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:c02:0:0:0:93**
* 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-APITCPC 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-APITCPC 8

Exemple



**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;**

**}**

PR-APITCPC 9

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, socklent\_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**
* **socklent\_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 !!!!

PR - API TCP C 10

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[];**

**};**

PR - API TCP C 13

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**)

PR - API TCP C 14

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

**};**

PR - API TCP C 15

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;** |  |

**};**

PR - API TCP C 16

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);**

PR - API TCP C 17

Little-endian vs Big-endian

* Stockage de n=c3 x 2563 + c2 x 2562 + c1 x 256 + c0
* En **big-endian**

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  |  |
| c3 | c2 | c1 | c0 |
|  |  |  |  |



* En **lillte-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

PR - API TCP C 20

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

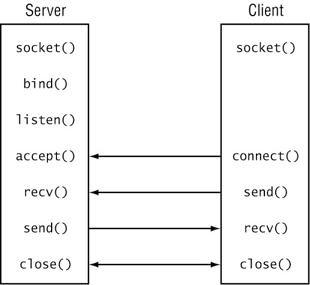
PR - API TCP C 21

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 !!!

PR - API TCP C 22

Schéma Client-Serveur en C



PR - API TCP C 23

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

PR - API TCP C 24

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);**

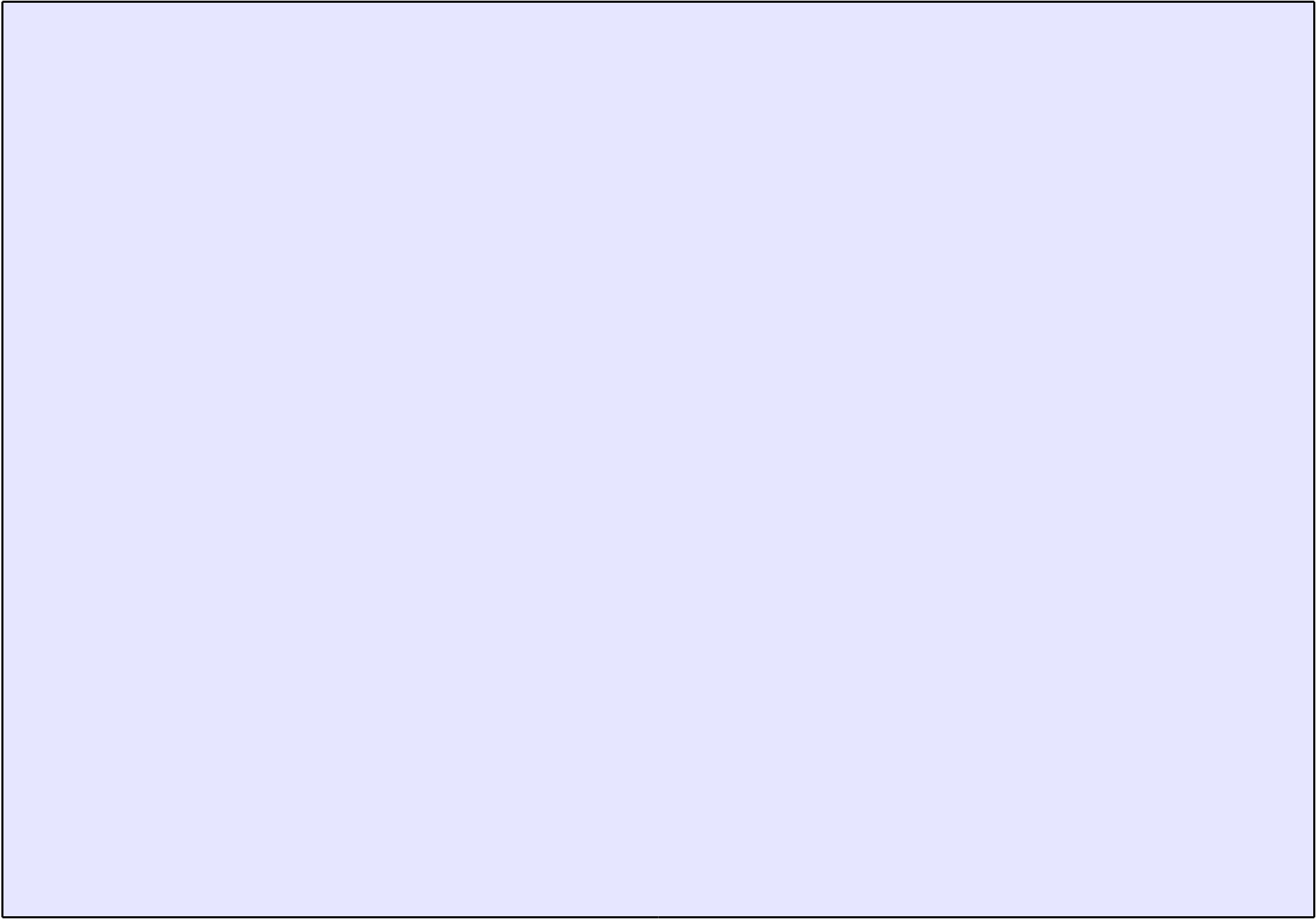
PR - API TCP C 26

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**

PR - API TCP C 27

Exemple



**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=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;**

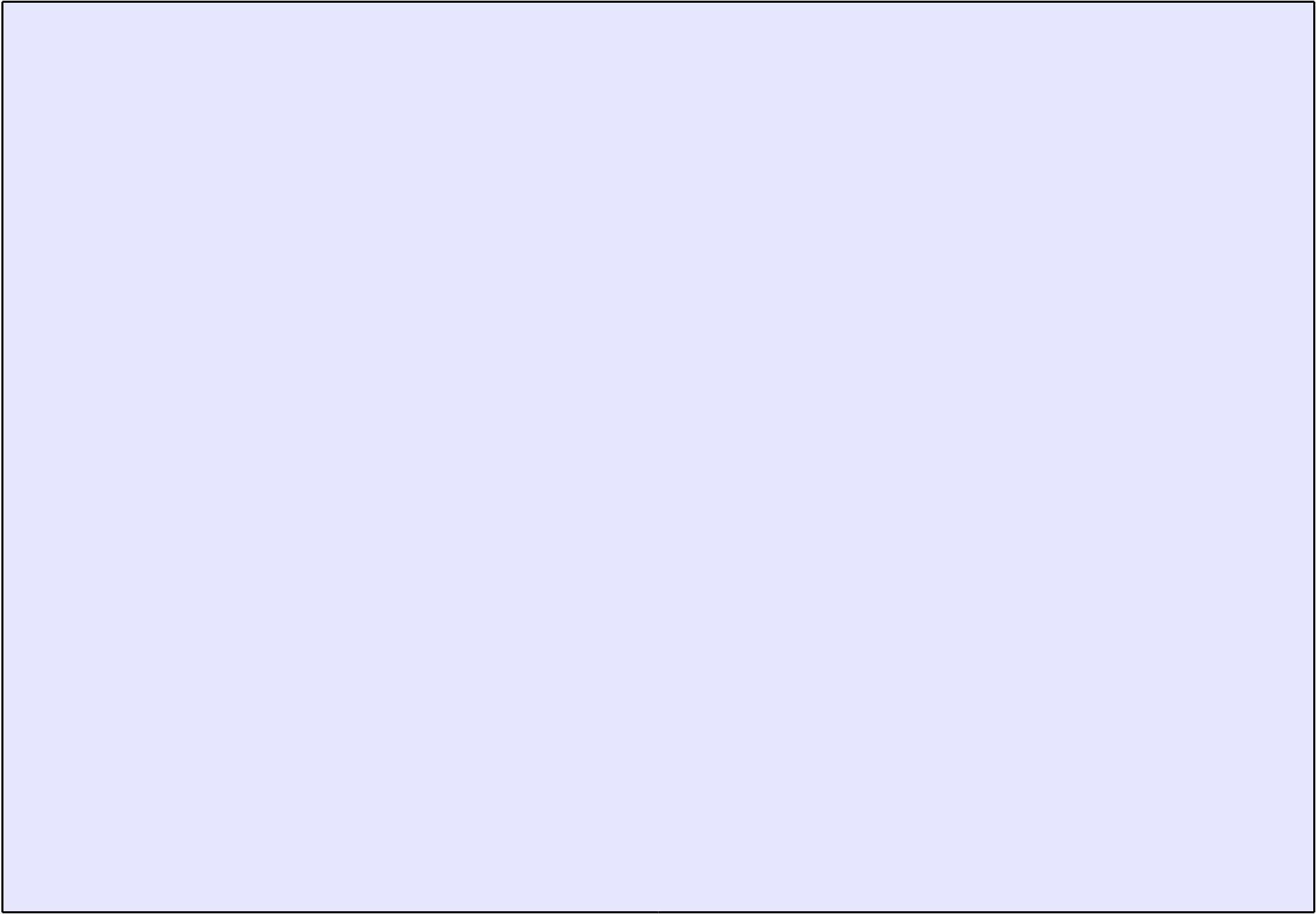
**}**

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

PR - API TCP C 30

Exemple



**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));**

**close(descr);**

**}**

**return 0;**

**}**