

COURS 2 : INTERFACE DES SOCKETS

1. Généralités
2. Interface des sockets : Mode connecté
3. Source Socket : Mode connecté
4. Source Socket : Mode connecté avec fichier de haut niveau

A quoi servent les sockets ?

- Applications client/serveur
 - Transfert de fichiers, Connexion à distance, Courrier électronique, Groupe de discussions, Web, Jeux en réseau, etc.
- Applications réparties
 - Java RMI, CORBA, .NET Remoting

Comment implémenter de telles applications : **les sockets**
(« prises »)

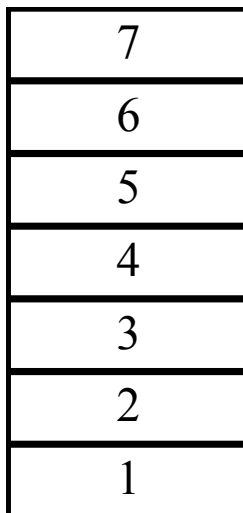
Le paradigme de communication Client/Serveur

- Interaction client/serveur
 - **Requête** envoyée par le client suivie d'une **réponse** envoyée par le serveur
 - Demande d'exécution d'un traitement à distance et réception de la réponse
- « Appel procédural » avec appelant et appelé non situés sur la même machine

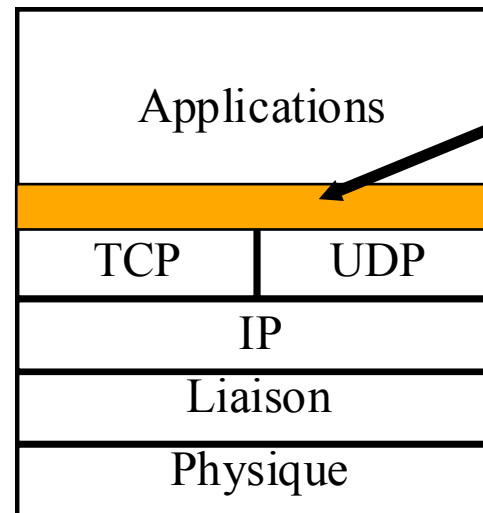
Les sockets

- Mécanisme de communication bidirectionnel inter-processus
- Présent dans tous les systèmes exploitation
 - 1983: L'université de Berkeley lance le BSD 4.2 incluant d'origine le protocole de communication TCP/IP et l'interface des sockets.
- Interface applicative
 - Interface entre les couches **application** et **transport**
 - Ensemble de primitives : **socket**, **connect**, **write**, **read**, **close**, **bind**, **listen**, **accept**...
 - Adaptée aux **protocoles TCP et UDP**...

Découpage en couches



Modèle OSI



Interface des
sockets

Modèle TCP/IP

Attributs des sockets

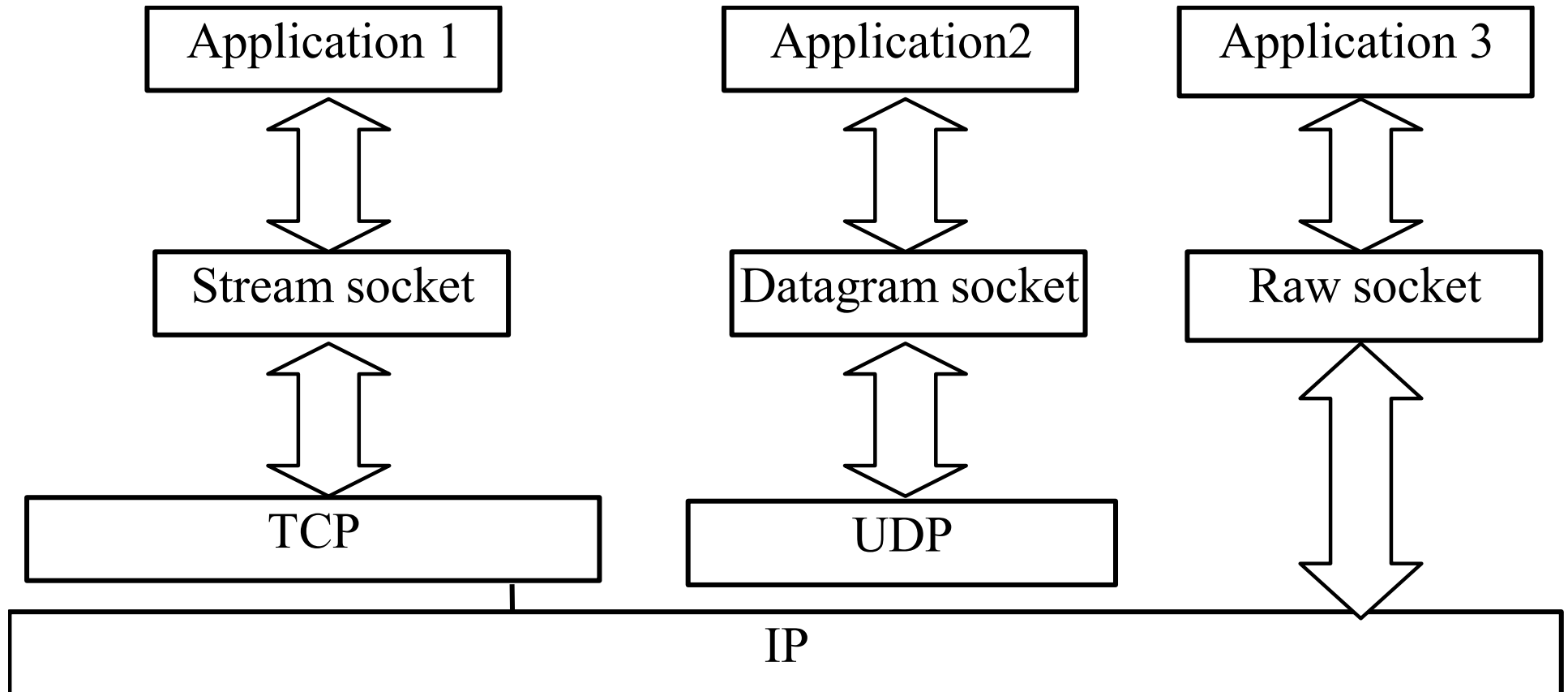
- un **nom**
 - Descripteur de fichier
- un **type**
 - **SOCK_STREAM** : mode connecté, remise fiable (TCP/IP)
 - **SOCK_DGRAM** : mode datagramme, non connecté + remise non fiable (UDP/IP)
 - **RAW** : mode caractère (pour accès direct aux couches inférieures comme IP)
- associé à un processus
- une **adresse** (adresse IP + n° port)

INTERFACE DES SOCKETS

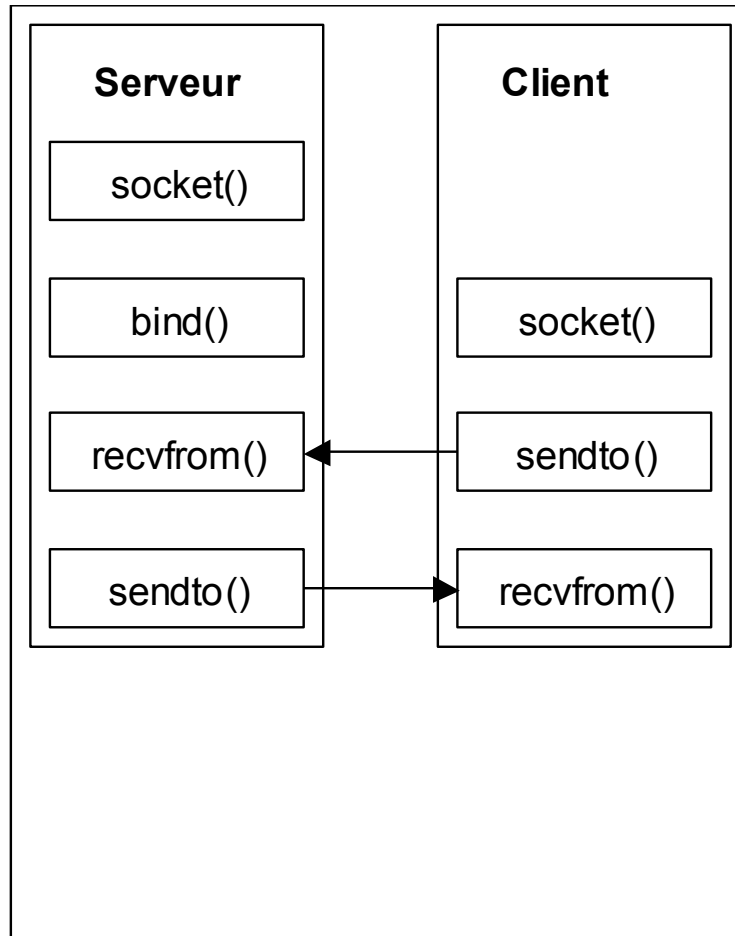
1. Généralités

2. Interface des sockets : Mode connecté
3. Source Socket : Mode connecté
4. Source Socket : Mode connecté avec fichier de haut niveau

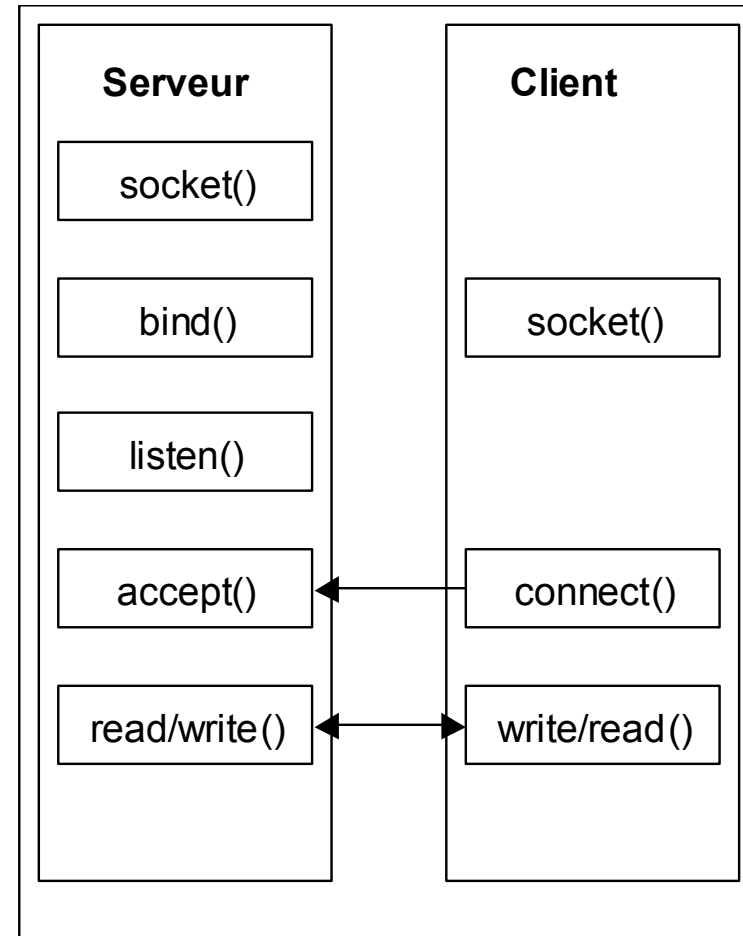
Types de socket et protocoles



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 d'une 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>)
 - Retourne :
 - un descripteur de socket
 - -1 si erreur
- **close(int socket)**
 - Ferme la connexion et supprime la structure de données «socket» associée
- **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)
 - Si n° de `port`=0
 - choix d'un numéro de port non utilisé
 - Si `adresse IP` = `INADDR_ANY`:
 - utilisation de l'adresse IP de la machine
 - `longueur-adresse` : `sizeof(struct sockaddr)`

Diverses primitives utiles 1/1

- `struct hostent` **gethostbyname**(char *name)
 - pour traduire un nom de machine en adresse IP

```
struct hostent *h;  
h=gethostbyname("stargate.ist.ga");  
printf("adresse IP: %s\n",  
       inet_ntoa(*((struct in_addr *)h->h_addr)));
```
- **getsockname**(int desc, struct sock_addr * p_adr, int * p_longueur)
 - pour récupérer l'adresse effective d'une socket (après bind)

Diverses primitives utiles 2/2

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

INTERFACE DES SOCKETS

1. Généralités
2. Interface des sockets : Mode connecté
3. Source Socket : Mode connecté
4. Source Socket : Mode connecté avec fichier de haut niveau

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 TCP

- **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 TCP

- newsock = **accept** (socket, adresse, lgr-adresse)
 - côté serveur
 - prise en compte d'une demande de connexion entrante sur un 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 !

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()`

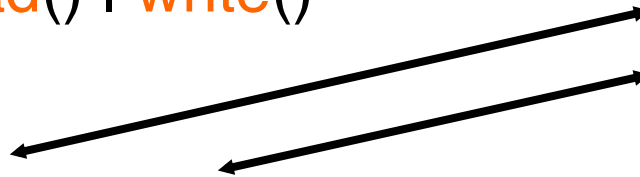
Tant que pas fini

envoyer une requête: `write()`

lire la réponse: `read()`

traiter la réponse

Fermer la socket : `close()`



INTERFACE DES SOCKETS

1. Généralités
2. Interface des sockets : Mode connecté
3. Source Socket : Mode connecté
4. Source Socket : Mode connecté avec fichier de haut niveau

Bigben-Serveur (1/3)

```
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 { /* père : repart a l'ecoute d'une autre connexion */
            close(fdTravail);
        }
    }
}
```

Bigben-Serveur (2/3)

```
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;
}
```


Bigben-Serveur (3/3)

```
void travail_files(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 (1/3)

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);
```

```
}
```

Bigben-Client (2/3)

```
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;
}
```

Bigben-Client (3/3)

```
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);
}
```

INTERFACE DES SOCKETS

1. Généralités
2. Interface des sockets : Mode connecté
3. Source Socket : Mode connecté
4. Source Socket : Mode connecté avec fichier de haut niveau

Bigben-Serveur - Fichier de haut niveau (1/2)

```
/* 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);
        }
    }
}
```

Bigben-Serveur-Fichier de haut niveau (2/2)

```
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 (1/2)

```
/* 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);
}
```


Bigben-Client - Fichier de haut niveau (2/2)

```
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);  
}
```