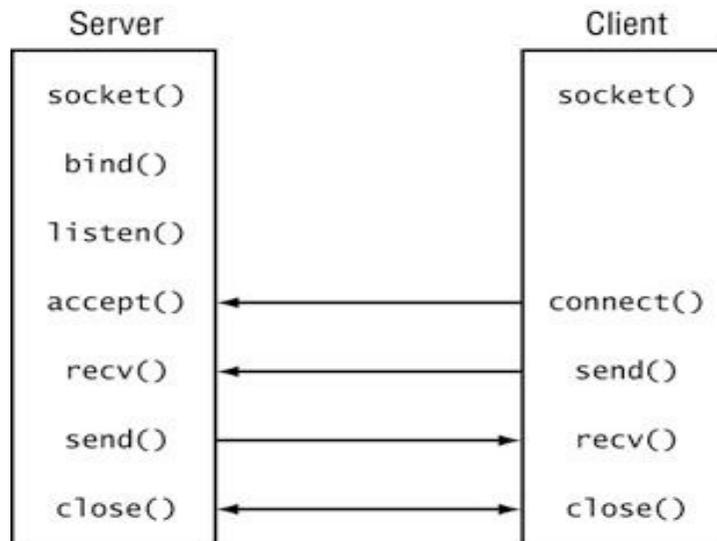
```
uint16_t ntohs(uint16_t netshort);
```

- **h** vaut pour **host** et **n** pour **network**

Passer d'une machine au réseau

- Ainsi pour passer le port numéro 3490 sur notre machine et le mettre dans la bonne représentation pour le réseau on fait :
 - **htons(3490)**
 - *host to network short* (short car le port est un entier short)
- **Nota Bene :**
 - En C il faut faire attention de bien faire les conversions pour toutes les données utilisées
 - Si votre machine est en big-endian, la conversion ne fait rien
 - Mais si vous testez le même programme sans conversion sur une autre machine, il pourrait y avoir des problèmes !!!

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 protocole de communication (mais pour TCP, on peut mettre 0 et le protocole est choisi de façon automatique)
- L'entier renvoyé sera le descripteur utilisé pour communiquer

Accès à une machine

int socket(int domaine, int type, int protocol)



Mais là on dit jamais l'adresse ou le port

- Et oui !!! On va le préciser après

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
- **int recv(int sockfd, void *buf, int len, int flags);**
 - Remplit le buffer **buf**
 - **len** est la taille maximale de **buf**
 - **flags** sera la plupart du temps mis à **0**
 - renvoie le nombre de données reçu (-1 si erreur et 0 si la connexion est fermée)
- Pour envoyer on va utiliser
- **int send(int sockfd, const void *msg, int len, int flags);**
 - Même principe que recv len est la taille en octet de msg
 - flags est aussi mis à 0 ici.
- On pourrait aussi utiliser **read** et **write**

Exemple

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

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

    int descr=socket(PF_INET,SOCK_STREAM,0);
    int r=connect(descr, (struct sockaddr *) &address_sock,
        sizeof(struct sockaddr_in));

    if(r!=-1){
        char buff[100];
        int size_rec=recv(descr,buff,99*sizeof(char),0);
        buff[size_rec]='\0';
        printf("Caracteres recus : %d\n",size_rec);
        printf("Message : %s\n",buff);
        char *mess="SALUT!\n";
        send(descr,mess,strlen(mess),0);
        close(descr);
    }
    return 0;
}
```

Tester le client précédent

- telnet est permet de simuler un client, on peut aussi utiliser un autre outil : netcat
- **netcat lulu 7** est équivalent à **telnet lulu 7**
- Mais netcat peut aussi simuler un serveur
- Si on fait **netcat -l 4242** , on a un serveur tcp qui attend une connexion sur le port 4242 et tout ce qui est tapé ensuite et envoyé au client
- Du coup vous pouvez tester le client précédent sur votre machine en lançant d'abord un terminal avec **netcat -l 4242** et dans un autre terminal vous lancez le client

Exemple

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

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

    int descr=socket(PF_INET, SOCK_STREAM, 0);
    int r=connect(descr, (struct sockaddr *) &address_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));
        close(descr);
    }
    return 0;
}
```

Pour communiquer (2)

- 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 read **nbyte** est la taille en octet de buf