

# Interface des sockets

Patrick Félix  
Année 2001-2002  
I.S.T / IUT Bordeaux I

# A quoi ça sert les sockets ?

---

## # Applications client/serveur

- Transfert de fichiers
- Connexion à distance
- Courrier électronique
- Groupe de discussions
- Web
- Et maintenant tous les domaines de l'informatique:
  - Multimedia, jeux, télémaintenance...

## # Comment implémenter de telles applications

- Interface des sockets

# Les sockets

---

- # Mécanisme de communication inter-processus
- # Bref historique:
  - UNIX BSD4.2 (Berkeley Software Distribution)
  - Windows de Microsoft, UNIX System V...
- # Interface applicative
  - Ensemble de primitives permettant des E/S avec la pile de protocole TCP/IP:
    - socket, connect, write, read, close, bind, listen, accept...
  - Interface entre couches application et transport
  - Adaptée aux protocoles TCP et UDP, mais utilisable avec d'autres familles de protocoles

# Communication sous UNIX

---

## # Rappels :

- tube (pipe) du système UNIX
- tube nommé

## # Sockets=généralisation des tubes nommés

- Point de communication bidirectionnelle par lequel un processus pourra émettre ou recevoir des informations
- processus sur des machines (éventuellement) différentes
- systèmes d'exploitation (éventuellement) différents

# Attributs des sockets

---

## # un nom

- Descripteur de fichier

## # un type

- SOCK\_STREAM: mode connecté, remise fiable (TCP/IP)
- SOCK\_DGRAM: mode non connecté, remise non fiable (UDP/IP)
- RAW : mode caractère (pour accès direct aux couches inférieures)

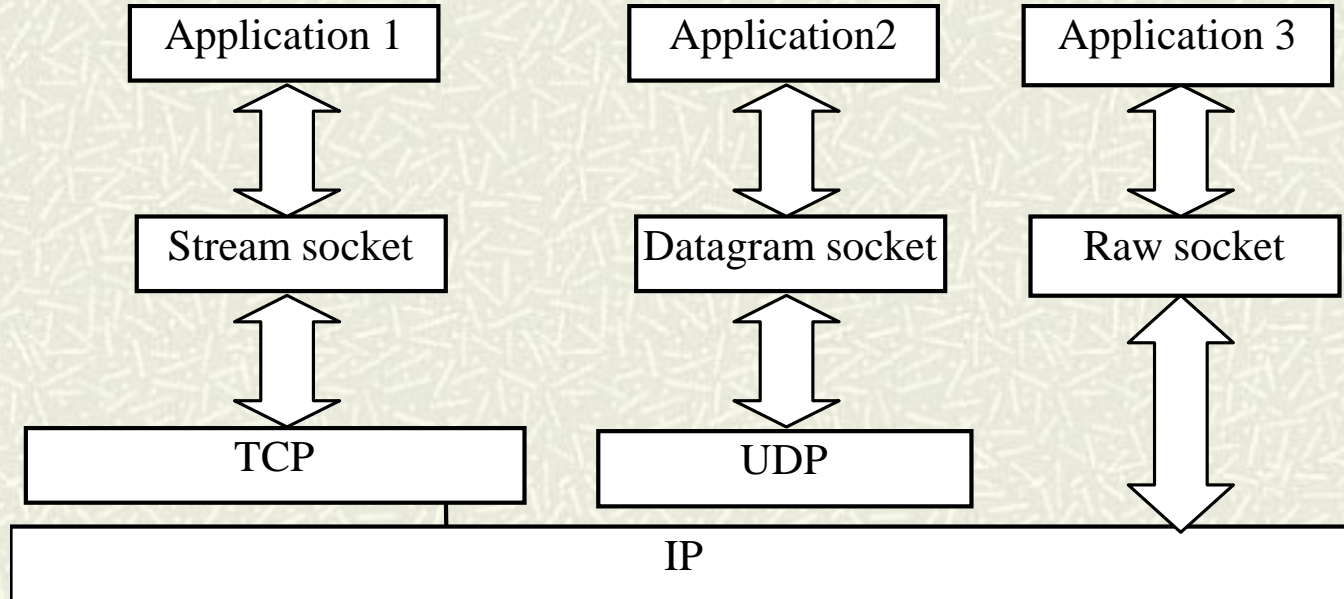
## # associé à un processus

## # une adresse (adresse IP + n° port)

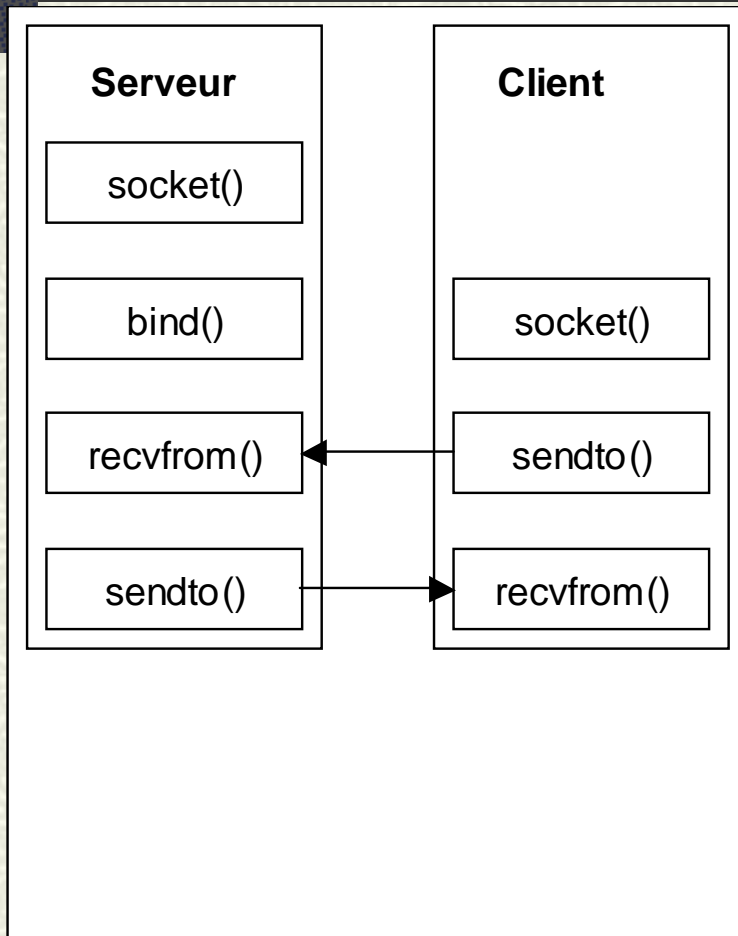
# Interface des sockets

## Partie I : Généralités

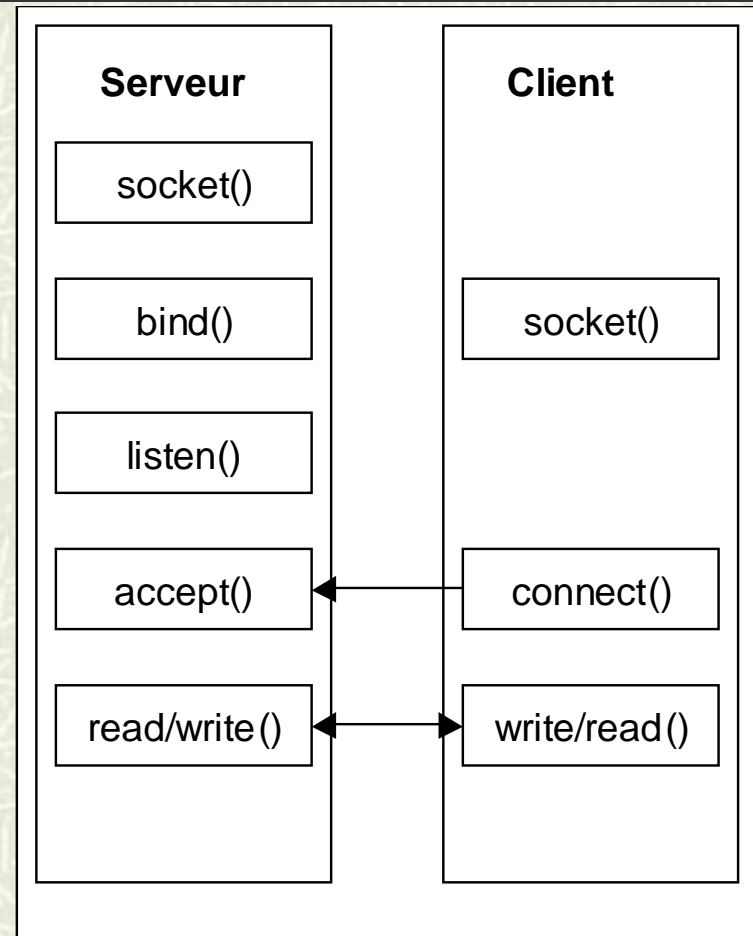
# Types de socket et protocoles



# Modes de dialogue et primitives



*Dialogue client/serveur en mode datagramme*



*Dialogue client/serveur en mode connecté*



# Structure de données

```
#include <sys/types.h>      //Bibliothèques requises
#include <sys/socket.h>

struct sockaddr {
    unsigned short sa_family; //famille de protocole pour cette adresse
    char sa_data[14];        // 14 octets d'adresse
}

struct sockaddr_in {        // _in pour Internet
    short sin_family;       //famille de protocole pour cette adresse
    u_short sin_port;       //numéro de port (0=port non utilisé)
    struct in_addr sin_addr; //adresse IP
    char sin_zero[8];       //non utilisé
}

struct in_addr {
    u_long s_addr; //soit 4 octets : bien pour une adresse IP !
};
```

# Création & fermeture

## # `int socket(int af, int type, int protocole)`

- création de la structure de donnée (appelée socket) permettant la communication,
- af = famille de protocole (TCP/IP, ou d'autres...)
  - AF\_INET : domaine Internet (domaine que nous utiliserons)
  - AF\_UNIX : domaine UNIX (pour donner un autre exemple)
- type = SOCK\_STREAM, SOCK\_DGRAM, RAW
- Protocole : 0 pour protocole par défaut (voir <netinet/in.h>)
- socket() retourne
  - un descripteur de socket
  - -1 si erreur

## # `close(int socket)`

- Ferme la connexion et supprime la structure de données associée à la socket

## # `Shutdown(int socket, int how)`

- how: 0/1/2 pour réception interdite/émission interdite/réception&émission interdite

# Spécification d'adresse locale

```
# int bind(int socket, struct sockaddr * adresse-  
locale, int longueur-adresse)
```

- Associe un numéro de port et une adresse locale à une socket, retourne `-1` si erreur.
- `socket` = descripteur de socket
- `adresse-locale` = structure qui contient l'adresse (adresse IP + n° de port)
- `adresse-locale: struct sockaddr *`
  - `sockaddr_un` si `AF_UNIX`
  - `sockaddr_in` si `AF_INET` (adresse IP)
- `longueur adresse` : `sizeof(struct sock_addr)`
- Si `sin_port=0` : choix d'un numéro de port non utilisé
- Si `sin_addr.s_addr= INADDR_ANY`: utilisation de l'adresse IP de la machine

# Diverses primitives utiles...

# **struct hostent gethostbyname(char \*name)**

- pour traduire un nom de domaine en adresse IP

```
struct hostent *h;
```

```
h=gethostbyname("stargate.ist.ga");
```

```
printf("adresse IP: %s\n",
```

```
        inet_ntoa(*(struct in_addr *)h->h_addr));
```

# **getservbyname()**

- pour traduire en n° de port le nom d'un service

# **getsockname(int desc, struct sock\_addr \* p\_adr, int \* p\_longueur)**

- pour récupérer l'adresse effective d'une socket (après bind)

## # Conversion Network Byte Order (68000) – Host Byte Order (Intel)

- `htons()`: 'Host to Network Short'
- `htonl()`: 'Host to Network Long'
- `ntohs()`: 'Network to Host to Short'
- `ntohl()`: 'Network to Host to Long'

## # ATTENTION: toujours mettre les octets dans l'ordre 'Network Order' avant de les envoyer sur le réseau

## # `in_addr inet_addr(char *)`

- Convertit une adresse 'ASCII' en entier long signé (en Network Order)
- `socket_ad.sin_addr.s_addr = inet_addr("172.16.94.100")`

## # `char * inet_ntoa(in_addr)`

- Convertit entier long signé en une adresse 'ASCII'  

```
char *ad1_ascii;  
ad1_ascii=inet_ntoa(socket_ad.sin_addr),  
printf("adresse: %s\n",ad1_ascii);
```

# Interface des sockets

## Partie II : Mode datagramme

# Lecture-Ecriture en mode datagramme (UDP)

---

- **int sendto( )**

- Permettent l'envoi de datagrammes en mode non-connecté
- Contient adresse du destinataire (sendto)

- **int recvfrom( )**

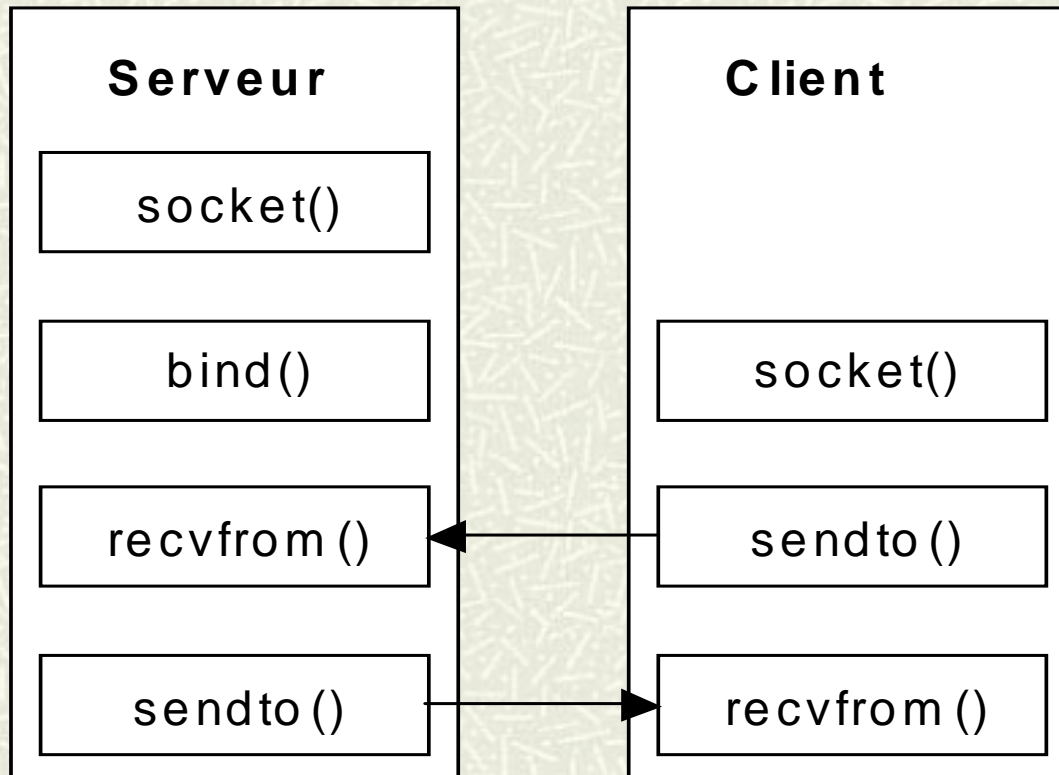
- réception (bloquante)
- Contient adresse de l'émetteur

# sendto et recvfrom : syntaxe

```
int sendto(
    int socket,          // descripteur socket
    void *tampon,       // message à envoyer
    int longueur,       // longueur du message
    int drapeaux,       // 0
    struct sockaddr *p_adresse, //adresse destinataire
    int longueur_adresse //longueur adresse
};
int recvfrom(
    int desc,           // descripteur socket
    void *tampon,      // zone pour recevoir le message
    int longueur,      // longueur de la zone réservée
    int option,        // 0 ou MSG_PEEK
    struct sockaddr *p_adresse, //adresse émetteur
    int *longueur_adresse //longueur adresse
};
```



# Dialogue (datagramme)



# Protocoles « application » utilisant UDP

---

## # Un protocole non normalisé : BigBen

C:HEURE

S:HH

S:MM

S:SS

## # Un schéma classique (mais non général):

- Envoi d'une requête par un client
- Réponse du serveur

## # Un protocole normalisé:

- TFTP (port : 69)
- Voir prochains cours sur les applications...



# Source Socket Mode Datagramme

# Interface des sockets

## Partie III : Mode connecté

# Communication en mode connecté

---

- # Dissymétrie lors de la connexion
  - Le serveur attend...
  - Le client demande une connexion
- # Symétrie dans l'échange d'informations
  - Le client ou le serveur peut
    - envoyer/recevoir des informations
    - Demander la fin de la connexion
- # Echange d'un flot continu de caractères
  - Pas de structure en message

# Connexion

---

- # `connect (socket, adr-destination, longueur-adr)`
  - Côté client
  - Pour établir une connexion TCP avec le processus serveur
  - L'adresse IP et le numéro de port sont spécifiés
  - Appel bloquant jusqu'à la fin de la prise en compte de la connexion par le serveur  
(configuration par défaut, peut-être modifiée...)

# Création d'une file d'attente

---

**# listen(int socket, int lgr-file)**

- Côté serveur
- crée une file d'attente pour les demandes de connexion
- Place la socket en 'mode connexion'
- lgr-file indique le nombre maximal de demandes de connexion autorisées dans la file (5, 10 ou 20)
- file d'attente exploitée par la primitive accept.

# Acceptation d'une connexion

# **newsock** = **accept** (socket, adresse, lgr-adresse)

- côté serveur
- prise en compte d'une demande de connexion entrante sur une socket de 'connexion'.
- primitive bloquante
- newsock : nouveau descripteur de socket sur laquelle s'effectuera l'échange de données
- adresse : adresse du client.
- Le processus peut traiter lui-même la nouvelle connexion, puis revenir à accept, ou bien se répliquer (fork() en UNIX) pour la traiter, le processus père étant toujours à l'écoute.



# Lecture-Ecriture TCP

---

- **write**(socket, tampon, longueur )  
**read**(socket, tampon, longueur)
  - Envoie/reçoit des données sur une connexion TCP
  - Plus besoin de l'adresse émetteur/destinataire !
- **send**(socket, tampon, longueur, drapeaux)  
**recv**(socket, tampon, longueur, drapeaux)
  - Idem, avec drapeaux permettant l'utilisation d'options

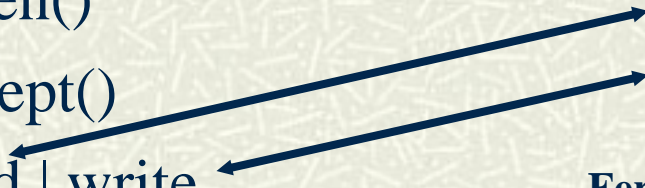
# Exemple de dialogue (connecté)

## # Sur le serveur

- socket()
- bind() : nommage
- listen()
- accept()
- read | write

## # Sur le client

- Créer une socket (socket)
- Connecter la socket au serveur (connect)
- Tant que pas fini
  - envoyer une requête (write)
  - lire la réponse (read)
  - traiter la réponse
- Fermer la socket (close)



# Telnet:un client universel

```
$>telnet stargate.ist.ga
```

```
User: felix
```

```
Password:
```

```
$>telnet stargate.ist.ga 110
```

```
+OK Qpopper (version 4.0.4) at stargate starting.
```

```
■ USER felixp
```

```
+OK Password required for felixp.
```

```
■ PASS Devine!!!
```

```
+OK felixp has 3 visible messages (0 hidden) in 4235 octets.
```

```
■ STAT
```

```
+OK 3 4235
```

■ *LIST*

*+OK 3 visible messages (4235 octets)*

*1 1508*

*2 1464*

*3 1263*

*.*

■ *RETR 3*

*+OK 1263 octets*

*Return-Path: <felixp@ist.ga>*

*Received: from komet ([172.16.94.87])*

*by stargate.ist.ga (8.12.3/8.12.3) with SMTP id  
g4S9hEpN004334*

*for <felixp@ist.ga>; Tue, 28 May 2002 10:43:20*

*+0100*

*Message-ID: <008201c2062b\$4c266420\$575e10ac@ist.ga>*

*Reply-To: "Patrick FELIX" <felix@labri.fr>*

*From: "Patrick FELIX" <felixp@ist.ga>*

*To: <felixp@ist.ga>*

*Subject: un message*

*Date: Tue, 28 May 2002 10:37:29 +0100*

*.../...*

*X-Mailer: Microsoft Outlook Express 6.00.2600.0000*

*C'est bientôt l'heure, alors "Bonne appetit " et tout a l'heure.*

*Patrick FELIX.*

-----  
*LaBRI - Universit  Bordeaux I 351 crs de la Lib ration - F-33405 TALENCE*  
*Tel. +33 5 56 84 69 15 - Fax. +33 5 56 84 66 69 - M l. felix@labri.fr*  
-----

*IUT Bordeaux 1 - D partement Informatique*  
*Domaine Universitaire - F-33405 TALENCE*  
*Tel. +33 5 56 84 58 19 - Fax. +33 5 56 84 58 86*  
*M l felix@info.iuta.u-bordeaux.fr*  
-----

*.*

*■ DELE 2*



# Source Socket Mode connecté

# Bigben–Serveur

```
int main(int argc, char * argv[])
{
    int fdTravail, port;
    ...
    /* initialisation du service */
    port=atoi(argv[1]);
    fd=init_service(port);
    /* gestion des connexions de clients */
    while(1) {
        /* acceptation d'une connexion */
        fdTravail=accept(fd,NULL,NULL);
        if (fdTravail<=0) FATAL("accept");

        if (fork()==0) { /* fils : gestion du dialogue avec client */
            close(fd);
            travail_fils(fdTravail);
            close(fdTravail);
            exit(0);
        }
        else { /* pere : repart a l'ecoute d'une autre connexion */
            close(fdTravail);
        }
    }
}
```

```
int init_service(int port)
{
    int fdPort;
    struct sockaddr_in addr_serveur;
    socklen_t lg_addr_serveur = sizeof addr_serveur;

    /* creation de la prise */
    fdPort=socket(AF_INET,SOCK_STREAM,0);
    if (fdPort<0) FATAL("socket");
    /* nommage de la prise */
    addr_serveur.sin_family      = AF_INET;
    addr_serveur.sin_addr.s_addr = INADDR_ANY;
    addr_serveur.sin_port        = htons(port);
    if (bind(fdPort,(struct sockaddr *)&addr_serveur, lg_addr_serveur) < 0)
        FATAL("bind");
    /* Recuperation du nom de la prise */
    if (getsockname(fdPort,(struct sockaddr *)&addr_serveur,
        &lg_addr_serveur) < 0)
        FATAL("getsockname");
    /* Le serveur est a l'ecoute */
    printf("Le serveur ecoute le port %d\n",ntohs(addr_serveur.sin_port));
    /* ouverture du service */
    listen(fdPort,4);
    return fdPort;
}
```



```
void travail_fils(int fdTravail)
{
    long horloge;
    struct tm *temps;
    char tampon[2];
    int h,m,s;

    /* preparation de la reponse */
    time(&horloge);
    temps=localtime(&horloge);
    h = temps->tm_hour;
    m = temps->tm_min;
    s = temps->tm_sec;

    /* envoi de la reponse */
    sprintf(tampon, "%02d", h);
    write(fdTravail,tampon,2);
    sprintf(tampon, "%02d", m);
    write(fdTravail,tampon,2);
    sprintf(tampon, "%02d", s);
    write(fdTravail,tampon,2);
}
```

# Bigben-Client

```
...  
  
int main(int argc, char * argv[])  
{  
    int port;  
    char *hostname;  
  
    ...  
  
    /* ouverture de la connexion */  
    hostname=argv[1];  
    port=atoi(argv[2]);  
    fd=connexion(hostname,port);  
  
    /* travail */  
    travail(fd);  
  
    close(fd);  
    exit(0);  
}
```

```
int connexion(char *hostname, int port)
{
    int fdPort;
    struct sockaddr_in addr_serveur;
    socklen_t lg_addr_serveur = sizeof addr_serveur;
    struct hostent *serveur;

    /* creation de la prise */
    fdPort=socket(AF_INET,SOCK_STREAM,0);
    if (fdPort<0) FATAL("socket");

    /* recherche de la machine serveur */
    serveur = gethostbyname(hostname);
    if (serveur == NULL) FATAL("gethostbyname");


    /* remplissage adresse socket du serveur */
    addr_serveur.sin_family      = AF_INET;
    addr_serveur.sin_port       = htons(port);
    addr_serveur.sin_addr       = *(struct in_addr *) serveur->h_addr;

    /* demande de connexion au serveur */
    if (connect(fdPort,(struct sockaddr *)&addr_serveur, lg_addr_serveur) < 0)
        FATAL("connect");
    return fdPort;
}
```

```
void travail(int fd)
{
    char h[3],m[3],s[3];

    /* recuperation reponse du serveur */
    if (read(fd,h,2) != 2) FATAL("read h");
    h[2]='\0';
    if (read(fd,m,2) != 2) FATAL("read m");
    m[2]='\0';
    if (read(fd,s,2) != 2) FATAL("read s");
    s[2]='\0';

    printf("Il est %s:%s:%s sur le serveur\n",h,m,s);
}
```



Source Socket  
Mode connecté  
Avec fichier de haut niveau

# Bigben–Serveur–Fichier de haut niveau

```
/* Taille maximale d'une ligne envoyee par serveur */
#define TAILLEMAXLIGNE 8
int main(int argc, char * argv[])
{
    int fdTravail, port;
    FILE *out;
    ...
    /* gestion des connexions de clients */
    while(1) {
        /* acceptation d'une connexion */
        fdTravail=accept(fd,NULL,NULL);
        if (fdTravail<=0)
            FATAL("accept");

        if (fork()==0) { /* fils : gestion du dialogue avec client */
            close(fd);
            /* Ouverture de fichiers de haut niveau (cf. polycop systeme) */
            out = fdopen(fdTravail,"w");
            /* travail */
            travail_fils(out);
            close(fdTravail);
            exit(0);
        }
        else { /* pere : repart a l'ecoute d'une autre connexion */
            close(fdTravail);
        }
    }
}
```

```
void ecrireligne(FILE *out, char ligne[])
{
    fprintf(out,"%s\n",ligne);
    fflush(out);
}
```

```
void travail_fils(FILE *out)
{
    long horloge;
    struct tm *temps;
    char tampon[TAILLEMAXLIGNE];
    int h,m,s;

    /* preparation de la reponse */
    time(&horloge);
    temps=localtime(&horloge);
    h = temps->tm_hour;
    m = temps->tm_min;
    s = temps->tm_sec;

    /* envoi de la reponse */
    sprintf(tampon, "%02d", h);
    ecrireligne(out,tampon);
    sprintf(tampon, "%02d", m);
    ecrireligne(out,tampon);
    sprintf(tampon, "%02d", s);
    ecrireligne(out,tampon);
}
```

# Bigben–Client–Fichier de haut niveau

```
/* Taille maximale d'une ligne recue du serveur */
#define TAILLEMAXLIGNE 8

int main(int argc, char * argv[])
{
    int port;
    char *hostname;
    FILE *in;

    ...

    /* ouverture de la connexion */
    hostname=argv[1];
    port=atoi(argv[2]);
    fd=connexion(hostname,port);

    /* Ouverture de fichiers de haut niveau (cf. polycop systeme) */
    in = fdopen(fd,"r");

    /* travail */
    travail(in);


    close(fd);
    exit(0);
}
```



```
char *lireligne(FILE *in, char ligne[])
{
    char *p;
    p = fgets(ligne, TAILLEMAXLIGNE, in);
    /* la lecture s'arrête après \n */
    return p;
}
```

*Affichage :*      13  
                          :15  
                          :25

```
void travail(FILE *in)
{
    char h[TAILLEMAXLIGNE], m[TAILLEMAXLIGNE], s[TAILLEMAXLIGNE];
    /* recuperation reponse du serveur */
    lireligne(in, h);
    lireligne(in, m);
    lireligne(in, s);
    printf("Il est %s:%s:%s sur le serveur\n", h, m, s);
}
```



Source Socket  
Mode connecté  
Scénario avec longs messages

## COTE SERVEUR

-----

```

void travail_fils(int fd)
{
    char tampon[4096];
    int rep, env;
    /* reception de la chaine */
    rep=read(fd, tampon, 4096);
    printf("recu    : %d\n",rep);
    /* envoi de la chaine recue */
    env=write(fd, tampon, rep);
    printf("envoye  : %d\n\n",env);
    close(fd);
}

```

## COTE CLIENT

-----

```

void travail(int fd)
{
    char tampon[4096];
    int rep, env, i;
    /* envoi de la chaine */
    for (i=0 ; i<4096 ; i++)
        tampon[i]='1';
    env=write(fd, tampon, 4096);
    printf("envoye  : %d\n",env);
    memset(tampon, 0, 4096);
    /* reception de la chaine */
    rep=read(fd, tampon, 4096);
    printf("recu    : %d\n",rep);
}

```

```

tuba~> serveur 3000
Le serveur ecoute le port 3000
recu    : 4096
envoye  : 4096

```

```

helicon~> client tuba 3000
envoye  : 4096

recu    : 4096

```

```
tuba~> serveur 3000
Le serveur ecoute le port 3000
recu      : 2896
envoye    : 2896
```

```
helicon~> client tuba 3000
envoye    : 4096

recu      : 2896
```

```
tuba~> serveur 3000
Le serveur ecoute le port 3000
recu      : 2896
envoye    : 2896
```

```
helicon~> client tuba 3000
envoye    : 4096

recu      : 1448
```