

Programmation TCP/IP

BOUZIDI

Année universitaire 2014-2015

D.BOUZIDI

1

Objectif

- Comprendre l'architecture client/serveur
- Maîtriser les notions essentielles de la communication via le réseau
- Comprendre le principe des protocoles réseaux
- Apprendre la programmation à base de socket

D.BOUZIDI

2

Plan

- Traitement des adresses Internet
- Les sockets
- Le Mode connecté
 - Socket Client
 - Socket serveur
 - Serveur multi-clients
- Mode non connecté
 - Sockets Serveur
 - Sockets Clients

Traitement des adresses Internet

Les réseaux informatique

- **Identification** : Interconnexion d'ordinateurs
- **Avantage** : Créé une synergie dans laquelle le tout est potentiellement supérieur à la somme de ses parties.
 - Partage des ressources : fichiers, applications, matériel, etc.
 - Communication : Messagerie, vidéoconférence, agenda, etc.
 - Optimisation des efforts : administration centralisée, assistance à distance, réplication des services, etc.
- **Les caractéristiques** :
 - Topologie : Bus, anneau, étoile, maillée
 - L'étendu géographique : PAN, LAN, MAN, WAN
 - Technologie utilisée : Bluetooth, infrarouge, Ethernet, Token Ring, etc.
 - Transmission : Diffusion, Multi-point, point à point
 - Commutation : Circuit, Message, Paquet(CV, CoC, CsC)
 - Support physique : coaxial, paires torsadées, FO, ondes radios, Satellites
 - Organisation : C/S, P2P, etc.

D.BOUZIDI

5

Normes et protocoles

- **Objectif** : Créer un référentiel commun pour harmoniser l'activité d'un secteur
- **Processus de normalisation** : Draft, Spécification, Norme, Protocole
- **Modèles** : OSI, TCP/IP, etc.
- **Organismes** :
 - Organisation Internationale de Normalisation (ISO) : *élaborer et publier des normes internationales et prendre des dispositions pour leur mise en application*
 - Union Internationale des Télécommunications (UIT) : *maintenir et promouvoir l'utilisation des services de télécommunications*
 - Internet Engineering Task Force (IETF) : *faire évoluer les technologies Internet.*
 - Consortium W3C : réaliser le plein potentiel du Web

D.BOUZIDI

6

L'adressage IP et services

- Un HOTE : ordinateur relié à un réseau
- Adresse IP :
 - Permet d'identifier un HOTE d'une manière unique
 - Séquence de nombres mise sur quatre /seize octets (IPv4/IPv6)
- Le DNS (Domaine Name Server) : associe une dénomination littérale à chaque adresse IP
- URL (Uniform Resource Locator) : Repère uniforme de ressource
 - Chaîne de caractère utilisée pour identifier les ressources dans le Web (document HTML, images, etc.)
 - défini par :
 - Le protocole à utiliser pour accéder à la ressource
 - Le nom du HOTE où la ressource se trouve
 - Le chemin d'accès à la ressource

D.BOUZIDI

7

Le type sockaddr

- Structure générique définie dans la bibliothèque **<sys/socket.h>**
- Décrit une adresse indépendamment du type de réseau (AppelTalk, DECnet, X.25, Novell, TCP/IP, Unix, etc.)
- Mise sur 16 octets
- Les champs :
 - int sa_family : représente la famille de l'adresse (2 octets)
 - char sa_data[14] : contient la valeur de l'adresse (14 octets)

sa_family	sa_data
-----------	---------

- Les adresses dépendent du type de réseaux, plusieurs types de structures sont proposées :

Type de réseau	Structure équivalente
Internet IPv4	sockaddr_in (sin_addr 4ø, sin_port 2ø)
Internet IPv6	sockaddr_in6 (sin6_addr 16ø, sin6_port 2ø)
IPX/SPX (Novell)	sockaddr_ipx
X25	sockaddr_x25

D.BOUZIDI

8

Le type sockaddr_in

- Représentation simplifiée des adresses Internet
- Séparation entre l'adresse du hôte et le port
- Définie dans le bibliothèque <netinet/in.h>
- Les champs de **sockaddr_in** sont :

Champ	Taille	Description
int sin_family	2	Famille d'adresse: indique la façon dont sont utilisés les 14 octets
short int sin_port	2	Port d'écoute
struct in_addr sin_addr	4	Adresse IP La structure in_addr est représentée par le champ s_addr qui est un entier de type in_addr_t
unsigned char sin_zero[8]	8	Bouffage pour que la structure sockaddr_in ait la même taille que la structure sockaddr

D.BOUZIDI

9

Initialisation d'une adresse

- Exemple :

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    struct sockaddr_in monAdresse;

    monAdresse.sin_family = AF_INET;
    monAdresse.sin_port = htons(80);
    inet_aton("127.0.0.1", &(monAdresse.sin_addr));
    // monAdresse.sin_addr.s_addr = inet_addr("127.0.0.1");
    memset(&(monAdresse.sin_zero), '0', 8);

    return 0;
}
```

D.BOUZIDI

10

Conversion de données (1/2)

- Les entiers sont stockés en mémoire de deux manières différentes :
 - big Endian : octet de poids **fort** à l'adresse la plus petite (Motorola, SPARC, etc.)
 - little Endian : octet de poids **faible** à l'adresse la plus petite (Intel)
- Les octets du numéro de port sont stockés dans l'ordre réseau (big-endian)
- Pour les machines de type little Endian une conversion du port vers le format **Network By Order** est nécessaire :

Fonction	Description
int htons (int val)	Host to Network Short
int htonl (int val)	Host to Network Long
int ntohs (int val)	Network to Host Short
int ntohl (int val)	Network to Host Long

```
monAdresse.sin_port=htons(numPort);
```

D.BOUZIDI

11

Conversion de données (2/2)

- Les octets de l'adresse IP sont stockés dans l'ordre réseau (big-endian)
- Pour mettre le champ sin_addr sous le format **Network By Order** et inversement :

Fonction	Description
in_addr_t inet_addr (const char *cp);	Fonction obsolète transformant l'adresse IP donnée au format "a.b.c.d" (cp) en un entier long (cas de 255.255.255.255 non supporté).
int inet_aton (const char *cp, struct in_addr in);	Transforme l'adresse IP donnée au format "a.b.c.d" (cp) en une structure in_addr (in).
char * inet_ntoa (struct in_addr in);	Transforme une adresse IP du format struct in_addr au format "a.b.c.d" (valeur de retour).

```
inet_aton("127.0.0.1", &monAdresse.sin_addr);
```

D.BOUZIDI

12

Quelques fonctions utiles

Fonction	Description
<code>void *memset (void *zone, int c, size_t n);</code>	Remplit une zone mémoire avec un caractère désiré. zone : Pointeur sur la zone mémoire à remplir c: Caractère de remplissage (code ASCII mis entre apos) t : Nombre de caractère à mettre dans la zone. Biblio : string.h
<code>bzero(&zone,sizeof(zone))</code>	Permet de mettre à zéro une zone , (fonction dépréciée) Biblio : string.h

```
void myFctZero(void *var, int var_size){
    char *p = (char*)var;
    int i;
    for(i = 0; i < var_size; i++)
        p[i] = 0;
}
```

D.BOUZIDI

13

La fonction gethostbyname

- Permet d'identifier un hôte à travers son nom en utilisant un DNS
- Retourne le résultat sous forme d'une structure hostent :

Champ	Description
<code>char *h_name</code>	Nom de l'hôte
<code>char **h_aliases</code>	Liste d'alias (surnom)
<code>int h_addrtype</code>	Type d'adresse de l'hôte (AF_INET pour IPv4, AF_INET6 pour IPv6)
<code>int h_length</code>	Longueur de l'adresse (sizeof(struct in_addr) ou sizeof(struct in6_addr))
<code>char **h_addr_list</code>	Liste d'adresses

- La fonction utilisée est
`struct hostent *gethostbyname(const char *nom)`
- Le fichier `/etc/hosts` peut être utilisé pour une résolution de nom local

D.BOUZIDI

14

Obtention de l'adresse IP

- Exemple :

```
host.c
1#include<string.h>
2#include<sys/socket.h>
3#include<netinet/in.h>
4#include<stdlib.h>
5#include<stdio.h>
6#include<netdb.h>
7int main(){
8    struct hostent *hote = NULL;
9    const char *nomHote = "www.ensias.ma";
10    struct in_addr ip;
11
12    hote = gethostbyname(nomHote);
13    if (hote == NULL){
14        fprintf(stderr, "Nom du hote %s inconnu.\n", nomHote);
15        exit(EXIT_FAILURE);
16    }
17    memcpy(&(ip.s_addr), hote->h_addr_list[0], sizeof(u_long));
18    printf("%s-----%s\n", nomHote, inet_ntoa(ip));
19    return 0;
20}
```

```
root@monOrdi:~/workspace/testServ
Fichier Édition Affichage Terminal Onglets Aide
[root@monOrdi testServ]# ./hostTest
www.ensias.ma-----196.200.134.233
[root@monOrdi testServ]#
```

D.BOUZIDI

15

Utilisation dans sockaddr_in

- Exemple :

```
useHostEnt.c
3#include<netinet/in.h>
4#include<stdlib.h>
5#include<stdio.h>
6#include<netdb.h>
7int main(){
8    struct hostent *hote = NULL;
9    const char *nomHote = "www.ensias.ma";
10    struct sockaddr_in aSrv;
11    aSrv.sin_family=AF_INET;
12    aSrv.sin_port=htons(80);
13    hote = gethostbyname(nomHote);
14    aSrv.sin_addr=((struct in_addr*)hote->h_addr_list[0]);
15    memset(&(aSrv.sin_zero), '0', 8);
16    int ds=socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
17    if(connect(ds, (struct sockaddr *)&aSrv, sizeof(struct sockaddr))!=-1)
18        printf("Erreur Impossible de se connecter au serveur !!!!!!!!!");
19    else printf("Votre client s'est connecte au serveur ");
20    close(ds);
21    return 0;
22}
```

D.BOUZIDI

16



Les sockets

D.BOUZIDI

17



Rôle de socket

- Un **socket** est un point de terminaison d'une communication bidirectionnelle,
- Les sockets se basent sur la philosophie d'UNIX : les E/S de toute nature (terminal, fichier ordinaire, connexion réseau) s'apparentent aux E/S sur fichiers
- Les données transitent sur le réseau selon des contraintes spécifiques, le cas D'Internet
 - Les paquets sont de taille limitée
 - Fractionnement et réassemblage,
 - Inclusion des en-têtes,
 - Analyse des en-têtes,
 - Accusé de réception des paquets, etc
- Les sockets exécutent automatiquement ces tâches :
 - Une connexion réseau peut être traitée comme un flux d'octets ordinaire.
 - Épargne au programmeur les détails de bas niveau du réseau : (Type des médias, Taille des paquets, Réexpédition, Adressage, etc.)

D.BOUZIDI

18

Principe de base de la communication

1. Chaque machine crée une socket,
2. Chaque socket sera associée à une adresse de la machine hôte,
3. Les deux sockets seront explicitement connectées si on utilise un protocole en mode connecté,
4. Chaque machine lit et/ou écrit dans sa socket,
5. Les données vont d'une socket à une autre à travers le réseau,
6. Une fois terminé chaque machine ferme sa socket.

D.BOUZIDI

19

Rôle des sockets

- **Socket** crée un point de communication, et renvoie un descripteur de socket.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

int main(){
    int ds;
    if((ds= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP))==-1)
        printf("Erreur lors de la création de la socket !!!!!!! ");
    else
        printf("le numéro de descripteur identifiant ce socket est %d",ds);
    return 0;
}
```

- Si la valeur du descripteur de socket est égale à -1, la socket ne s'est pas créée. Parmi les causes possibles :
 - Une des valeurs des paramètres est erronée
 - Manque de privilèges

D.BOUZIDI

20

Le domaine de socket

- Domaine ou **P**rotocol **F**amily (PF) spécifie la famille de protocole à utiliser avec la socket.

Type de domaine	Description
PF_INET	Pour les sockets IPv4
PF_INET6	Pour les sockets IPv6
PF_CCITT	Pour l'interface avec X25
PF_LOCAL	Pour rester en mode local (pipe).
PF_UNIX	Idem AF_LOCAL
PF_ROUTE	Accès à la table de routage

- On peut utiliser le préfixe AF (**A**ddress **F**amily) à la place de PF (l'équivalence est définie dans le fichier socket.h).

Le Type de socket

- Il spécifie le type de communication désiré.

Constructeurs	Description
SOCK_STREAM	Support de dialogue garantissant l'intégrité, fournissant un flux de données d'octets, et intégrant un mécanisme mode connecté, fiable et séquencée,
SOCK_DGRAM	Transmissions sans connexion, non garantie, composé de datagrammes de longueur fixe, généralement de petite taille.
SOCK_RAW	Dialogue direct avec la couche IP

Le protocole de socket

- Désigne le protocole à utiliser
- Il peut être mis à zéro, vu que l'association de la famille de protocole et du type de communication définit explicitement le protocole de transport
- Les protocoles de communication qui implémentent les sockets SOCK_STREAM garantissent qu'aucune donnée n'est perdue ou dupliquée.
 - **Exemple** : Si un bloc de données, pour lequel le correspondant a suffisamment de place dans son buffer, n'est pas transmis correctement dans un délai raisonnable, la connexion est considérée comme inutilisable, et les appels-système renverront une valeur -1 en indiquant une erreur ETIMEDOUT dans la variable globale errno
- Le protocole peut être identifié par un numéro (ou une constante) (voir fichier /etc/protocols). C'est le numéro utilisé dans l'entête IP.

Combinaison	Type de protocole possible
PF_INET + SOCK_STREAM	IPPROTO_TCP
PF_INET + SOCK_DGRAM	IPPROTO_UDP
PF_INET + SOCK_RAW	IPPROTO_RAW, IPPROTO_ICMP

D.BOUZIDI

23

Les Sockets mode connecté

D.BOUZIDI

24

Le mode connecté

- Support de dialogue garantissant l'intégrité des données,
- Fournissant un flux de données binaires,
- Itégrant un mécanisme pour les transmissions de données hors-bande.
- Une socket de type SOCK_STREAM est un flux d'octets full-duplex, similaire aux tubes (pipes).
- Elle ne préserve pas les limites d'enregistrements.
- Une socket SOCK_STREAM doit être dans un état connecté avant que des données puisse y être lues ou écrites.
- La primitive **connect** permet à un client de se connecter à un serveur distant.
- Une fois connectée les données y sont transmises par les primitives **send/recv** (read/write est une autre alternative)

D.BOUZIDI

25

Socket Client

- La mise en œuvre d'un socket client :
 - Le socket est créé au moyen d'un constructeur socket()
 - Le nouveau socket tente de se connecter à un hôte distant
 - Une fois la connexion établie, le système local et le système distant mettent en place un canal de communication à partir du socket
 - Canal utilisé pour échanger des données
 - La nature des infos échangées dépend du protocole employé les commandes FTP#HTTP
 - Un des deux interlocuteurs peut initier la fermeture de la connexion.
 - N.B :
 - HTTP exige de clore celle-ci après chaque requête honorée,
 - FTP autorise le traitement de plusieurs requêtes au cours d'une même connexion

D.BOUZIDI

26

Ouverture et fermeture de connexion

- L'appel de la fonction connect retourne

- -1 en cas d'erreur
- 0 si la connexion a réussi.

```
int connect(int ds, (struct sockaddr *)&addr_serveur, int sizeof(struct sockaddr));
```

- La fonction close retourne :

- -1 si une erreur s'est produite
- 0 si l'appel a réussi

```
int close(int ds)
```

- shutdown ferme la connexion d'une socket full-duplex dans un sens ou dans les deux:

- -1 si une erreur s'est produite
- 0 si l'appel a réussi

```
int shutdown (int ds, int valPossible)
```

Valeur possible	Description	valeur
SHUT_RD	en lecture seulement	0
SHUT_WR	en écriture seulement	1
SHUT_RDWR	en lecture-écriture	2

27

Connexion à un serveur

- Exemple : scan des ports actifs d'un serveur distant

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    struct sockaddr_in addsrv;

    addsrv.sin_family = AF_INET;
    addsrv.sin_port = htons(80);
    inet_aton("127.0.0.1", &(addsrv.sin_addr));
    memset(&(addsrv.sin_zero), '0', 8);

    int ds=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    int result=connect(ds, (struct sockaddr *)& addsrv, sizeof(struct sockaddr));
    if(result!=-1) printf("le serveur est actif\n" );
    else printf("le serveur est inactif\n" );
    close(ds);
    return 0;
}
```

D.BOUZIDI

28

Envoi des données

- **send** : envoie un ensemble d'octets dans une socket, retourne
 - -1 en cas d'erreur de transmission
 - Nombre d'octets envoyés

```
int send(int ds, const void*Msg, unsigned int lg_message, int flags)
```

```
char * msg=calloc(1024, sizeof(char));
strcpy(msg, "GET /\n")
send(int ds, msg, strlen(msg), 0);
```

- Le Flag avec une valeur à 0 indique un envoi normal

Valeur possible	
MSG_OOB	indiquera que les données urgentes doivent être envoyées
MSG_NOSIGNAL	demande de ne pas envoyer de signal SIGPIPE d'erreur sur les sockets connectées lorsque le correspondant coupe la connexion
MSG_DONTWAIT	active le mode non-bloquant. Une opération qui devrait bloquer renverra EAGAIN à la place
MSG_DONTROUTE	on utilise les interfaces sans se soucier des routes

29

Réception des données

```
int recv(int ds, char * msg, unsigned int lg_msg, int flag)
```

- La fonction **recv** retourne :
 - -1 en cas d'erreur de réception
 - 0 fin de la lecture
 - le nombre d'octets reçus

```
char * msg=calloc(1024, sizeof(char));
recv(int dsc, msg, 1024, 0);
printf("le message reçu est : %s ",msg);
```

- Flags :

Valeur possible	Description
MSG_OOB	on ne lit que les données prioritaires
MSG_PEEK	Lecture sans retirer de la file d'attente de réception. Une nouvelle lecture renverra la même chose
MSG_WAITALL	La lecture est bloquante jusqu'à ce que exactement lg_msg soient lus ou si une erreur survient
MSG_DONTWAIT	La lecture est réalisée à la réception de toutes données

30

Communication avec un serveur HTTP

▪ Exemple

```
#include<stdlib.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
int main(int argc, char **argv) {
    char * reqt=calloc(1024, sizeof(char));
    char * reps=calloc(1024, sizeof(char));
    struct sockaddr_in a;
    a.sin_family=AF_INET;
    a.sin_port=htons(80);
    inet_aton("127.0.0.1",&(a.sin_addr));
    memset(&(a.sin_zero), '0',8);
    int ds=socket (PF_INET, SOCK_STREAM, 6);
    connect (ds, (struct sockaddr *)&a,sizeof(struct sockaddr_in));
    printf("connecté au serveur !!!!!!!!!!!!!\n");
    strcpy(reqt, "GET /\n");
    send(ds,reqt, strlen(reqt), 0);
    printf("envoi au serveur : %d!!!!!!!!!!!!\n",strlen(reqt));
    printf("requete est : %s\n",reqt);
    while(recv(ds, reps, 1024, 0)>0)
        printf("%s", reps);
    close(ds);
    return 0;
}
```

31

Socket client vs Serveur

- Une communication cliente est assurée par quatre opérations élémentaires :
 1. Se connecter à une machine distance (connect)
 2. Envoyer des données (send)
 3. Recevoir des données (recv)
 4. Clore une connexion (close)
- Une communication serveur est assurée par six opérations élémentaires :
 1. S'attacher à un port (bind)
 2. Attendre les demandes de connexions émanant des machines distance (listen)
 3. Accepter les demandes de connexions sur le port local (accept)
 4. Envoyer des données (send)
 5. Recevoir des données (recv)
 6. Clore une connexion (close)

D.BOUZIDI

32

Lier le socket à une adresse

- La fonction **bind** permet d'associer un socket à une adresse réseau
- Le client demandant une connexion doit spécifier cette adresse
- le serveur spécifie sa propre adresse
- Celle du client est initialisée automatiquement par le système lors de sa connexion

```
int bind(int dss, (struct sockaddr*)&adrSrv, sizeof(struct sockaddr))
```

- L'appel retourne
 - -1 en cas d'erreur
 - 0 s'il est réussi

D.BOUZIDI

33

Ecouter de nouvelles connexions

- La fonction **listen** permet de mettre le serveur à l'écoute de nouvelles demandes de connexion
- Le client demandant la connexion est mis dans une file d'attente si le serveur n'est pas disponible
- Une fois le serveur est disponible il cherche une demande d'un client mis dans la file d'attente pour l'accepter
- lgFileAttente : indique la taille maximale de la file d'attente maintenue par la fonction listen

```
int listen(int dss, int lgFileAttente)
```

- L'appel retourne
 - -1 en cas d'erreur
 - 0 s'il est réussi

D.BOUZIDI

34

Accepter une nouvelle connexion

- La fonction **accept** permet d'accepter une demande de connexion d'un client
- Le serveur communique avec les clients via ces nouveaux sockets.
- Le serveur obtient un nouveau socket pour un client se connectant via l'appel `accept()`
- C'est ce nouveau socket qui sera utilisé pour l'envoi/reception des données
- **accept** vérifie si la file d'attente maintenue par `listen` contient au moins une connexion en attente :
 - Si la file est vide, `accept()` bloque jusqu'au moment où une connexion arrive
 - Si la file contient une connexion en attente, la variable `addrClt` sera initialisée automatiquement par l'adresse du client

```
int dsc accept(int dss, (struct sockaddr*)&addrClt, (socklen_t*)&tailleAddrClt)
```

- L'appel retourne
 - -1 en cas d'erreur
 - >0, s'il est réussi

D.BOUZIDI

35

Exemple

Serveur Echo

```
cltEcho.c  srvEcho.c
#include <sys/types.h>
int main() {
    int dss, dsc, lgAdrClt, i;
    struct sockaddr_in adrSrv, adrClt;
    char * reqt = calloc(1024, sizeof(char));
    char * reps = calloc(1024, sizeof(char));
    adrSrv.sin_family = AF_INET;
    adrSrv.sin_port = htons(99);
    inet_aton("127.0.0.1", &(adrSrv.sin_addr));
    memset(&adrSrv.sin_zero, '0', 8);
    dss = socket(PF_INET, SOCK_STREAM, 6);
    bind(dss, (struct sockaddr*)&adrSrv, sizeof(struct sockaddr));
    listen(dss, 10);
    while(1){
        dsc = accept(dss, (struct sockaddr *)&adrClt, (socklen_t *)&lgAdrClt);
        strcpy(reps, "Donnez votre message : "); send(dsc, reps, 1024, 0);
        recv(dsc, reqt, 1024, 0);
        send(dsc, reqt, 1024, 0);
        close(dsc);
    }
    return 0;
}
```

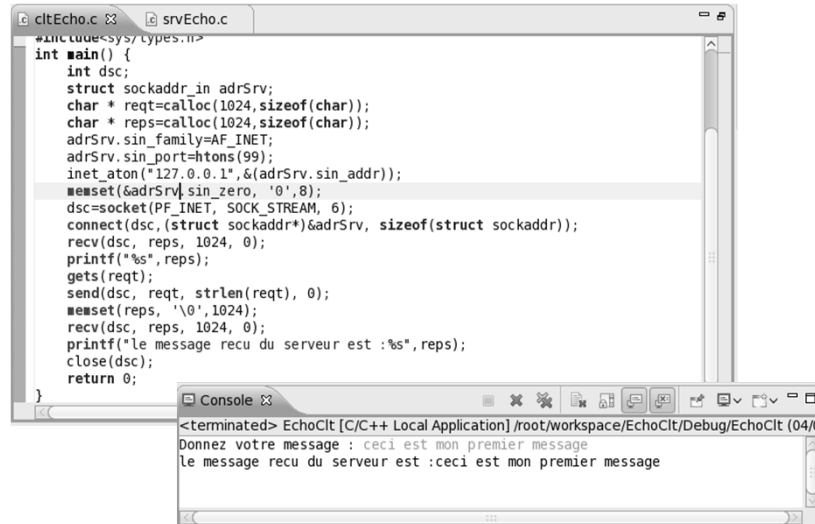
```
root@monOrdi:~# telnet 127.0.0.1 99
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Donnez votre message :
ceci est mon premier message !!!!
ceci est mon premier message !!!!
Connection closed by foreign host.
[root@monOrdi ~]#
```

D.BOUZIDI

36

Client Echo

- Exemple



```
cltEcho.c | srvEcho.c
#include<sys/types.h>
int main() {
    int dsc;
    struct sockaddr_in adrSrv;
    char * reqt=calloc(1024,sizeof(char));
    char * reps=calloc(1024,sizeof(char));
    adrSrv.sin_family=AF_INET;
    adrSrv.sin_port=htons(99);
    inet_aton("127.0.0.1",&(adrSrv.sin_addr));
    memset(&adrSrv.sin_zero, '0',8);
    dsc=socket(PF_INET, SOCK_STREAM, 6);
    connect(dsc,(struct sockaddr*)&adrSrv, sizeof(struct sockaddr));
    rcv(dsc, reps, 1024, 0);
    printf("%s",reps);
    gets(reqt);
    send(dsc, reqt, strlen(reqt), 0);
    memset(reps, '\\0',1024);
    rcv(dsc, reps, 1024, 0);
    printf("le message recu du serveur est :%s",reps);
    close(dsc);
    return 0;
}
```

Console

```
<terminated> EchoClit [C/C++ Local Application] /root/workspace/EchoClit/Debug/EchoClit (04/0
Donnez votre message : ceci est mon premier message
le message recu du serveur est :ceci est mon premier message
```

D.BOUZIDI

37

Le serveur parallèle

D.BOUZIDI

Serveur multi-clients

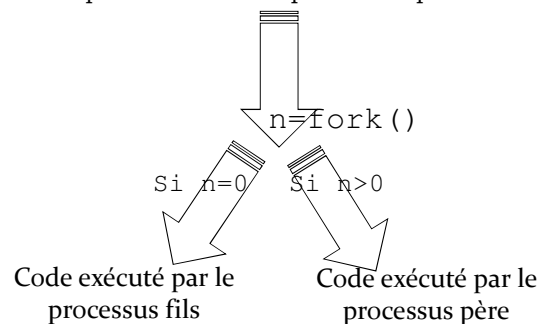
- Un serveur itératif : accepte une connexion, réalise le traitement, puis libère la connexion, pour servir la demande d'un autre client en attente
- Ce traitement lent (ex. transfert de fichier) pénalise les nouvelles connexion en attentes

☹ **Dégradation des performances**

- Un serveur multi-clients permet de contourner ce problème :
 - ☺ **Solution 1 : la fonction « fork » permettant de lancer simultanément des processus fils**
 - ☺ **Solution 2 : la fonction « select » offrant l'écoute et la lecture simultanée des données sur plusieurs descripteurs**

Fonction système fork

- Son appel duplique le contexte du processus père: on dit qu'on clone le processus appelant.
- Chaque processus à son propre bloc de données
- La fonction fork() retourne
 - La valeur 0 aux processus fils,
 - Le PID du processus fils au processus père.



Des contextes différents

```

1#include<stdio.h>
2#include<stdlib.h>
3#include<unistd.h>
4int main() {
5    int i=5;
6    if(fork()!=0){
7        i+=3;
8        printf("je suis le pere et mon pid est : %d\n",getpid());
9        printf("Après l'ajout de 3 à la valeur de i on obtient : %d\n",i);
10    }else {
11        i+=4;
12        printf(" je suis le fils et mon pid est : %d \n", getpid());
13        printf("Le pid de mon pere est : %d\n",getppid());
14        printf("Après l'ajout de 4 à la valeur de i on obtient : %d\n",i);
15        //exit(-1);
16    }
17    printf("je suis le processus ayant le pid : %d "
18          "et la valeur de i est %d\n",getpid(), i);
19    return 0;
20}

```

je suis le fils et mon pid est : 5449
 Le pid de mon pere est : 5448
 Après l'ajout de 4 à la valeur de i on obtient : 9
 je suis le processus ayant le pid : 5449 et la valeur de i est 9

je suis le pere et mon pid est : 5448
 Après l'ajout de 3 à la valeur de i on obtient : 8
 je suis le processus ayant le pid : 5448 et la valeur de i est 8

Fonction select

- Au lieu d'utiliser plusieurs processus fils pour un serveur multi-Clients
- La fonction **select** permet de surveiller au même temps plusieurs sockets et d'être averti de chaque changement réalisé.

```

#include <sys/select.h>
#include <sys/time.h>
#include <unistd.h>
int select(int n,
           fd_set *readfds,
           fd_set *writefds,
           fd_set *exceptfds,
           struct timeval *timeout);

```

L'identificateur du socket maximum qui sera testé lors du select

Descripteurs à surveiller en lecture : si des caractères deviennent disponibles en lecture

Descripteurs à surveiller en écriture

Descripteurs à surveiller pour des cas exceptionnelles

Délai d'attente maximal avant le retour (si null le délai est infini)

- N.B : Utilisez la valeur NULL pour ne pas tenir compte d'un des arguments

Les macros associées à select

- Le type utilisé regroupant un ensemble de descripteur à surveiller est `fd_set` (<sys/types.h>)
- Quatre macros sont disponibles pour la manipulation des ensembles :

Macros	Description
FD_SET (int <i>ds</i> , fd_set * <i>mySet</i>);	Ajoute l'identificateur de socket <i>ds</i> à l'ensemble de descripteur <i>mySet</i>
FD_CLR (int <i>ds</i> , fd_set * <i>mySet</i>);	Supprime l'identificateur de socket <i>ds</i> de l'ensemble <i>mySet</i>
FD_ZERO (fd_set * <i>mySet</i>);	Remet à vide l'ensemble de descripteur <i>mySet</i>
FD_ISSET (int <i>ds</i> , fd_set * <i>mySet</i>);	Après le select, cette fonction retourne une valeur différente de 0 si le descripteur <i>ds</i> a été sélectionné dans <i>mySet</i> ou non suite à un changement

43

Fonction select

- Exemple

```
#include <sys/select.h>
#include <sys/time.h>
#include <unistd.h>
int main(){
    fd_set mySet;
    int ds1,ds2,maxDS;
    //initialisation de l'ensemble de socket
    FD_ZERO(&mySet);
    FD_SET(ds1, mySet);
    Select(maxDS, &mySet, NULL,NULL,NULL);
    FD_CLR(ds1, mySet);
    ...
}
```

44



Annexes

D.BOUZIDI



Traitement des chaînes de caractères

D.BOUZIDI

Traitement de chaine de caractères

fonction	Description
char * strcpy(char * dest, const char * src);	copie une chaîne dans une autre chaîne et renvoie un pointeur sur cette dernière. Elle copie tous les caractères, y compris le caractère nul '\0'
size_t strlen (const char * str);	permet de récupérer la taille de n'importe quelle chaîne de caractères sans compter le '\0' final
int strcmp(const char *s1, const char *s2);	renvoie une valeur entière qui sera négative, nulle ou positive si la première chaîne est respectivement plus petite, de même taille ou plus grande que la deuxième chaîne
char * strncpy(char * dest, const char * src, size_t n);	copie n caractères de la source dans la destination
int strncmp(const char * s1, const char * s2, size_t n);	permet de ne comparer que les n premiers caractères des deux chaînes
char * strchr (const char * str, int c);	renvoie un pointeur sur la première occurrence de ce caractère, sinon elle renvoie NULL . strchr retourne un pointeur sur le dernier caractère rencontré

Traitement de chaine de caractères

fonction	Description
char * strstr(const char * str1, const char * str2);	Cherche une chaîne dans une autre. elle renvoie un pointeur sur cette chaîne, sinon elle renvoie NULL
char * strcat(char * dest, const char * src);	concaténer le contenu src à la suite du dest et renvoie un pointeur sur cette dernière. La chaîne dest est modifiée.
char * strncat(char * dest, const char * src, size_t n);	Copie n caractères de la chaîne src à la suite de la chaîne dest
char toupper(char c)	fonction de ctype.h permettant la conversion en majuscule d'un caractère (voir aussi : tolower , isupper , islower)
atoi(char * str)	permet de convertir une chaîne de caractères en un entier (voir aussi atol et atof)
sprintf(char * str, "%d",int entier)	Permet de convertir un entier (long"%l", float"%f") vers une chaîne de caractères

Traitement des chaines de caractères

- Exemple : Mettre une chaine de caractères en majuscule

```
1#include<stdio.h>
2#include<string.h>
3#include<ctype.h>
4#include<stdlib.h>
5int main(int argc, char **argv) {
6    int i;
7    char * chaine=calloc(1024,sizeof(char));
8    gets(chaine);
9    for(i=0;i<strlen(chaine);i++)
10        chaine[i]=toupper(chaine[i]);
11    printf("la chaine saisie est %s \n",chaine);
12    return 0;
13
14}
```

D.BOUZIDI

49

Le traitement de la date

D.BOUZIDI

Traitement du temps dans C

Type	Description
time_t	utilisé pour manipuler l'heure UNIX, il est vraisemblablement un entier (sur 32/64 bits) dépendamment de la machine (32/64 bits)
struct tm	structure contenant un instant mais utilisant les différentes unités et informations que l'on emploie : tm_year : nombre d'année écoulée depuis 1900 tm_mon : le numéro du mois (0-11) tm_mday : le numéro du jour du mois (1-31) tm_hour : nombre d'heure écoulée (0-23) tm_min : nombre minute écoulée (0-59) tm_sec : nombre de seconde écoulée (0-59) tm_yday : nombre de jour écoulé depuis 1 janv (0-365) tm_wday : nombre de jour écoulé de la semaine depuis le dimanche (0-6)
Fonction	Description
time_t time (time_t * ladata)	Permet de récupérer la date système sous la forme d'un entier (timestamp) indiquant le nombre de secondes écoulées depuis le 1er janvier.
struct tm * localtime (const time_t * ladata)	Permet de convertir l'entier « time_t » sous la forme d'une structure de type tm.
char* asctime (const struct tm* ladata)	Transforme une structure de type tm en chaîne de caractères.

D.BOUZIDI

51

Traitement du temps en C

- Affichage de la date système

```

gTemps.c
#include<stdio.h>
#include <time.h>

int main() {
    time_t laDate;
    time(&laDate);

    struct tm laDateFmt=*localtime(&laDate);

    printf("la date du système est : %s\n",asctime(&laDateFmt));
    //accès au différents champs de la date
    printf("On est le %d",laDateFmt.tm_mday);
    printf(", le mois %d", (laDateFmt.tm_mon+1));
    printf(", l'année %d", (laDateFmt.tm_year+1900));
    return 0;
}

```

- Sous fedora
 - date : commande permettant d'afficher la date système
 - Pour la changer d'une manière graphique : system-config-date

D.BOUZIDI

52

Les entrées sorties

D.BOUZIDI

Terminologies

- Entrées-sorties
 - Entrée : le programme lit des données de l'extérieur
 - Sortie : le programme écrit des données vers l'extérieur
- Flux de données : Chaque entrée ou sortie est vue comme une source ou une destination
- Stream (Canal) : contrôle un flux de données en le découpant en unités d'information (de taille fixe ou variable)
- Codages des caractères
 - Ascii sur un octet
 - Unicode sur 2 octets

D.BOUZIDI

54

Les fichiers

▪ Fichiers texte :

- Séquence de caractère organisés en lignes
- Fin de ligne : \r\n (sous Win) \r (sous unix)
- L'accès s'effectue à travers un descripteur : pointeur sur une structure de type FILE
- Ouverture selon un mode : r, w, a, etc.

Mode	Description
r	Lecture du contenu sans modification. Le fichier doit avoir été créé au préalable
w	Ecriture dans le fichier sans lecture. Si le fichier n'existe pas, il sera créé
a	Ajout du texte en partant de la fin du fichier. Si le fichier n'existe pas, il sera créé.
r+	Lecture et écriture dans le fichier. Le fichier doit avoir été créé au préalable
w+	Lecture et écriture dans le fichier avec suppression du contenu au préalable
a+	Ajout en lecture / écriture à la fin.

▪ Méthodes de manipulation

Méthode	Description
FILE * fopen (nomFichier, mode)	Ouverture du fichier. Si le pointeur vaut NULL, c'est que l'ouverture du fichier n'a pas fonctionné
fgets (ligne, nbreOctetsLuALaFois, descF)	Lecture d'une ligne
fputs (ligne, descF)	Ecriture d'une ligne.
close (descF)	Fermeture du fichier

D.BOUZIDI

55

Les fichiers

▪ Lecture d'une chaîne via le clavier :

- Séquence de caractère organisés en lignes
- Fin de ligne : \r\n (sous Win) \r (sous unix)
- L'accès s'effectue à travers un descripteur : pointeur sur une structure de type FILE
- Ouverture selon un mode : r, w, a, etc.
- Méthodes de manipulation

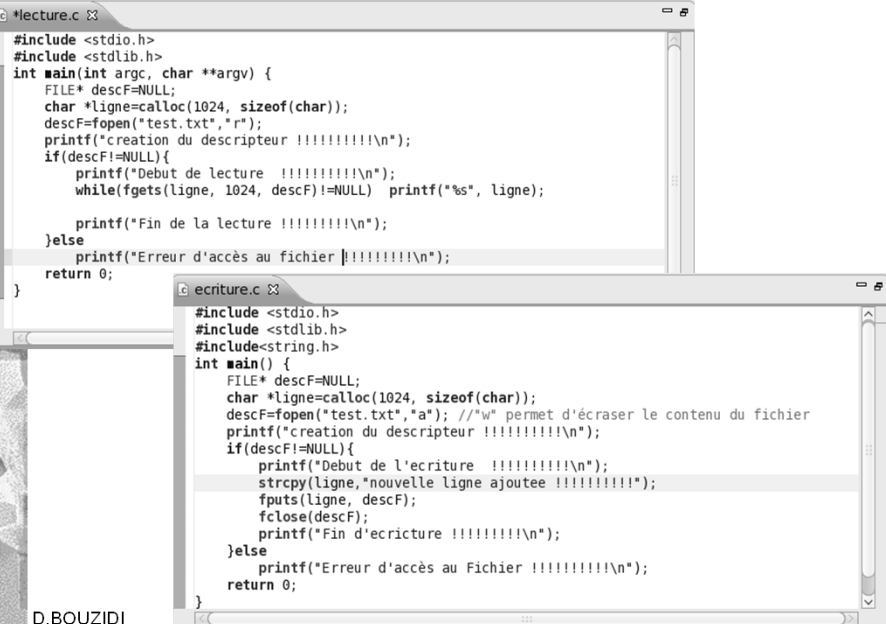
Mode	Description
scanf	Lecture du contenu sans modification. Le fichier doit avoir été créé au préalable
gets(chaine)	Déconseillée vu qu'on peut écraser un autre programme dans le cas de débordement de lecture N.B : la fonction copie tous les caractères saisis, mais sans le caractère de validation '\n'
getc	Ajout du texte en partant de la fin du fichier. Si le fichier n'existe pas, il sera créé.
fgets(chaine, taille, stdin)	Évite le problème de débordement de la fonction gets, vu que la taille de la zone réservée pour la lecture est fixée N.B : la fonction copie tous les caractères saisis, y compris le caractère de validation '\n'

D.BOUZIDI

56

Lecture/Ecriture d'un fichier

Exemple :



The image shows two overlapping code editors. The top editor, titled 'lecture.c', contains code for reading a file. The bottom editor, titled 'ecriture.c', contains code for writing to a file. Both programs use 'test.txt' as the filename and include standard headers for file operations and memory management.

```
#lecture.c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    FILE* descF=NULL;
    char *ligne=calloc(1024, sizeof(char));
    descF=fopen("test.txt","r");
    printf("creation du descripteur !!!!!!!!!\n");
    if(descF!=NULL){
        printf("Debut de lecture !!!!!!!!!\n");
        while(fgets(ligne, 1024, descF)!=NULL) printf("%s", ligne);

        printf("Fin de la lecture !!!!!!!!!\n");
    }else
        printf("Erreur d'accès au fichier !!!!!!!!!\n");
    return 0;
}
```

```
ecriture.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    FILE* descF=NULL;
    char *ligne=calloc(1024, sizeof(char));
    descF=fopen("test.txt","a"); // "w" permet d'écraser le contenu du fichier
    printf("creation du descripteur !!!!!!!!!\n");
    if(descF!=NULL){
        printf("Debut de l'écriture !!!!!!!!!\n");
        strcpy(ligne,"nouvelle ligne ajoutée !!!!!!!!!");
        fputs(ligne, descF);
        fclose(descF);
        printf("Fin d'écriture !!!!!!!!!\n");
    }else
        printf("Erreur d'accès au Fichier !!!!!!!!!\n");
    return 0;
}
```

D.BOUZIDI