

# Applications Client/Serveur

## Type non connecté

## ➡ Le descripteur de socket.

C'est un entier retourné par le noyau au processus à l'issue de l'appel système `socket`. Le processus passera ce descripteur à tous les appels système qui lisent/écrivent sur le réseau. Exemples :

```
int sfd;  
sfd=socket(AF_INET, SOCK_DGRAM,0); // Pour obtenir un descripteur de socket utilisant IPv4 en mode non connecté.  
ou  
sfd=socket(AF_INET6, SOCK_DGRAM,0); //Pour obtenir un descripteur de socket utilisant IPv6 en mode non connecté.
```

## ➡ Les adresses de socket.

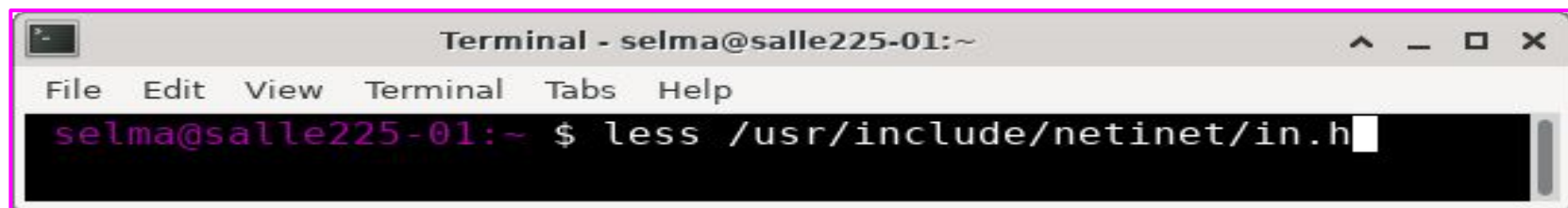
Une adresse de socket est une structure de données qui portent des informations sur un point qui communique ou qui peut communiquer sur le réseau. Retenons que c'est essentiellement un numéro de port et une adresse IP.

*`struct sockaddr_in`* est le type décrivant une adresse de socket Internet IPv4 (famille AF\_INET).

*`struct sockaddr_in6`* est le type décrivant une adresse de socket Internet IPv6 (famille AF\_INET6).

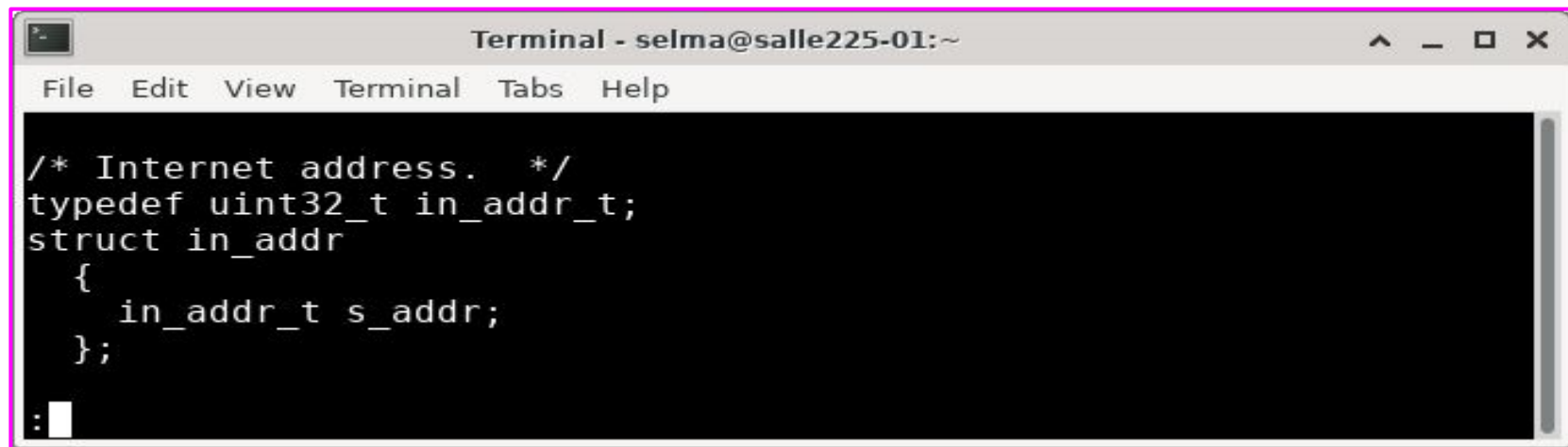
*`struct sockaddr_storage`* est le type d'adresse de socket utilisé dans les applications IPv4/IPv6 indépendantes.

*`struct sockaddr`* est le type générique d'adresse de socket (pas seulement Internet).



A terminal window titled "Terminal - selma@salle225-01:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command `less /usr/include/netinet/in.h` is entered at the prompt.

```
Terminal - selma@salle225-01:~
File Edit View Terminal Tabs Help
selma@salle225-01:~ $ less /usr/include/netinet/in.h
```



The same terminal window now displays the contents of the `netinet/in.h` header file. The visible text includes a comment, a typedef, and a struct definition for an internet address.

```
Terminal - selma@salle225-01:~
File Edit View Terminal Tabs Help

/* Internet address.  */
typedef uint32_t in_addr_t;
struct in_addr
{
    in_addr_t s_addr;
};

:
```



```
Terminal - selma@salle225-01:~
File Edit View Terminal Tabs Help
/* IPv6 address */
struct in6_addr
{
    union
    {
        uint8_t __u6_addr8[16];
        uint16_t __u6_addr16[8];
        uint32_t __u6_addr32[4];
    } __in6_u;
};
```

```
Terminal - selma@salle225-01:~
File Edit View Terminal Tabs Help
/* Ditto, for IPv6. */
struct sockaddr_in6
{
    __SOCKADDR_COMMON (sin6_);
    in_port_t sin6_port; /* Transport layer port # */
    uint32_t sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr; /* IPv6 address */
    uint32_t sin6_scope_id; /* IPv6 scope-id */
};
```

```
Terminal - selma@salle225-01:~
File Edit View Terminal Tabs Help
selma@salle225-01:~ $ less /usr/include/bits/socket.h
```

```
Terminal - selma@salle225-01:~
File Edit View Terminal Tabs Help

/* Structure describing a generic socket address.  */
struct sockaddr
{
    __SOCKADDR_COMMON (sa_);    /* Common data: address family and length.  */
    char sa_data[14];          /* Address data.  */
};

/* Structure large enough to hold any socket address (with the historical
   exception of AF_UNIX).  */
#define __ss_aligntype  unsigned long int
#define __SS_PADSIZE \
    (__SS_SIZE - __SOCKADDR_COMMON_SIZE - sizeof (__ss_aligntype))

struct sockaddr_storage
{
    __SOCKADDR_COMMON (ss_);    /* Address family, etc.  */
    char __ss_padding[__SS_PADSIZE];
    __ss_aligntype __ss_align; /* Force desired alignment.  */
};

:
```

## ➡ Une programmation des sockets avec indépendance IPv4/IPv6.

On utilisera seulement le type `struct sockaddr_storage` pour les adresses de socket.

On utilisera la fonction `getaddrinfo` pour préparer des adresses de socket destinées aux appels systèmes *bind*, *connect*.

On utilisera la fonction `getnameinfo` pour convertir des adresses de socket en un nom/adresse de host et un nom/numéro de service, afin de les afficher par exemple pour une lecture humaine.

## ➡ L'appel système *bind*.

C'est via cet appel que le lien est fait dans le noyau entre un descripteur de socket (référence locale au processus) et une adresse de socket, c'est à dire un numéro de port et une adresse IP de la machine (du host) qui tourne le processus appelant. A l'issue d'un appel *bind* réussi, le port correspondant est "ouvert" dans la machine qui tourne le processus.

## ➡ Séquence des appels système côté serveur.

*getaddrinfo* pour obtenir une liste d'adresses de sockets valides avec une adresse IP du host et un numéro de service, avec possibilités d'utiliser des critères qui limitent l'ensemble des adresses de sockets.

Tester chaque cellule de la liste retournée par *getaddrinfo* jusqu'à ce que *socket* et *bind* réussissent simultanément.

```
Terminal - selma@blacksail:~/IS3/TP/NET_PGRMS
File Edit View Terminal Tabs Help
int main(int argc, char *argv[]){
    int n,l,ll,r,sfd;

    struct addrinfo criteria;
    struct addrinfo s,*res;
    struct sockaddr_storage scksrv,sckclt; //To hold ipv4/6 independent socket addresses

    char host[NI_MAXHOST], service[NI_MAXSERV],buf[BFSZ];

    if (argc < 2){
        fprintf(stderr,"Usage: %s <port_num>\n",argv[0]);
        exit(1);
    }

    memset(&criteria,0,sizeof(struct addrinfo));

    criteria.ai_family=AF_UNSPEC; /* Allow IPv4 or IPv6 */
    //criteria.ai_family=AF_INET6;
    criteria.ai_socktype=SOCK_DGRAM; //socket with DGRAM type only
    criteria.ai_protocol=0; /* Any protocol */
    //criteria.ai_flags=AI_NUMERICSERV ;
    criteria.ai_flags=AI_PASSIVE|AI_NUMERICSERV ;
    criteria.ai_canonname=NULL;
    criteria.ai_addr=NULL;
    criteria.ai_next=NULL;

    r=getaddrinfo(NULL,argv[1],&criteria,&res);
    :
```



```
Terminal - selma@blacksail:~/IS3/TP/NET_PGRMS
File Edit View Terminal Tabs Help

int sock_bind(struct addrinfo *res,struct addrinfo *s){
    int r,sfd=-1;
    while (res!=NULL){
        sfd=socket(res->ai_family,res->ai_socktype,res->ai_protocol);
        if (sfd>0){
            r=bind(sfd,res->ai_addr,res->ai_addrlen);
            if (r==0){//bind succeeds
                *s=*res;
                break;
            }
            else {
                close(sfd);
                sfd=-1;
            }
        }
        res=res->ai_next;
    }
    return(sfd);
}
```

## ➡ Séquence des appels système côté client.

```
Terminal - selma@blacksail:~/IS3/TP/NET_PGRMS
File Edit View Terminal Tabs Help
int main(int argc, char *argv[]){
    int n,r,sfd;
    socklen_t l,ll;
    unsigned int qry;

    struct addrinfo criteria;
    struct addrinfo s,*res;
    struct sockaddr_storage scksrv,sock_peer;

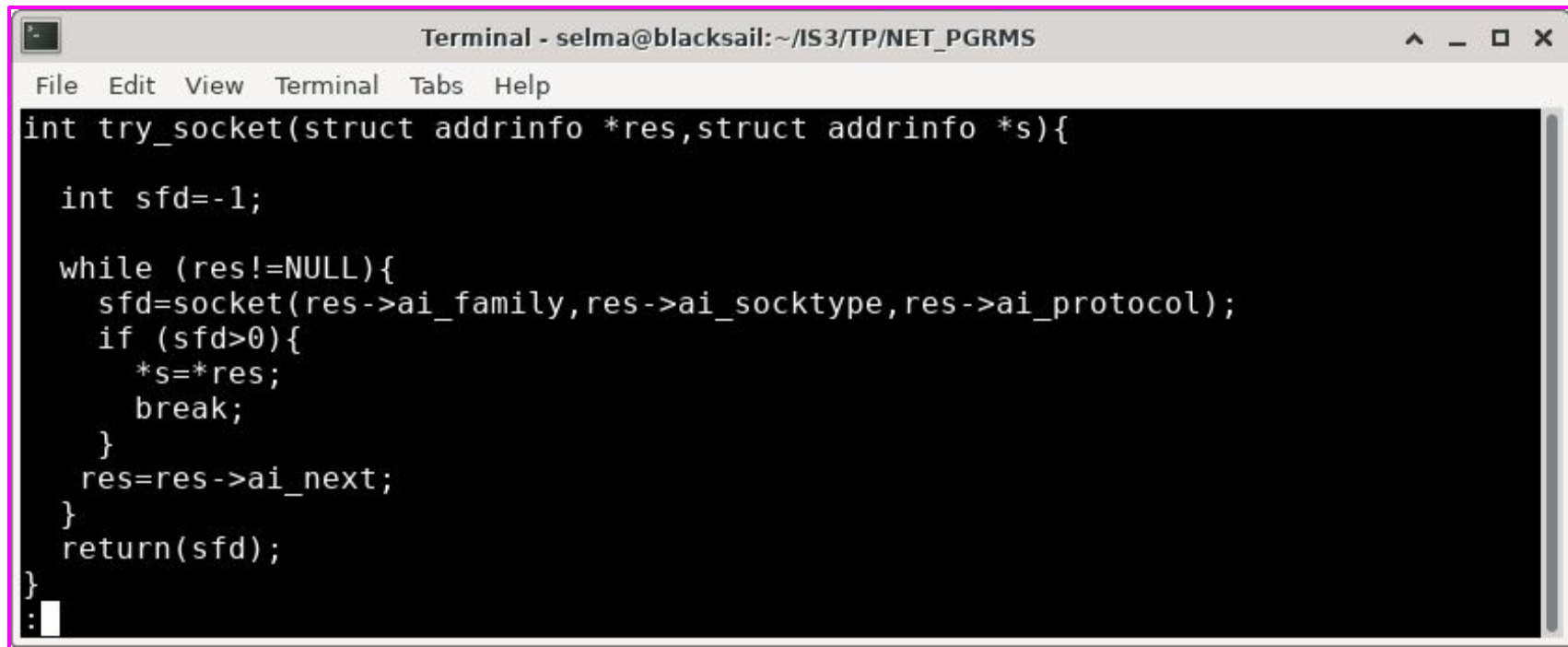
    char host[NI_MAXHOST], service[NI_MAXSERV],buf[BFSZ];

    if (argc < 3){
        fprintf(stderr,"Usage: %s <SERVER> <port_service>\n",argv[0]);
        exit(1);
    }

    memset(&criteria,0,sizeof(struct addrinfo));

    criteria.ai_family=AF_UNSPEC; /* Allow IPv4 or IPv6 */
    //criteria.ai_family=AF_INET6;
    criteria.ai_socktype = SOCK_DGRAM; //socket with DGRAM type only
    criteria.ai_protocol = 0; /* Any protocol */
    criteria.ai_flags = AI_NUMERICSERV ;
    criteria.ai_canonname = NULL;
    criteria.ai_addr = NULL;
    criteria.ai_next = NULL;

    r=getaddrinfo(argv[1],argv[2],&criteria,&res);
    :
```

A terminal window with a title bar "Terminal - selma@blacksail:~/IS3/TP/NET\_PGRMS". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area is black with white text showing a C function definition for try\_socket. The code is as follows:

```
int try_socket(struct addrinfo *res,struct addrinfo *s){  
    int sfd=-1;  
    while (res!=NULL){  
        sfd=socket(res->ai_family,res->ai_socktype,res->ai_protocol);  
        if (sfd>0){  
            *s=*res;  
            break;  
        }  
        res=res->ai_next;  
    }  
    return(sfd);  
}
```

The cursor is at the end of the last line, indicated by a white prompt character ":".

Après *socket/bind* réussi, le serveur entre dans une boucle où il fait *recvfrom*, suivi éventuellement de *sendto*.

Exemple.

```
#define BFSZ 1024
```

```
int l, ll;
```

```
struct sockaddr_storage sckclt ; char buf[BFSZ];
```

```
memset(buf,0,BFSZ);
```

```
l=sizeof(struct sockaddr_storage);
```

```
n=recvfrom(sfd,buf,BFSZ,0,(struct sockaddr*)&sckclt,&ll);
```

Après *socket* réussi, le client fait un ou plusieurs *sendto* chacun suivi ou pas par d'un *recvfrom*.

Exemple.

```
#define BFSZ 1024
```

```
struct addrinfo s, *res;
```

```
getaddrinfo(argv[1],argv[2],&criteria,&res);
```

```
sfd=try_socket(res,&s);
```

```
memset(buf,0,BFSZ);
```

```
//Préparer buf
```

```
n=sendto(sfd,buf,strlen(buf),0,s.ai_addr,s.ai_addrlen);
```