

### Socket Client (TCP)



En mode connecté (TCP) la socket doit être branchée de manière explicite à la partie distante pendant toute la durée des échanges de messages.

- > Demande de connexion pour le client : connect()
- > Attente de connexion pour le serveur : listen() puis accept

### Socket serveur (TCP)



### Création de socket

```
int socket (int domaine, int type, int protocole);
```

int domaine      *AF\_UNIX ou AF\_INET ou PF\_INET*

int type          *SOCK\_DGRAM ou SOCK\_STREAM*

int protocole    *0: protocol par défaut (UDP ou TCP)*

INADDR\_ANY      *adresse de la machine*

AF\_INET          *Address Family, Internet = IP Addresses*

PF\_INET          *Protocol Family, Internet = IP, TCP/IP or UDP/IP*

AF\_UNIX =        *communication entre processus au sein d'une*  
 AF\_LOCAL        *même machine*

SOCK\_STREAM     *pour sockets orientés **stream***

Côté client nous devons configurer: la partie **local** et la partie qui sera **connecté au serveur**

### Attachement socket

```
int bind (
int descripteur,                    descripteur
struct sockaddr* adr,               point -> adr attachement
int longueurAdr                    long. adr en octets
);
```

```
long inet_addr (const char* adrIP)
```

### Configuration de socket

```
struct in_addr {                    Add IP de la machine
u_long s_addr;
};
```

```
struct sockaddr_in {                Addr d'1 socket
short sin_family;                  AF_INT
u_short sin_port;                  num port
struct in_addr sin_addr;          adr IP
char sin_zero[8];                  champ de 8 '\0'
};
```

### EXEMPLE D'ATTACHEMENT

#### //Declaration

```
struct sockaddr_in adrLocale;
```

#### //Configuration

```
memset((char*)&adrLocale, '\0', sizeof(struct
sockaddr_in));
adrLocale.sin_family = AF_INET;
adrLocale.sin_addr.s_addr = htonl(INADDR_ANY);
adrLocale.sin_port = htons(numPort);
bind(descripteur, (struct sockaddr) &adrLocale,
sizeof(struct sockaddr));
```

### Fonctions d'utilisation de socket

```
int connect (                        même prototye que bind
descripteurSocket,                  int descripteurSocket
&adrServeur,                        struct sockaddr* adrServeur
longueur_adr                        int longueur_adr
);
```

```
int listen (
descSocketEcoute,                   int descripteurSocketEcoute
nbPendantes                        int nbMax de connexions pendantes
);
```

```
int accept (
descSockEcoute,                    int descripteurSockEcoute
&servStore,                        struct sockaddr* servStore
longueurAdr                        int longueurAdr
);
```



### Fonctions d'utilisation de socket (cont)

```
int send = (
    descripteurSocket,      int descripterSocket
    message,                char message[]
    strlen(message),       int longueurMessage
    option                  0
);

int recv (
    descripteurSocket,     int descripterSocket
    message,               char message[]
    strlen(message),       int longueurMaxiMsg
    option                  0
);

int shutdown(
    descripteurSocket,     int descSocket
    sens                    0: lect; 1:ecrit; 2: les deux
);

int close(int descripteur);
```

### Inclusions bibliothèques nécessaires

```
#include <sys/socket.h>      socket() et bind()
#include <netinet/in.h>      in_addr et sockaddr_in
#include <arpa/inet.h>       htonl, htons,inet_addr
#include <unistd.h>          close()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <netdb.h>
```

### Spécificateur de type lors de printf ou scanf

%d	int
%s	string
%C	char
%f	float
%p	pointer
%u	unsigned int

### Fonctions de conversion

uint32_t htonl(uint32_t hostlong);	host to network long
uint16_t htons(uint16_t hostshort);	host to network short
uint32_t ntohl(uint32_t netlong);	network to host long
uint16_t ntohs(uint16_t netshort);	network to host short

### Squelette Client

```
int main() {
    //DECLARATION VARIABLES
    int clientSocket, connexion, reception, envoie,
    deconnexion;
    struct sockaddr_in serverAddr;
    struct sockaddr_in localAddr;
    socklen_t addr_size;
    char requete[] = "test";
    char reponse[1024];
    long port = 10500;
    long ip = inet_addr("193.50.225.215");
    //TRAITEMENT

    // Création de la socket
    printf("Creation de la socket.....\n");
    clientSocket = socket(PF_INET, SOCK_STREAM, 0);

    // Configuration des parametres de ma struct de
    l'adresse locale
    printf("Configuration de la struct d'adresse
    locale\n");
```

### Squelette Client (cont)

```
memset(localAddr.sin_zero, '\0',
sizeof(localAddr.sin_zero));
localAddr.sin_family = AF_INET;
localAddr.sin_port = htons(0); //port locale
localAddr.sin_addr.s_addr = htonl (INADDR_ANY); //Adr.
locale
bind(clientSocket, (struct sockaddr) &localAddr,
sizeof(struct sockaddr)); //Association IP et port*
// Configuration des parametres de la struct de
l'adresse du serveur
printf("Configuration de la struct d'adresse
serveur\n");
memset(serverAddr.sin_zero, '\0',
sizeof(serverAddr.sin_zero));
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port); //port du serveur
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
// Adr. du serveur
//Connexion du serveur
addr_size = sizeof serverAddr;
connect(clientSocket, (struct sockaddr *) &serverAddr,
//Boucle de communication avec le serveur
while(!fini) {
    //Envoie de la requete au serveur
    send(clientSocket, requete, strlen(requete), 0);
    //Attente de la réponse du serveur
    recv(clientSocket, reponse, strlen(reponse), 0);
}
// Deconnexion et fermeture de la socket
shutdown(clientSocket, 2);
close(clientSocket);
return 0;
}
```

### Squelette serveur

```
int main(){
//DECLARATION
int socketEcoute, socketService, attachement,
connexion, reception, envoie, deconnexion, fermeture;
int dialogueClient = 1;
char reponse[1024];
struct sockaddr_in serverAddr;
struct sockaddr_storage serverStorage;
socklen_t addr_size;
long port = 10500;
//TRAITEMENT
// Creation de la socket ecoute
printf("Creation de socketEcoute....\n");
socketEcoute = socket(PF_INET, SOCK_STREAM, 0);
// Configuration des parametres de la struct de
l'adresse du serveur
memset(serverAddr.sin_zero, '\0', sizeof
serverAddr.sin_zero);
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
//Attachement
printf("Attachement de socketEcoute...\n");
attachement = bind(socketEcoute, (struct sockaddr *)
&serverAddr, sizeof(serverAddr));
//Mise à l'ecoute (maxi 5 connections)
if(listen(socketEcoute,5)==0)
    printf("Mise a l'ecoute....\n\n");
else
    printf("Erreur de mise a l'ecoute\n\n");
//Attente connexion
while(1) {
addr_size = sizeof serverStorage;
socketService = accept(socketEcoute, (struct sockaddr
*) &serverStorage, &addr_size);
//Dialogue avec le client
```

### Squelette serveur (cont)

```
while(1) {
//Lecture de la requete du client
//Envoie de la reponse au client
}
// Terminaison et fermeture socket de service
shutdown(socketService, 2);
}
//Fermeture de la socket d'ecoute
close(socketService);
return 0;
}
```

### Fonctions en C

memset(char* chaine, char car, int n)	Retourne la <b>chaine</b> avec <b>char</b> au debut <b>n</b> fois
int toupper(str[i])	Mettre le car en param. en majuscule
int tolower(str[i])	Mettre le car en param. en minuscule
int strlen(str)	retourne longueur de la chaîne
int strcmp(str1, str2)	Valeurs de retours: < 0 si str1 < str2 > 0 si str1 > str2 = 0 si str1== str2
strcpy(str1, str2)	Copier contenu de str2 dan str1

### Exemple de fonction en c:

```
int chercherIndexCar(char car, char jeu[], int
tailleJeu) {
int colonne = -1;
int c = 0;
while(c < tailleJeu) {v
if (jeu[c] == car) {
colonne = c;
break;
}
c++;
}
return colonne;
}
```

### Exemple d'appel de fonction en c:

```
int index = chercherIndexCar(chercherIndexCar(car,
jeu, nbLignes*nbColonnes);
```

### Fonctions, Procédures et passage de paramètre

void proc(char chaine[])	chaine est passé par adresse
{...}	
void proc(int num) {...}	num est passé par valeur
void proc(int *num) {...}	num est passé par adresse

### Declaration

```
void afficher(int lignes, int colonnes, char chaine[]
[colonnes]){...}
char morpion[3][3];
```

### Appel

```
afficher(3, 3, morpion);
```

### Fonction d'affichage de tableau di-dim:

```
void afficher(int lignes, int colonnes, char chaine[]
[colonnes]) {
int i = 0;
int j = 0;
while (i < lignes) {
while (j < colonnes) {
printf("%c", chaine[i][j]);
j++;
}
printf("\n");
i++;
j = 0;
}
}
```