Interfaces de mode message asynchrone - Socket Berkeley

RSX 102
Anne WEI
CNAM Paris

1

Plan

1 Introduction des interfaces en mode message asynchrone
2 Interfaces sockets Berkeley – TCP/UDP
3 Primitives de l'interface socket

2

Introduction (1)

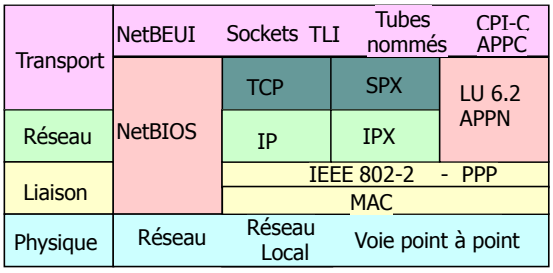
- Pour créer une connexion entre un client et un serveur, les *sockets réseau* permettent de gérer les multi-tâches et le flux à l'aide d'API (*Application Programming Interface*). Le rôle d'API est de contrôler et d'utiliser les sockets réseau.

- Interfaces de mode message asynchrone
- les interfaces dans les systèmes OSI sont les interfaces logicielles
- on peut distinguer deux types d'interfaces :
- Interfaces «transport» assure la communication au niveau Transport
- Interfaces «application» assure la communication par messages au niveau Application. Ce type des interfaces offre des extensions plus ou moins significatives par rapport au niveau transport

3

Introduction (2)

- Interfaces de « transport »
- la communication est de tout en bout
- les piles de protocoles de couches basses résumées au niveau Transport
- les interfaces logicielles pour l'accès à des piles de protocoles ; API (*Application Programming Interface*)
- Socket, TLI, NetBEUI, APPC/CPI-C, Tubes nommés



4

Introduction (3)

- Les API du niveau Transport
- Sockets (prises)
- L' interface de programmation (modèle de communication inter processus) pour la suite de la pile protocolaire de TCP/UDP/IP. La socket de Berkeley (BSD, *Berkeley Software Distribution)*, crée dans les années 80, est une interface du système UNIX.
- L'API sockets sous Windows est baptisée WinSock.

- TLI - *Transport Layer Interface*
- l'interface crée par AT&T standardisée comme XTI (*X/Open Transport Interface*) peut fonctionner avec la pile TCP/IP

- NetBEUI – *NetBios Extended Basic Input Output System*
- l'interface introduite pour le PC Network ; la pile protocolaire de NetBIOS (IBM et Microsoft). Le PC Network est un système de réseau local avec NetBIOS.
- Utilisé pour la pile protocolaire de SPX/IPX(*Sequenced Packet Exchange/Internetwork Packet Exchange)* de Novell

5

Introduction (4)

- Les API du niveau Transport (suite)
- les tubes nommés (Named pipes)
- L'interface entre processus IPC (*Inter Process Communication*) introduite sous Unix BSD pour étendre la notion de tube (pipe, exécution parallèle).
- Elle gère les échanges réseaux comme des accès fichiers (disponible sur TCP/IP et SPX/IPX)

- APPC et CPI-C
- Advanced Program to Program Communication et Common Programming Interface for Communication
- APPC est l'interface de programmation pour l'accès aux services SNA (*Systems Network Architecture*) d'IBM LU6.2 pour tous types de produits (60 primitives)
- CPI-C simplifie l'interface APPC (40 primitives)
- Evolution de SNA vers une architecture réseau incorporant tout type de matériels grands, moyens et petits systèmes sous le nom d'APPN (*Advanced Peer to Peer Network*) d'IBM

6

Introduction (5)

- Les API du niveau Application
- MOM (Message Oriented Middleware) – Interlogiciel orienté message
- Une interface **universelle** pour des échanges en mode message asynchrone (opposé au type requête/réponse) qui masque les différentes API du niveau Transport
- Notion de files d'attentes de messages (message queues) :
- Les files d'attente sont **persistantes** (sur disque) ou non persistantes
- Avec une file non persistante le comportement est celui du transport
- Avec une file persistante un site qui n'est pas opérationnel sera atteint lors de son réveil (fonctionnement asynchrone similaire au courrier électronique)

- Produits commerciaux
- AMQP (*Advanced Message Queuing Protocol)*
- MSMQ (*Microsoft Message Queueing)*
- JMS (*Java Message Service)*
- Autres produits : MQ Series IBM, TIBCO, SUN

7

Plan

1 Introduction des interfaces en mode message asynchrone
2 Interfaces sockets Berkeley – TCP/UDP
3 Primitives de l'interface socket

8

Interfaces Sockets Berkeley (1)

- Sockets Berkeley est l'interface de programmation de communication :
- créées dans les années 80 pour Unix BSD, elles contient un libraire de développement, langage en C. Depuis des années, Windows supportent également les sockets (sockets Windows)

- **Objectifs généraux** sont de fournir des moyens de communications entre processus (IPC) en toutes circonstances (échanges locaux ou réseaux) ; de masquer les détails d'implantation des couches de transport aux usagers ; de masquer les différences entre protocoles de transports hétérogènes sous une même interface (TCP, Novell XNS, OSI) et de fournir une interface d'accès qui se rapproche des *accès fichiers* pour simplifier la programmation.

9

Interfaces Sockets Berkeley (2)

- Une socket est un point d'accès de service au niveau Transport, les protocoles TCP/UDP (ou des autres protocoles) par exemple.
- Une socket contient
- *Un nom* : un identifiant unique sur chaque site
- *Un type* : quel protocole, quelle sémantique

- *Un ensemble de primitives* (un service pour l'accès aux fonctions de transport)
- Des données encapsulées

- Des exemples de fonctionnalités (primitives)
- socket () : création d'un nouveau point d'accès
- send () : envoyer des données à une socket
- recv () : recevoir des données venues d'une socket

10

Désignation des sockets

- Identifier une socket dans un réseau et au niveau Transport par un couple NSAP (*Network Service Access Point*) et TSAP
- Exemple du protocole TCP (*Transmission Control Protocol*): *N° de port* associé à *une adresse IP*
- N° de port est en 16 bit (unsigned integer) ; entre 0 et 65535

- IANA *(Internet Assigned Number Authority)* gère
- N° de ports réservés: des numéros de ports réservés pour des services généraux «bien connus» ou « well-known ports» (de 0 à 1023)
- exemples ports TCP: Telnet 23, DNS 53, HTTP 80
- exemples ports UDP: SQL service 118, Echo 7
- TCP et UDP utilisent certains ports communs (Telnet 23, et DNS 53 par exemple)
- N° de ports enregistrés (registered) : entre 1024 et 49151 pour des applications ayant fait une demande. La signalisation H.323 – port 1720, par exemple
- N° de ports privés (private/dynamiques): entre 49152 et 65535, sont attribués dynamiquement aux sockets utilisateurs (clients)

11

Désignation des sockets

- Identifier une socket dans un réseau et au niveau Transport: un couple NSAP (*Network Service Access Point*) et TSAP
- Exemple du protocole TCP: *N° de port* associé à *une adresse IP*

- IANA *(Internet Assigned Number Authority)* gère
- N° de ports réservés : des numéros de ports réservés pour des services généraux «bien connus» ou « well-known ports» (de 0 à 1023)
- exemples ports TCP : Telnet 23, DNS 53, HTTP 80
- exemples ports UDP : SQL service 118, Echo 7
- TCP et UDP utilisent certains ports communs (Telnet 23, et DNS 53 par exemple)
- N° de ports enregistrés (registered) : entre 1024 et 49151 pour des applications ayant fait une demande. La signalisation H.323 – port 1720, par exemple
- N° de ports privés (private/dynamiques) entre 49152 et 65535 sont attribués dynamiquement aux sockets utilisateurs (clients)

12

Conception des sockets avec TCP

- TCP est un transport fiable en connexion et en mode bidirectionnel point à point grâce à ses fonctionnalités (gestion de congestion, contrôle de flux…)
- une socket TCP peut être utilisée par plusieurs connexions TCP simultanément
- une connexion est identifiée par le couple d'adresses socket des deux extrémités.
- Un exemple : le client X ayant l'@ 146.86.3.15 et le port *56123*
  le serveur Y ayant l'@146.86.5.20 et le port du TCP *80* pour HTTP

- Un échange TCP est du type orienté flot d'octets
- les zones de données qui correspondent à des envois successifs ne sont pas connues à la réception
- Pour optimiser, TCP peut tamponner les données et les émettre ultérieurement

** @IP représente la machine, le port représente le processus d'application*

13

Conception des sockets avec TCP

- TCP est un transport fiable en connexion et en mode bidirectionnel point à point grâce à ses fonctionnalités (gestion de congestion, contrôle de flux…)
- une socket TCP peut être utilisée par plusieurs connexions TCP simultanément
- une connexion est identifiée par le couple d'adresses socket des deux extrémités.
- Un exemple: le client X ayant l'@ 146.86.3.15 et le port *12345*
  le serveur Y ayant l'@146.86.5.20 et le port du TCP *80*

La socket est identifiée par <146.86.3.15 : *12345*, 146.86.5.20 : *80*>

- Un échange TCP est du type orienté flot d'octets
- les zones de données qui correspondent à des envois successifs ne sont pas connues à la réception
- Pour optimiser, TCP peut tamponner les données et les émettre ultérieurement

14

Conception des sockets avec UDP

- UDP (*User Datagram Protocol*) est un transport non fiable sans connexion et en mode bidirectionnel point à point. C'est-à-dire, il n'y a pas d'accusé de réception, de re-transmission et de contrôle de flux.
- une connexion est identifiée par le couple d'adresses socket des deux extrémités à la même forme que celui d'une socket TCP (@IP et N° de port)
- Un échange UDP est sans connexion (échange de datagrammes)
- les zones de données qui correspondent à des envois successifs sont respectées à la réception (si l'option de « total de contrôle » est utilisée)

15

Plan

16

Rappel – généralités de l'Internet (1)

- Adressage uniforme de sous-réseaux
- une machine est identifiable par son adresse MAC (01-23-45-67-89-ab, par exemple) et son adresse IP
- l'Internet définit des Adresses Uniques Universelles :
  - IPv4, Notation Décimale Pointée sur 4 champs sous format : (A.B.C.D). (N°Réseau, N°station)
    un exemple: l'adresse IP est sur **32 bits- 4 octets**
    192.200.25.1
  - IPv6, Notation Décimale Pointée sur 16 champs sous format : (A:B:C:D:E:F:G:H). (N°Réseau, N°station)
    un exemple: l'adresse IP est sur **128 bits- 16 octets**
    2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - Les Adresses IP sont répertoriées sur un site dans le fichier /etc/hosts ou gérées dans la base NIS (Yellow Pages) des "hosts" équivalente ou par un serveur de noms BIND par nslookup().

17

Rappel – généralités de l'Internet (1)

- Adressage uniforme de sous-réseaux
- une machine est identifiable par son adresse MAC (01-23-45-67-89-ab, par exemple) et son adresse IP
- l'Internet définit des Adresses Uniques Universelles:
  - IPv4, Notation Décimale Pointée sur 4 champs sous format : (A.B.C.D). (N°Réseau, N°station)
    un exemple: l'adresse IP est sur **32 bits- 4 octets**
    192.200.25.1
  - IPv6, Notation Décimale Pointée sur 16 champs sous format : (A:B:C:D:E:F:G:H). (N°Réseau, N°station)
    un exemple: l'adresse IP est sur **128 bits- 16 octets**
    2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - Les adresses IP sont répertoriées sur un site dans le fichier /etc/hosts ou gérées dans la base NIS - *Network Information Service* (Yellow Pages) des "hosts" équivalente ou par un serveur de noms BIND par nslookup().

18

Rappel – généralités de l'Internet (2)

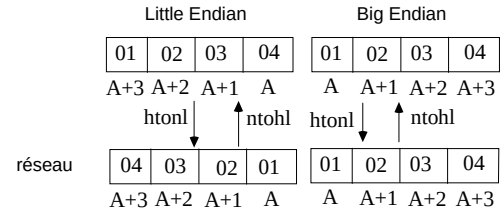
- noms de machines
- utilisateur azimov.cnam.fr
  administrateur 163.173.128.6/a3ad8006 en hexadécimal (fichiers de configuration)
  programmeur azimov.cnam.fr
  gethostbyname (char * name, int name_len)) fournit une adresse IP
  163.173.128.6
  a3ad8006 ou 0680ada3 dépendant machine
- DNS (*Domaine Name System*) sert à convertir les noms symboliques (azimov.cnam.fr, par exemple) en adresses IP

* les adresses MAC apparaissent à travers les commandes arp ou ifconfig/ipconfig

19

Rappel – généralités de l'Internet (3)

- L'ordre d'émission (de gauche à droite par exemple) est différent d'une machine à l'autre
- la représentation de données dans les machines Big Endian : on numérote les octets de gauche à droite. 0x 01020304 (4 octets en hexadécimal), l'octet le plus significatif est de 01
- la représentation de données dans les machines Little Endian : on numérote les octets de droite à gauche. 0x 01020304 , l'octet le plus significatif est de 04
- la conversion entre la représentation de données dans les machines et celle dans le réseau



*htonl = host to network long integer

Rappel – généralités de l'Internet (4)

- des primitives de conversion d'entiers du format «*host*» au format «*network*» :
  htonl() long - entier 32 bits , //"long" depend de la machine
  htons() court - entier 16 bits *(host to network short)*
- des primitives de conversion d'entier du format «*network*» au format «*host*»:
  ntohl()
  ntohs()

- le problème de conversion est résolu pour les données applicatives par ASN1-BER (*Abstract Syntax Notation One - Basic Encoding Rules*) avec la couche Présentation de l'ISO ou SNMP (*Simple Network Management Protocol*), ou par CDR (*Common Data Representation*) avec CORBA.

21

Primitive Socket Berkeley (1)

- Permet la création d'un nouveau point d'accès de service transport avec
  - Son nom et son type
  - Son allocation de l'espace des données

- Trois paramètres d'appel
  - Famille d'adresses réseaux utilisées locale, réseau IP, réseaux OSI…
  - Type de la socket (du service) sémantique de la communication
  - Protocole de transport utilisé

- Un paramètre résultat: le numéro de descripteur socket

- Profil d'appel des primitives (en-tête du fichier) en C
  - # include<sys/types.h>
  - #include <sys/socket.h>
  - # include <netinet.h>

22

Primitive Socket Berkeley (1)

- Permet la création d'un nouveau point d'accès de service transport avec
  - Son nom et son type
  - Son allocation de l'espace des données
- Trois paramètres d'appel
  - Famille d'adresses réseaux utilisées locale, réseau IP, réseaux OSI…
  - Type de la socket (du service) sémantique de la communication
  - Protocole de transport utilisé
- Un paramètre résultat : le numéro de descripteur socket
- Profil d'appel des primitives (en-tête du fichier) en C
  - # include<sys/types.h> (gestion de données, de processus…)
  - #include <sys/socket.h> (primitives)
  - # include <netinet.h> (adresse Internet)

23

Primitive Socket Berkeley (2)

- Paramètre Famille
  - AF_UNIX : communication locale (i – node)
  - AF_INET : Communication Internet (AF : Address Family)
  - AF_ISO: Communication ISO
  - PF_INET (PF : Protocol Family)
- Paramètre Type
  - SOCK_STREAM: flot d'octets en mode connecté (TCP)
  - SOCK_DGRAM: Datagramme en mode non connecté (UDP)
  - SOCK_RAW: Accès aux couches basses (IP)
  - SOCK_SEQPACKET: Format structuré ordonné (protocoles différents de l'Internet)
- Paramètre Protocole
  - IPPROTO_TCP : SOCK_STREAM
  - IPPROTO_UDP : SOCK_DRAM
  - IPPROTO_ICMP : SOCK_RAW ; InternetControlMessageProtocol
  - IPPROTO_RAM : SOCK_RAW

24

Primitive Socket Berkeley (2)

- Paramètre Famille
  - AF_UNIX : communication locale (i – node)
  - AF_INET : Communication Internet (AF: Address Family)
  - AF_ISO: Communication ISO
  - PF_INET (PF: Protocol Family)
- Paramètre Type
  - SOCK_STREAM : flot d'octets en mode connecté (TCP)
  - SOCK_DGRAM : Datagramme en mode non connecté (UDP)
  - SOCK_RAW: Accès aux couches basses (IP)
  - SOCK_SEQPACKET: Format structuré ordonné (protocoles différents de l'Internet)
- Paramètre Protocole
  - IPPROTO_TCP (SOCK_STREAM)
  - IPPROTO_UDP (SOCK_DRAM)
  - IPPROTO_ICMP (SOCK_RAW); InternetControlMessageProtocol
  - IPPROTO_RAM (SOCK_RAW)

25

Primitive Socket Berkeley (2)

- Paramètre Famille
  - AF_UNIX : communication locale (i – node)
  - AF_INET : Communication Internet (AF: Address Family)
  - AF_ISO: Communication ISO
  - PF_INET (PF: Protocol Family)
- Paramètre Type
  - SOCK_STREAM: flot d'octets en mode connecté (TCP)
  - SOCK_DGRAM: Datagramme en mode non connecté (UDP)
  - SOCK_RAW: Accès aux couches basses (IP)
  - SOCK_SEQPACKET: Format structuré ordonné (protocoles différents de l'Internet)
- Paramètre Protocole
  - IPPROTO_TCP : SOCK_STREAM
  - IPPROTO_UDP : SOCK_DGRAM
  - IPPROTO_ICMP : SOCK_RAW ; Internet Control Message Protocol
  - IPPROTO_RAM : SOCK_RAW

26

Primitive Socket Berkeley (3)

- un exemple d'une socket du protocole TCP/IP
- int socket (*int* family, *int* type, *int* protocol)
- int s;

s = socket (AF_INET, SOCK_STREAM, 0)

ou

s = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)

- *0:* on ne définit aucun protocole, dont le protocole correspond au paramètre « SOCK_STREAM ». C'est-à-dire, le protocole TCP

27

Primitive bind (1)

- l'attribution d'un port local et d'une adresse réseau. Cette attribution est nécessaire avant l'utilisation de socket

- Profil d'appel de la primitive

```
#include <sys/types.h>
#include <sys/socket.h>
int  bind (int socket, struct sockaddr_in * address, int address_length)
```

- Trois paramètres d'appel
- Numéro du descriptif de socket
- Structure d'adresse de socket pour internet type « sockaddr_in »
- Longueur de la structure d'adresse

28

Primitive bind (2)

- Structure d'adresse de socket

```
#include <sys/socket.h>
struct sockaddr_in {
short                   sin_family;
u-short                 sin_port;
struct    in_addr       sin_addr;
char                    sin_zero [8];};
```

- un exemple:

```
struct          sockaddr_in     sin;
int             s;
s = socket (AF_INET, SOCK_STREAM,0);
sin.sin_family = AF_INET;
sin.sin_port = htons (9999);  //un numéro local vers un numéro réseau
sin.sin_addr.s_addr = INADDR_ANY;
bind (s, (struct sockaddr  *) &sin, sizeof (sin)); // si bind = -1, l'erreur du n° de port
```

29

Primitive bind (2)

- un exemple:

```
struct          sockaddr_in     sin;
int             s;
s = socket (AF_INET, SOCK_STREAM,0);
sin.sin_family = AF_INET;
sin.sin_port = htons (9999);  //un numéro local vers un numéro réseau
sin.sin_addr.s_addr = INADDR_ANY;
```

- INADDR_ANY permet d'indiquer automatiquement l'adresse IP d'un site (192.1.2.2, par exemple); htons /htonl convertir dans le bon ordre, car les données ne sont pas représentées de la même façon suivant les processeurs

30

Primitive bind (3)

- Il est possible de laisser au système le choix d'un numéro de port libre au moment de l'appel à la fonction *bind*
  - le cas d'un client
  - dans ce cas, sin_port = 0
- il est possible d'associer à une socket l'ensemble des adresse IP de la machine (les adresses locales) dans le cas où elle est connectée à plusieurs réseaux
  - Le cas d'un serveur
  - Dans ce cas, sin_addr = INADDR_ANY

31

Primitive connect (1)

- La primitive ***connect*** permet à un client de demander l'ouverture (active) de connexion à un serveur
  - L'adresse du serveur doit être fournie
  - La partie extrémité local relative au client est renseignée automatiquement
  - Le client ne fournit plus l'adresse du serveur pour chaque appel mais le descriptif de la socket (qui contient l'adresse serveur)
- Profil d'appel de la primitive

```
#include <sys/types.h>
#include <sys/socket.h>
int  connect (int socket, struct sockaddr_in * addr_serv, int addr_lg_serv)
```

32

Primitive connect (2)

- Un exemple

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

struct          sockaddr_in     sin;
int     s, c;

s = socket (AF_INET, SOCK_STREAM,0);
sin.sin_family = AF_INET;
sin.sin_port = htons (9999);  //un numéro local vers un numéro réseau
sin.sin_addr.s_addr = inet_addr (servIP);  //format : 192.143.12.2, par exemple

c = connect (s, &sin,  sizeof(sin))
```

33

Primitive listen

- Primitive ***listen*** permet d'indiquer le nombre d'appels maximum attendu dans le mode connecté lorsque plusieurs clients sont susceptible d'établir plusieurs connexions avec un serveur
- Primitive *listen* est immédiate (non bloquante)
- Profil d'appel de la primitive

```
#include <sys/types.h>
#include <sys/socket.h>
int  listen (int socket, int max_connexion)
```

- un exemple:

Listen (s, 5); le nombre 5 est la taille de file d'attente du serveur

34

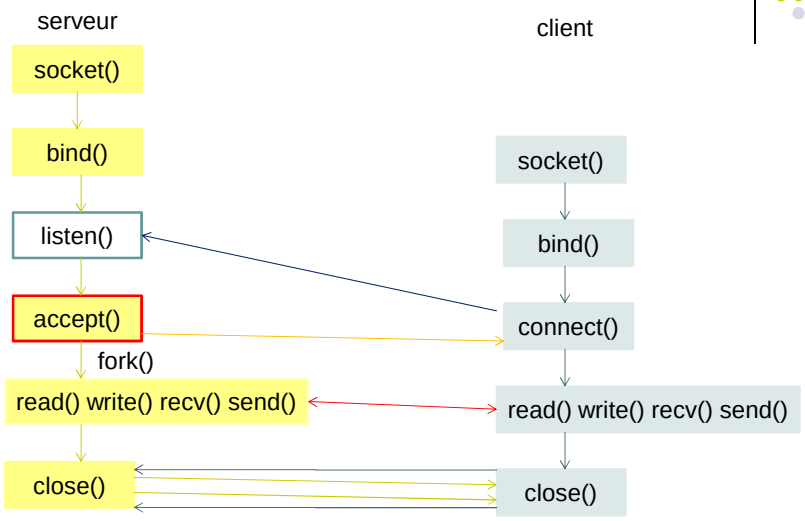
Primitive accept

- La primitive ***accept*** permet de se bloquer en attente d'une nouvelle demande de connexion en mode connecté TCP
- L'adresse du client doit être fournie
- Pour chaque nouvelle connexion entrante la primitive fournit un pointeur sur un nouveau descriptif de socket qui est du même modèle que le descriptif précédemment crée (un nouveau numéro de la socket avec le même port et la même adresse).
- Profil d'appel de la primitive

```
#include <sys/types.h>
#include <sys/socket.h>
int  accept (int socket, struct sockaddr_in * addr_client, int addr_lg_client)
```

35

Le couple listen-accept (1)



Le couple listen-accept – un exemple (2)

```
#include <sys/socket.h>
// adresse socket du client appelant

Struct sockaddr_in    from;
Nb_max = 5;
If (listen (s, nb_max) <0)
// cas d'erreur

//on accepte des appels successifs. Pour chaque appel on crée un processus
If ((g == accept (f, &from, sizeof(from)))<0)
// cas d'erreur

If (fork() == 0)
//processus traitant de connexion
```

37

Primitive fork

- La primitive ***fork*** crée un nouveau processus (appelé *fils*) à l'identique du processus *père*
- Pour les sockets il faut que les deux processus père et fils ferment la socket pour qu'elle soit complètement libérée
- la valeur retournée par la primitive *fork* : père = PID (Processus Identifier) du fils ; fils = 0
- Profil d'appel de la primitive

```
#include <sys/socket.h>
int  fork ()
```

38

Primitives send et recv

- Les primitives ***send*** et ***recv*** (bloquantes) permettent l'échange des données lors que la connexion est établie
- L'échange des données est contrôlé par le paramètre « flage »
- les flages sont les options (données urgents = SMG_OOB; données ne doivent pas passer par routeur =SMG_DONTROUTE….)

- Profil d'appel des primitives

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

int  send (int socket, char *message, int msg_len, int flags);
int recv (int socket, char *buffer, int buf_len, int flags);
```

39

Primitives sendto et recvfrom

- Les primitives ***sendto*** et ***recvfrom*** permettent l'échange des données avec UDP (mode non connecté)
- Il faut préciser l'adresse destinataire dans toutes les primitives *sendto* et l'adresse émetteur dans toutes les primitives *recefrom*

- Profil d'appel des primitives

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

int  sendto (int socket, char *message, int msg_len, int flags, struct sockaddr
    *addr_dest, int addr_len);
int recvfrom (int socket, char *buffer, int buf_len, int flags, struct sockaddr
    *addr_emet, int addr_len);
```

40

Primitives shutdown et close

- Les primitives ***shutdown*** et ***close*** permettent la fermeture d'une connexion
- *Shutdown* permet de préciser le sens de fermeture, la connexion n'est pas complètement fermée.
- Sens:
  - 0: fermeture en entrée;
  - 1: fermeture en sortie;
  - 2: fermeture dans les deux sens (= close)
- *close* permet de fermer la connexion et de libérer le client et le serveur
- la fermeture est symétrique (le client ou le serveur peut commencer la fermeture)
- retourne -1 s'il y a une erreur

- Profil d'appel des primitives

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

int close (int socket); **int shutdown (int socket, int sens)**

41

Primitives read et write

- Les primitives ***read*** et ***write*** permettent la réception et l'émission de données
- Elles sont identiques aux primitives habituelles d'E/S en C

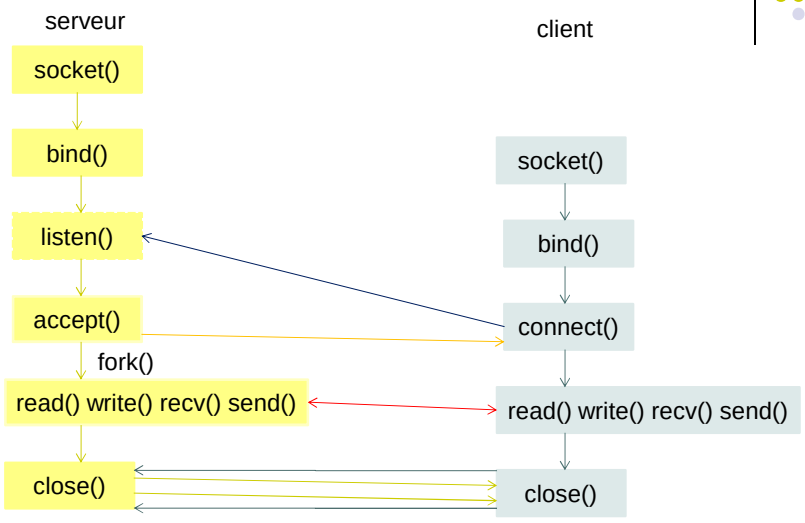
- Profil d'appel des primitives

```
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
```

int read (int socket, char * buffer, int buf_len); //retourne le nombre d'octets reçu
int write (int socket, char * buffer, int buf_len) //retourne le nombre d'octets envoyés

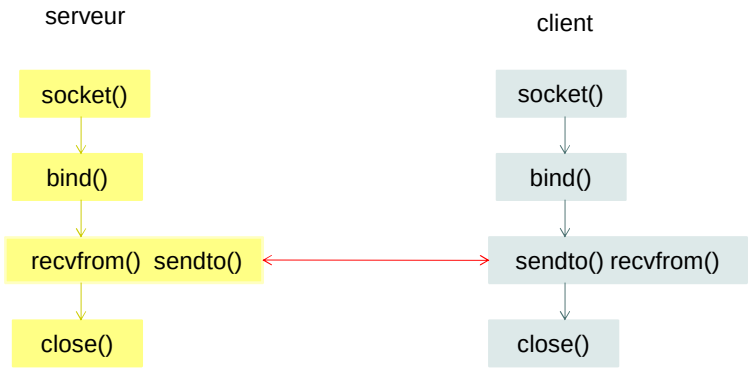
42

Résumé - socket avec TCP



43

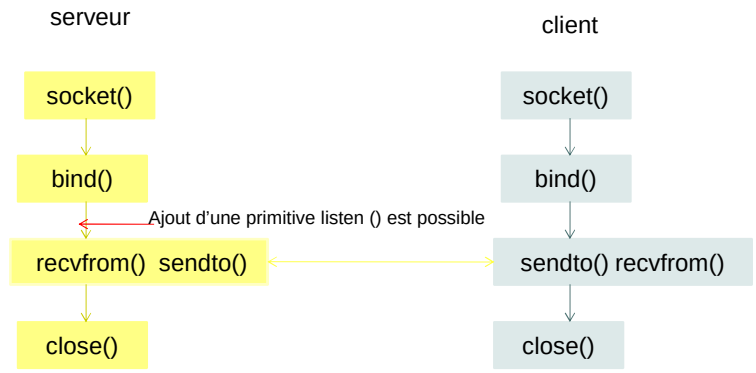
Résumé - socket avec UDP (1)



44

* le récepteur possède un buffer d'une taille fixe

Résumé - socket avec UDP (1)



* le récepteur possède un buffer d'une taille fixe

45

Conclusion (1)

- Socket est un nouveau point d'accès à la communication
- Interfaces des sockets sont des librairies de primitives d'accès aux deux protocoles transport d'Internet : TCP et UDP. L'interface a été développée par l'Université de Berkeley, donc, Berkeley Socket Interface
- Interface est disponible dans les différents langages (C. Java…).

- Elle permet de programmer des applications client-serveur
- le serveur se met en attente de demandes (passif), listen(), par exemple
- Le client initie le dialogue par une demande (actif)
- la connexion établi est une connexion bidirectionnelle et symétrique

- les avantages UDP par rapport à TCP: il est plus rapide et moins coûteux pour le réseau; c'est intéressant si la communication « question/réponse »
- les inconvénients UDP par rapport à TCP: pas de contrôle de flux (perte de paquets et l'erreur de transmission…).

46

Conclusion (1)

- Socket est un nouveau point d'accès à la communication
- Interface des sockets sont des librairies de primitives d'accès aux deux protocoles transport d'Internet: TCP et UDP. L'interface a été développée par l'Université de Berkeley, donc, Berkeley Socket Interface
- Interface est disponible dans les différents langages (C. Java…).

- Elle permet de programmer des applications *client-serveur*
  - le serveur se met en attente de demandes (passif), listen(), par exemple
  - le client initie le dialogue par une demande (actif)
- la connexion établie est une connexion bidirectionnelle et symétrique

- les avantages UDP par rapport à TCP : il est plus rapide et moins coûteux pour le réseau ; c'est intéressant si la communication « question/réponse »
- les inconvénients UDP par rapport à TCP : pas de contrôle de flux (perte de paquets et l'erreur de transmission…).

47

Conclusion (2)

- Ce cours n'aborde que la base d'interfaces socket. Le standard RFC 2553 « Basic Socket Interface Extensions for IPv6 » en 1999 définit l'interface socket si l'adressage IP est en IPv6.
  - Interface socket IPv6 doit être compatible avec celui IPv4
  - Des nouvelles fonctionnalités sont ajoutées

- La sécurité de connexion par l'interface socket est assurée par SSL (Secure Sockets Layer)

48

Bibliographe

- R. Gilligan and al, RFC 2553 « Basic Socket Interface Extensions for IPv6 », RFC 2553, IETF, 1999
- I. Baz Castillo and al, « The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP) », en cours, IETF 2012
- S. Turner and al, « Prohibiting Secure Sockets Layer (SSL) Version 2.0 », RFC 6176, IETF, 2011

- G. Florin, « supports en RSX102 » années 2000-2008