Communication en mode message asynchrone

Interfaces de mode message asynchrone

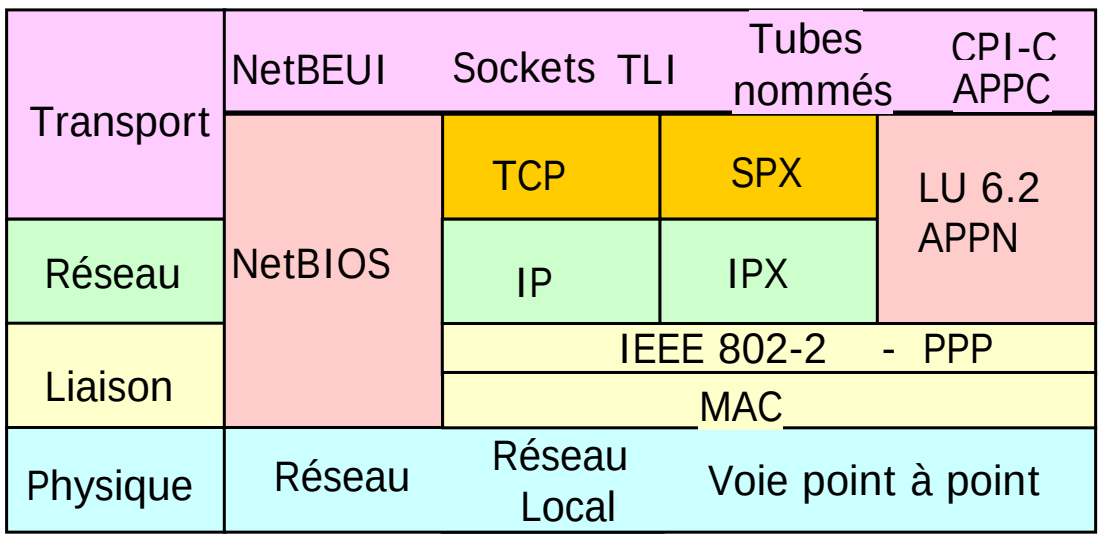
Exemple de l'interface socket Berkeley

Introduction: interfaces de mode message dans les réseaux

- **Deux approches principales (beaucoup d'approches annexes)**
  - Comme interface d’un service de niveau transport.
  - Comme interface d’un service de niveau application.
    - Un mode unifié de communication par messages pour des applications.
    - Offrant des extensions plus ou moins significatives par rapport au niveau transport.

Les interfaces de "transport" en mode message asynchrone

- Communication de bout en bout : "**End to end communication**"
- Existence **de piles de protocoles** de couches basses résumées au niveau transport: TCP/IP, IPX/SPX, NetBIOS, ...
- Existence **d'interfaces logicielles pour l'accès à des piles** de protocoles : API utilisable de préférence pour plusieurs piles
- **Sockets, TLI, NetBEUI, APPC/CPI-C,** ...



Les API de "transport" (1)

- **"Sockets" (prises)**
  - Interface de programmation pour la suite TCP/IP. 1981. Système UNIX Berkeley BSD.
  - Le standard UNIX de facto.
  - L'API sockets sous Windows est baptisée WinSock.
- **TLI ("Transport Layer Interface")**
  - Proposition AT&T d'une interface TCP IP plus performante et plus indépendante du réseau sous-jacent (1986, 25 primitives).
- **NetBEUI"NetBios Extended Basic Input Output System"**
  - Interface de programmation introduite en 1984 pour le PC Network (utilisé pour la pile NetBIOS") pile de protocoles pour les réseaux locaux (IBM et Microsoft).
  - Utilisé pour la pile de communication SPX/IPX de Novell ("Sequenced Packed Exchange/ Internet Packet Exchange").

Les API de "transport" (2)

- **Les tubes nommés ("Named pipes")**
  - Interface d'échange entre processus IPC ("InterProcessCommunication") introduite sous UNIX BSD pour étendre la notion de tube.
  - Ils permettent de considérer les échanges réseaux comme des accès fichiers (disponibles sur TCP/IP, SPX/IPX).
- **APPC et CPI-C**
  - "Advanced Program to Program Communication" et "Common Programming Interface for Communication“.
  - 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").
  - APPC est une interface de programmation pour l'accès aux services SNA LU6.2 pour tous types de produits (60 primitives versions incompatibles).
  - CPI-C (40 primitives) simplifie l'interface APPC et masque les différences.

Interfaces d'application en mode message : notion de MOM

- **MOM : "Message Oriented Middleware" (Intergiciel orienté messages)**
  - **Origine 1993** : Consortium autour de IBM puis normalisation (OSI).
  - **A) Une Interface universelle pour des échanges en mode message** asynchrone (**qui cache les différentes API transport**).
  - **B) 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**
  - **MQ series** : Message Queues Series IBM.
  - **MSMQ** : Microsoft Message Queueing.
  - Autres produits: TIBCO, SUN.

Interfaces en mode message asynchrone

Exemple d'un service pour TCP et UDP : les sockets Berkeley

Généralités interface "socket"

- **Définition en 1982** : une interface de programmation d'applications réseaux (API) pour la version UNIX BSD.
- **Existence de plusieurs autres interfaces réseaux :** TLI, NETBEUI, ...
- **Objectifs généraux**
  - Fournir des moyens de communications entre processus (IPC) **utilisables en toutes circonstances**: échanges locaux ou réseaux.
  - Cacher **les détails d'implantation** des couches de transport aux usagers.
  - Si possible cacher les **différences entre protocoles de transport hétérogènes** sous une même interface (TCP, Novell XNS, OSI)
  - Fournir une interface d'accès qui se rapproche **des accès fichiers pour simplifier la programmation** => En fait des similitudes et des différences importantes entre programmation socket et fichier.

Choix de conception des sockets

- **Une "socket" (prise) est un point d'accès de service pour des couches transport :** essentiellement TCP/UDP mais aussi d'autres protocoles (OSI, DECNET...).
- **Une socket est analogue à un objet** (de communication)
  - **Un type**:
    - Pour quel protocole de transport est-elle un point d'accès de service?
    - Quelle est la sémantique de l'accès au service?
  - **Un nom:** identifiant unique sur chaque site (en fait un entier 16 bits).
  - **Un ensemble de primitives** : un service pour l'accès aux fonctions de transport.
  - **Des données encapsulées :**
    - **un descriptif** (pour sa désignation et sa gestion)
    - **des files d'attente** de messages en entrée et en sortie.

Désignation des sockets

- **Identifier complètement une socket** dans un réseau et pour une couche transport : un couple NSAP,TSAP.

  Exemple Internet TCP: Numéro de port , Adresse IP
- **Gestion par l'IANA**
- **A) Numéros de ports réservés :** numéros de ports réservés pour des services généraux **bien connus** ou **"well-known ports"** (numéros inférieurs à 1023).
  - **Exemples ports UDP** : Echo server: 7, TFTP: 69.
  - **Exemples ports TCP** : Telnet: 23, DNS: 53, HTTP: 80.
- **B) Numéros de ports enregistrés ('registered'):** (entre 1024 et 49151) pour des applications ayant fait une demande.
- **C) Numéros de ports privés ('private') (dynamiques) :** les autres numéros entre 49152 et 65535 qui sont attribués dynamiquement aux sockets utilisateurs (clients).

Choix de conception des sockets avec TCP

- TCP est un transport **fiable** en **connexion** et en mode **bidirectionnel point à point**.
- **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 échange TCP est 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.
  - L'option **"push"** qui permet de demander l'émission immédiate d'un segment.
  - L'option **"urgent"** qui devrait permettre l'échange rapide de données exceptionnelles avec signalement d'arrivée.

Choix de conception des sockets avec UDP

- UDP est une couche transport **non fiable, sans connexion,** en mode **bidirectionnel** et **point à point**.
- **L'adresse UDP d'une socket** (Numéro de port UDP , Adresse IP) sur l'Internet à la même forme que celle d'une socket TCP.
- **Mais les deux ensembles d'adresses sont indépendants** : une communication UDP n’a rien à voir avec une communication TCP.
- Un échange UDP est sans connexion (échange de **datagrammes)**.
- Les zones de données qui correspondent à des envois successifs sont **respectées** à la réception.

Exemple des protocoles et services de transport INTERNET

Les primitives de l'interface socket

Exemple en langage C en UNIX.

Primitive socket

- **Permet la création d'un nouveau point d'accès de service transport:**
  - définition de son type.
  - allocation de l'espace des données.
- **Trois paramètres d'appel**
  - **Famille** d'adresses réseaux utilisées locale, réseau IP, réseau 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 de la primitive en C**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int socket (int famille, int type, int protocole);
```

Approfondissement des paramètres de la primitive socket

- **Paramètre Famille**
  - AF_UNIX : Communication locale (i-node)
  - AF_INET : Communication Internet
  - AF_ISO : Communication ISO

    ....
- **Paramètre Type**
  - **SOCK_STREAM** : Flot d'octets en mode connecté
    (ne préserve pas les limites de l'enregistrement)
  - **SOCK_DGRAM** : Datagramme en mode non connecté
    (préserve les limites de l'enregistrement)
  - **SOCK_RAW** : Accès aux couches basses.
  - **SOCK_SEQPACKET** : Format structuré ordonné
    (protocoles différents de l'Internet)
- **Paramètre Type de protocole**

Valeur	Relation avec le paramètre type
IPPROTO_TCP	SOCK_STREAM
IPPROTO_UDP	SOCK_DGRAM
IPPROTO_ICMP	SOCK_RAW
IPPROTO_RAW	SOCK_RAW

Primitive bind

- **Primitive pour l'attribution d'une adresse de socket** à un descripteur de socket.
- **N’est pas réalisé** lors de la création du descriptif (socket).
  - Un serveur (qui accepte des connexions) doit définir sur quelle adresse.
  - Un client (qui ouvre des connexions) n'est pas forcé de définir une adresse (qui est alors attribuée automatiquement).
- **Profil d'appel de la primitive**

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (    int s,
              struct sockaddr_in *mon_adresse,
              int longueur_mon_adresse        )
```

- **Trois paramètres d'appel**
  - Numéro du descriptif de Socket (s).
  - Structure de donnée adresse de socket Pour internet type sockaddr_in.
  - Longueur de la structure d'adresse.

Approfondissement concernant la primitive bind

- **Descripteur 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 d'exécution de "bind" pour les protocoles Internet.**

```
struct servent *sp
struct sockaddr_in sin

/* Pour connaître le numéro de port */
if((sp=getservbyname(service,"tcp")==NULL)
/* cas d'erreur */

/* Remplissage de la structure sockaddr */
/* htonl convertit dans le bon ordre */
/* INADDR_ANY adresse IP du site local */
sin.sin_family= AF_INET;
sin.sin_port = sp -> s_port;
sin.sin_addr.s_addr=htonl(INADDR_ANY):

/* Création d'une socket internet */
if ((s=socket(AF_INET,SOCK_STREAM,0))<0)
/* cas d'erreur */

/* Attribution d'une adresse */
if (bind(s, &sin, sizeof(sin)) < 0)
/* cas d'erreur */
```

Primitive listen

- **Utilisé dans le mode connecté lorsque plusieurs clients** sont susceptibles d'établir plusieurs connexions avec un serveur.
- **Indique le nombre d'appel maximum attendu** pour réserver l'espace nécessaire aux descriptifs des connexions.
- **La primitive listen est immédiate** (non bloquante).
- **Profil d'appel :** int listen (int s , int max_connexion)

  s : Référence du descripteur de socket

  max_connexion : Nombre maximum de connexions.

Primitive accept

- **La primitive accept** permet de se bloquer en attente d'une nouvelle demande de connexion (donc en mode connecté TCP).
- **Après accept,** la connexion est complète entre les deux processus.
- **Le site qui émet accept exécute une ouverture passive.**
- **Pour chaque nouvelle connexion entrante** la primitive fournit un pointeur sur un nouveau descriptif de socket qui est du même modèle que le descritif précédemment créé.
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int accept (    int ns,
                struct sockaddr_in *addr_cl,
                int lg_addr_cl)

ns              : Référence nouvelle socket
addr_cl         : L'adresse du client.
lg_addr_cl: La longueur de l'adresse.
```

Approfondissement concernant les primitives listen et accept

- **Exemple de code UNIX :** pour un serveur qui accepte des connexions successives et qui créé un processus pour traiter chaque client.

```
#include <sys/socket.h>
/* Adresse socket du client appelant */
struct sockaddr_in from;
quelen = ... ;
if (listen (s, quelen) <0 )
        Cas d'erreur
/* On accepte des appels successifs */
/* Pour chaque appel on créé un processus */
if((g=accept(f,&from,sizeof(from)))<0)
        Cas d'erreur
if ( fork ...
/* Processus traitant de connexion*/
```

Primitive connect

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

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

s              : La référence de la socket
addr_serv      : L'adresse du serveur.
lg_addr_serv : La longueur de l'adresse.
```

Primitives send, recv

- **Les primitives send, recv (bloquantes)** permettent l'échange effectif des données.
- **Le profil d'appel est identique à celui des primitives read et write sur fichiers** avec un quatrième paramètre pour préciser des options de communications.
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int send  (int s, char *zone,
           int lg_zone, int options_com)
int recv  (int s,char *zone,
           int lg_zone, int options_com)
s                          : La référence de la socket
zone                       : La zone à échanger.
lg_zone          : La longueur de la zone.
options_com      : Les options (données urgentes , ....)
```

Primitives sendto, recvfrom

- **Les primitives sendto, recvfrom** permettent l'échange des données dans le mode non connecté UDP.
- **On doit préciser l'adresse destinataire** dans toutes les primitives **sendto** et **l'adresse émetteur** dans les **recvfrom**.
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto (int s,
            char *zone,
            int lg_zone,
            int options_com,
            struct sockaddr_in *addr_dest,
            int lg_addr)
int recvfrom (int s,
            char *zone,
            int lg_zone,
            int options_com,
            struct sockaddr_in *addr_emet,
            int *lg_addr)
addr_dest  : L'adresse du destinataire.
addr_emet  : L'adresse de l'émetteur.
lg_addr    : La longueur de l'adresse.
```

Primitives shutdown, close

- **Shutdown** permet la terminaison des échanges sur une socket suivi de la fermeture de la connexion :
  - **Profil d'appel** : int shutdown(s , h); Pour la socket s.
  - **h = 0** : l'usager ne veut plus recevoir de données
  - **h = 1** : l'usager ne veut plus envoyer de données
  - **h = 2** : l'usager ne veut plus ni recevoir, ni envoyer.
- **Close :** Permet la fermeture d'une connexion et la destruction du descriptif.
  - Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>
int close ( int s )
```

Résumé : Interface socket

- **Fonctionnement en TCP**
  - **- Serveur.**
    - socket
    - bind
    - listen
    - accept
    - recv, send
    - close
  - **- Client.**
    - socket
    - connect
    - recv, send
    - close
- **Fonctionnement en UDP**
  - socket
  - recvfrom, sendto
  - close