E.N.S.I.M.A.G

Année Spéciale Informatique
1993/1994

PROGRAMMATION RESEAU

SUR TCP/IP

L'INTERFACE DES SOCKETS

Responsable de projet :
Serge Rouveyrol

Projet réalisé par

Jazouli Abdel illah
Radi Nour-Eddine
Zghal Tarek

REMERCIEMENT

Nous tenons à remercier vivement M Rouveyrol, qui a bien voulu nous divertir de la mort en nous proposant cette étude sur la programmation réseau.
Nous le remercions aussi pour son ouverture et pour ses précieux conseils.

PREMIERE PARTIE
PRESENTATION DE TCP/IP

INTRODUCTION

I - LE MODELE DE REFERENCE OSI

II - LE MODELE SIMPLIFIE DU PROTOCOLE TCP/IP

1 - Présentation de TCP/IP
    1-1 La couche TCP
    1-2 Le protocole UDP
    1-3 La couche IP
2 - La couche ETHERNET
3 - La couche Application

III - PRESENTATION DES UNITES DE DONNEES DE PROTOCOLE CIRCULANT ENTRE LES COUCHES

1 - Le datagramme IP
2 - Le paquet UDP
3 - Le paquet TCP

DEUXIEME PARTIE
LES SOCKETS

I - LE MODELE CLIENT/SERVEUR

1 - Définitions
2 - Applications orientées connexion ou sans connexion

II - LES SOCKETS
INTRODUCTION : LES SOCKETS RACONTES AUX ENFANTS

III - ADRESSES DE SOCKETS

1 - Les famille d'adresse
2 - Les structures d'adresse

IV - LES APPELS SYSTEMES

1 - L'appel système socket
2 - L'appel système bind
3 - L'appel système connect
4 - L'appel système listen
5 - L'appel système accept

V - ECHANGES D'INFORMATIONS SUR UN SOCKET

1 - Emission d'information
2 - Réception d'information

VI - ENCHAINEMENT DES APPELS SYSTEMES

1 - Mode connecté
2 - Mode non connecté

TROISIEME PARTIE
LES PROGRAMMES DEVELOPPES

- Programme clients.c
- Programme setsock.c
- Pogramme DialogueTCP.c
- Programme DialogueUDP.c
- Programme Serveur.c
- Programme ServeurTCP_iteratif.c
- Programme ServeurTCP_parallèle.c
- Programme ServeurUDP_parallèle.c
- Programme DialogueTCP_serveur.c
- Programme DialogueUDP_serveur.c

BIBLIOGRAPHIE

Ce document est accessible par ftp anonyme sur la machine ftp.imag.fr fichier /pub/DOC.UNIX/SOCKETS/sockets.ps.
Les exemples se trouvant dans ce document se trouvent dans le fichier /pub/DOC.UNIX/SOCKETS/

INTRODUCTION

Pour comprendre les réseaux, il est important de savoir que la recherche et le développement se sont effectués en trois temps. Avant les années 1960, la question fondamentale était : ***<< Comment est-il possible d'acheminer, de façon fiable et efficace, des débits le long d'un support de transmission ? >>***. Les résultats ont inclus le développement de la théorie du signal.

Aux environ des années 1965, l'intérêt s'est concentré sur la commutation de paquets et la question est devenue : ***<< Comment est-il possible de transmettre des paquets , de façon efficace et fiable, sur un support de communication?>>.*** Les résultats ont abouti au développement des technique de communication par paquets et des réseaux locaux.

Des années 1975 jusqu'à aujourd'hui, l'attention s'est portée sur les architectures de réseaux et sur la question : ***<< Comment fournir des services de communication à travers une interconnexion de réseaux >>.*** Les résultats incluent les techniques d'interconnexion, les modèles de protocoles en couches, les datagrammes, les services de transport en mode connecté et le modèle client/serveur.

Ce rapport se focalise sur la programmation d'applications basées sur le modèle **client/serveur** utilisant les protocoles **TCP/IP** Internet du D.A.R.P.A. (Défense Advanced Research Projets Agency).

Un protocole est un ensemble de règles et de conventions nécessaires pour la communication entre ordinateurs. Comme ces protocoles sont complexes, ils sont découpés en plusieurs couches pour faciliter leurs implémentation. Chacune étant construite sur la précédente. Le nombre de couches, leur nom et leur fonction varie selon les réseaux. Cependant, dans chaque réseau, l'objet de chaque couche est d'offrir certains services aux couches plus hautes.

La couche N d'une machine gère la conversation avec la couche N d'une autre machine en utilisant un protocole de niveau N. En réalité, aucune donnée n'est transférée directement de la couche N d'une machine à la couche N de l'autre machine mais chaque couche passe les données et le contrôle à la couche immédiatement inférieure, jusqu'à la plus basse.

En dessous de la couche 1 se trouve le support physique qui véhicule réellement la communication. L'ensemble des couches et protocoles est appelé l'architecture du réseau**.**

I - LE MODELE DE REFERENCE OSI

Sept couches sont définies dans le modèle normalisé connu sous le nom de modèle OSI (Open Systems Interconnection ou Interconnexion de systèmes ouverts). Cette norme de communication repose sur l'empilement de 7 couches pouvant communiquer verticalement entre elles (fig 1).

Les différentes couches du modèle sont présentées ci dessous :

7 APPLICATION
6 PRESENTATION
5 SESSION
4 TRANSPORT
3 RESEAU
2 LIAISON
1 PHYSIQUE

figure 1: Modèle OSI en 7 couches

1- *La couche Physique*

C'est la couche de niveau 1 du modèle . Elle offre les services de l'interface entre l'équipement de traitement informatique (ordinateur ou terminal) et le support physique de transmission. L'unité de transfert gérée par cette couche est l'information élémentaire binaire.

2- *La couche de Contrôle de la liaison*

Pour qu'une succession de bits prenne sens, cette couche définie des conventions pour délimiter les caractères ou des groupes de caractères. Elle a, donc pour but d'acheminer sur une voie physique toutes les informations qui lui sont transmises par la couche supérieure. L'unité de traitement à ce niveau est appelée trame de données (de quelques centaines d'octets) .

3- *La couche Réseau*

Cette couche permet de gérer le sous-réseau, la façon dont les paquets (ensemble de trames) sont acheminés de la source au destinataire. La couche réseau a de ce fait pour rôle essentiel d'assurer les fonctions de routage (fonction détaillée dans la suite) .

4- *La couche Transport*

La fonction de base de cette couche est d'accepter des données de la couche session, de les découper, le cas échéant, en plus petites unités, et de s'assurer que tous les morceaux arrivent correctement de l'autre côté.

La couche transport crée une connexion réseau par connexion transport requise par la couche session. Elle détermine également le type de services à fournir à la couche session et, finalement, aux utilisateurs du réseau. Le type de service est déterminé à la connexion.

5- *La couche Session*

Cette couche effectue la mise en relation de deux applications et fournit les protocoles nécessaires à la synchronisation des échanges, leur arrêt et leur reprise. Elle permet d'attribuer un numéro d'identification à l'échange d'une suite de messages entre applications.

6- *La couche Présentation*

Elle permet la traduction des données transmises dans un format , commun à toutes les applications, défini par la norme afin de permettre à des ordinateurs ayant différents formats de communiquer entre eux.

7- *La couche Application.*

C'est la couche de plus haut niveau : le niveau 7. Elle exécute des fonctions de traitement diverses et permet la normalisation de services d'applications typées telles que :

- Le transfert de fichiers (**FTP**),
- La soumission de travaux à distant (**TELNET**),
- La messagerie (**MAIL, TALK**), ...

La communication entre deux machines distantes se résume par la figure 2.

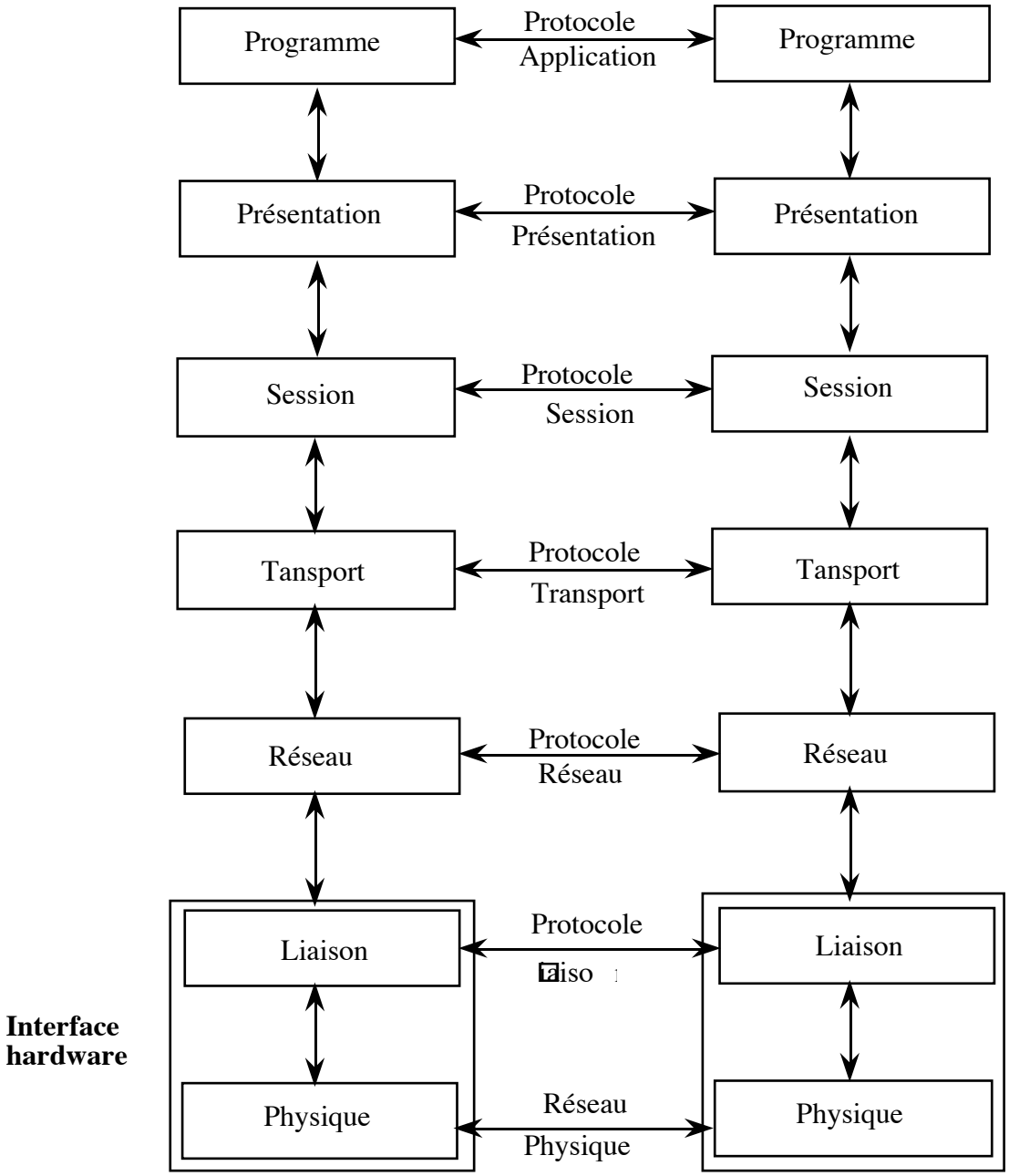

Fig2

II- MODELE SIMPLIFIE DU PROTOCOLE TCP/IP

Introduction

Les protocoles existants ne suivent pas toujours le modèle OSI. En effet certains protocoles n'utilisent que certaines couches du modèle. Généralement ils se limitent à une représentation en 5 couches. Le plus répandu de ces protocoles est TCP/IP ou UDP/IP.

TCP/IP ou UDP/IP peuvent être utilisés pour communiquer au sein d'un ensemble quelconque de réseaux interconnectés.

Les protocoles tels que TCP/IP définissent les règles utilisées pour échanger les unités de données, les détails de leurs structure et indiquent comment gérer le cas d'erreur. De plus, ils nous permettent de traiter la question des normes de communication indépendamment du matériel réseau de tel ou tel constructeur.

De point de vue de l'utilisateur, une interconnexion TCP/IP se présente comme un ensemble de programmes d'application qui utilisent le réseau pour effectuer des tâches utiles sans comprendre la technologie TCP/IP, la structure de l'interconnexion sous-jacente, ni même le chemin emprunté par les données pour atteindre leur destination. Les services d'application les plus populaires et les plus répandus comprennent :

- Le courrier électronique.
- Le transfert de fichier.
- La connexion à distance.

Le modèle en couche utilisant TCP/IP ou UDP/IP est représenté sur la figure 3.
L'interface hardware représente les couches 1 et 2 du modèle OSI.
Généralement les couches 1 et 2 sont représentées par Ethernet mais on peut aussi avoir token-ring, X25 ....
La couche réseau est représentée par IP (Internet Protocol).
La couche transport est représentée par TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol) .
La dernière couche est la couche application.

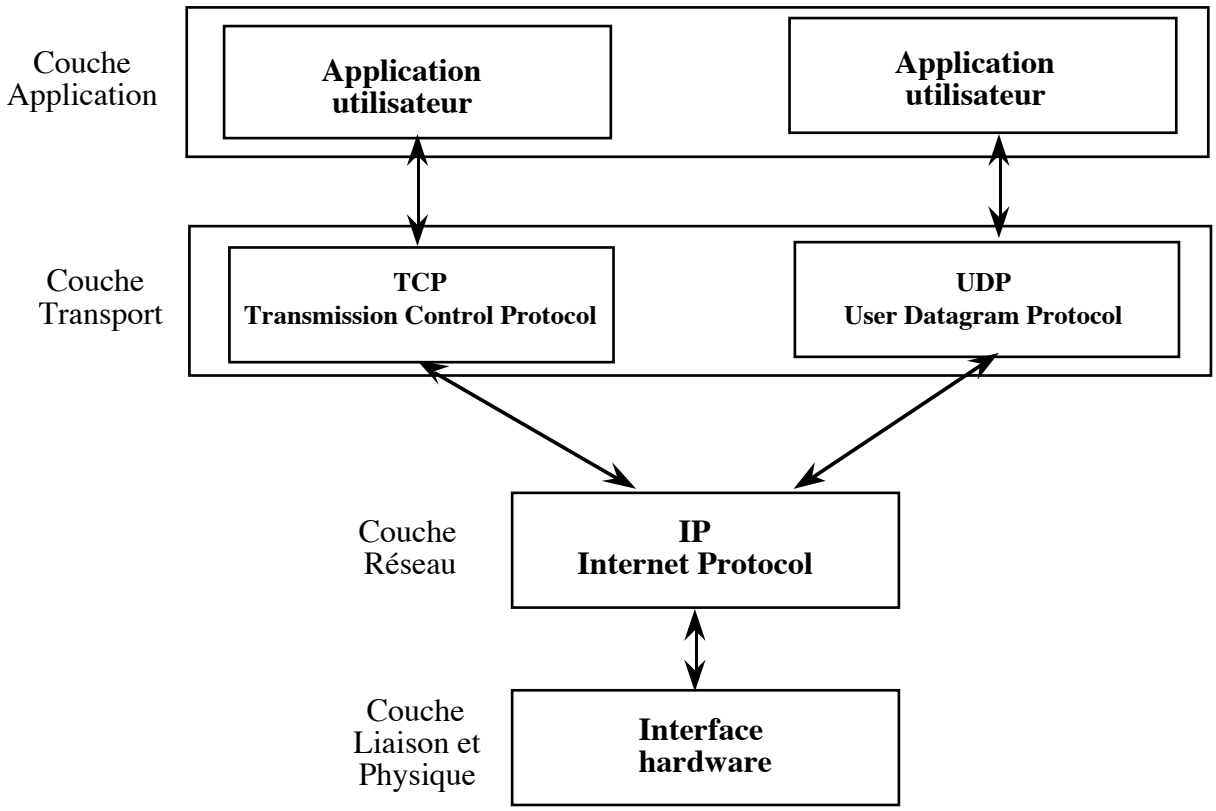

Fig3 RESEAU EN COUCHES UTILISANT LE PROTOCOL TCP/IP

Dans la partie suivante on va détailler les protocoles Ethernet, TCP, UDP et IP.

1 - PRESENTATION DE TCP/IP

TCP/IP est un ensemble de protocoles organisés en couche. Pour éclaircir cette conception en couche il est intéressant d'étudier un exemple.

Une situation typique est l'envoi d'un mail. Le protocole définit un ensemble de commandes qu'une machine doit envoyer à une autre, i.e. les commandes nécessaires pour spécifier l'émetteur du message, le destinataire, et le contenu du message. En fait ce protocole suppose l'existence d'un moyen de communication fiable entre les deux machines.

Le mail définit seulement un ensemble de commandes et de messages à envoyer à l'autre machine. Il est sensé être utilisé avec TCP et IP. TCP assure la livraison des données à la machine distante et a la charge de retransmettre ce qui est perdu par le réseau. Si un message est plus grand qu'un datagramme (ex : le texte d'un mail), TCP se charge de le découper en plusieurs datagrammes, et s'assure qu'ils sont tous correctement transmis.

Comme plusieurs applications sont sensées utiliser ces fonctions, elles sont mises ensembles dans un protocole séparé.

On peut imaginer TCP comme une librairie de routines que les applications peuvent utiliser quand elles désirent avoir une communication réseau fiable avec une machine distante. De la même façon que les applications demandent des services à la couche TCP, celle ci demande des services à la couche IP.

Comme pour TCP, on peut imaginer IP est une librairie de routines que TCP appelle. Certaines applications n'utilisent pas TCP. Elles peuvent cependant faire appel aux services fournis par IP.

C'est cette stratégie de construction par niveau de protocole qu'on appelle modèle en couche. On présente donc l'application, TCP et IP comme étant des couches séparées, chacune d'entre elles demandant des services à la couche en dessous. Généralement les applications TCP/IP utilisent 5 couches:

- un protocole d'application comme le mail par exemple.
- un protocole comme TCP qui propose des services utilisés par plusieurs applications.
- IP qui fournit les services de livraison des datagrammes.
- un protocole spécifique au moyen de transmission utilisé.

TCP/IP est basé sur un modèle qui suppose l'existence de plusieurs réseaux indépendants qui sont interconnectés par des passerelles et que l'utilisateur doit avoir la possibilité d'accéder aux ressources de chacune des machines présentes sur cet ensemble de réseaux interconnectés.

Les datagrammes transiteront souvent par des dizaines de réseaux différents avant d'être délivrées à leur destinataire. Le routage (c'est à dire la manière d'acheminer les données) nécessaire pour accomplir ceci doit être invisible à l'utilisateur.

Il y a une correspondance unique entre le nom d'une machine et son adresse internet. Cette correspondance se trouve dans un fichier /etc/hosts. Voici une partie de ce fichier sur ensisun.

```
#       fichier ~comimag/fede/hosts_entete
#       contient l'entete du /etc/hosts de imag

# LISTE DES MACHINES DU RESEAU DE L'IMAG (/etc/hosts)

129.88.40.4   woody              # MAC VX C210 G. Kuntz p. 4321
129.88.40.5   pixou              # SUN 4/110 C205 J. Lemordant
129.88.40.6   filochard          # SUN 3/60 C205 Lemordan (4498)
129.88.40.7   riri               # TX NCD16 C205 Lemordant
129.88.40.8   goofy              # NEXT Color 12/400  C207
129.88.40.9   zaza               # TX NCD MCX17 p.5321 Kuntz C210
129.88.40.10  safira             # MAC IIsi C. Le Faou p      4299
129.88.40.11  droopy droopy1     # SPARC 10/41 C105A P.
129.88.40.12  cubitus            # INTEL GX C205 J. Lemordant
129.88.40.13  esmeralda          # SUN 4/75 C207      C. Le Faou
129.88.40.14  loulou             # TX NCD17C couleur C205
129.88.40.15  aguamarina         # SUN 4/65 1+ C201 C. Le Faou
129.88.40.16  granada            # SUN 4/40 C205      C. Le Faou
129.88.40.17  gemma              # TX NCD 19r H.Bouamama C205
129.88.40.20  ametista           # SUN 4/40 C201      C. Le Faou
129.88.40.23  topazio            # MAC IIci C. Le Faou p      4299
```

```
129.88.100.73        tetiaroa              # BULL DPX2/25    K51  routeur   C
129.88.102.9  pc2                  # essais routage
```

La seule information dont doit disposer l'utilisateur pour pouvoir accéder à une machine distante est son adresse internet. Une adresse internet est un nombre de 32 bits généralement écrit sous forme de 4 nombres décimaux représentant chacun 8 bits de l'adresse (ex : première colonne).

Généralement on désigne un système par un nom plutôt qu'une adresse. Dans ce cas, la machine locale consulte une base de données et retrouve l'adresse internet correspondante au nom spécifié.

TCP/IP est conçu avec une technologie sans connexion. Les informations sont envoyées comme une séquence de datagrammes indépendants.
Un datagramme est un ensemble de données qui est envoyé comme un seul message. Chaque datagramme est envoyé indépendamment des autres. Par exemple supposons qu'on ait à transférer un fichier de 10.000 octets.

Les protocoles ne peuvent pas supporter de datagramme de 10.000 octets. Le protocole se chargera donc de découper le fichier en 200 datagrammes de 500 octets et de les envoyer à leur destinataire. A l'autre bout, ces datagrammes seront rassemblés en un fichier de 10.000 octets.

Il est important de noter que lors du transit de ces données, le réseau n'a aucune idée sur la relation qui existe entre ces datagrammes.
Il est donc possible que le datagramme 14 arrive avant le datagramme 13. Il est aussi possible qu'un datagramme soit perdu en cours de route.

TCP assure aussi le controle de fulx de bout en bout entre les deux extrémités de la connexion afin d'éviter des problèmes de congestion.

1.1 LA COUCHE TCP

Deux protocoles sont invoqués lors de l'utilisation de datagrammes TCP/IP.
- TCP (Transmission Control Protocol) qui a pour rôle de découper les messages en datagrammes, de les réassembler à l'autre extrémité, réenvoyer les datagrammes perdus et les remettre dans le bon ordre à la réception.
- IP (Internet Protocol) qui a la charge du routage individuel des datagrammes. C'est cette couche qui gère les incompatibilités entre les différents supports physiques.

Il peut vous paraître bizarre qu'on ait parlé d'adresse IP et qu'on n'ait pas expliqué comment gérer plusieurs connexions sur une même machine. En fait il ne suffit pas de remettre un datagramme à la bonne destination. TCP doit connaître la connexion concernée par le datagramme. Cette fonction est connue sous le nom de démultiplexage.

L'information nécessaire pour le démultiplexage est contenue dans un ensemble d'entêtes. Une entête est tout simplement quelques octets supplémentaires placés au début du datagramme.

Par exemple supposons qu'on désire envoyer le message suivant:

.....

TCP découpe ce message en **... ... ...** puis ajoute une entête à chacun de ces datagrammes. L'entête contient au moins 20 octets mais les plus importants sont les numéro de port de la source et de destination et le numéro de séquence du datagramme.

Les numéros de ports servent à différencier les conversations sur une même

machine. Le numéro de séquence du datagramme permet au destinataire de remettre les datagrammes dans le bon ordre et de detecter des pertes sur le réseau.

En réalité TCP ne numérote pas les datagrammes mais les octets. Ainsi, si la taille du datagramme est de 500 octets, le premier aura le numéro 0, le second le numéro 500.

Enfin mentionnons le checksum qui représente des octets de redondance permettant de détecter certaines erreurs et de les corriger éventuellement sans avoir à demander la reémission du datagramme.

Si nous représentons l'entête par T, notre fichier de départ ressemblerait à :
T**...** T**...** T**...** T**...** T**...** T**...** T**...**

L'entête contient aussi un champ représentant un numéro d'acquittement. Par exemple l'envoi d'un datagramme avec 1500 comme numéro d'acquittement suppose la réception de toutes les données préçédant l'octet 1500.

Si l'émetteur ne reçoit pas d'acquittement pendant une certaine période de temps, il réenvoit la donnée supposée être perdue dans le réseau.

1-2 LE PROTOCOLE UDP

Rappelons que TCP découpe les messages longs en datagrammes avant de les envoyer. Seulement pour certaines applications les messages mis en jeu pourront être transportés dans un seul datagramme. Par exemple rares sont les systèmes qui possèdent une base de données complète qui résoud le problème de correspondance entre les noms des machines et leur adresse. Le système devra donc envoyer une requète à un autre système ayant cette base de données.

Cette requête a de fortes chances de nécessiter un seul datagramme pour la requête et un seul pour la réponse. Il peut donc nous paraître utile de définir un protocle qui puisse jouer le rôle de TCP mais qui ne possèderait pas toutes ses fonctions (segmentation et réassemblage ...).

L'alternative la plus connue est UDP (User Datagram Protocol). Comme pour TCP, UDP possède une entête. cette entête est positionnée avant le bloc de données. UDP délivre ensuite les données à IP qui rajoute l'entête IP, en mentionnant le type de protocole (UDP).

UDP fournit un service non fiable puisque ce protocole ne permet pas de vérifier le séquencement des datagrammes. En effet le numéro de séquence des datagrammes n'existe pas dans l'entête UDP.

fragmentation et reassemblage des datagrammes

TCP/IP a été conçu pour s'adapter à différents types de réseaux.
Seulement les concepteurs des réseaux n'ont pas toujours pris des paquets de même taille. On doit donc pouvoir s'adapter aux grandes et aux petites tailles de paquets.

En fait TCP peut négocier la taille des datagrammes. Lors de l'établissement d'une connexion TCP, les deux extrémités envoient la taille maximale qu'elles peuvent supporter pour un datagramme. C'est le plus petit de ces deux nombres qui sera considéré pour la suite de la connexion.

Le problème de cette solution est qu'elle ne prend pas en considération les passerelles entre les deux machines. Par exemple si une passerelle ne peut pas manipuler des datagrammes de la taille choisie, ils seront découpés en pièces (c'est la

fragmentation).

L'entête IP contient un champ indiquant que le datagramme a été découpé et l'information permettant de les remettre ensemble pour reformer le datagramme initial (c'est le réassemblage) .

1.3 LA COUCHE IP

TCP remet chacun de ses datagrammes à IP en lui indiquant l'adresse IP destination. Notons bien que IP n'a pas besoin de plus d'informations pour acheminer les données. IP n'est pas concernée par le contenu du datagramme et par l'entête TCP.

IP a simplement pour rôle de trouver un chemin lui permettant de livrer le datagramme à sa destination. Pour permettre aux passerelles et aux réseaux intermédiaires de faire suivre le datagramme, IP lui ajoute une entête.

Les données les plus importantes dans ce nouvel entête sont les adresses source et destination, le numéro de protocole et un autre checksum.
L'adresse source est l'adresse de votre machine. Ce qui permet au destinataire de connaître l'émetteur.
L'adresse destination est l'adresse du destinataire du message.
Le numéro de protocole identifie la couche potocolaire qui est au dessus de IP (pas forçement TCP)
Le checksum permet à la couche IP destinataire de vérifier l'intégralité des données reçues. C'est pour des raisons d'efficacitéet de sûreté que IP rajoute un checksum.
Si on note I l'entête IP, notre fichier de départ sera de la forme:
IT... IT... IT... IT... IT... IT... IT...

le routage

La description de la couche IP la présente comme étant responsable de l'acheminement des données et de leur livraison. C'est ce qu'on se propose de décrire dans ce paragraphe. Cette fonction porte le nom de routage.

On supposera que le système sait comment envoyer des données aux machines connectées sur son réseau. Le problème surgit lorsque la machine destinataire n'est pas sur le même réseau que la machine qui émet le message.

Ce problème est résolu par les passerelles. ces dernières sont généralement des systèmes qui relient un réseau à un ou plusieurs autres réseaux. Les passerelles sont donc supposés avoir plusieurs interfaces réseau.

Par exemple supposons posséder une machine unix qui possède deux interfaces Ethernet, chacune connectée à un réseau. Cette machine peut faire fonction de passerelle entre ces deux réseaux.

Le routage au niveau IP est entièrement basé sur les numéros d'adresse. Chaque système possède une table de numéros d'adresses. Pour chaque numéro réseau, correspond une passerelle. C'est le numéro de la passerelle à utiliser pour joindre le réseau désiré.

Notons que la passerelle n'a pas à être directement connectée au réseau destinataire. Elle ne représente qu'un intermédiaire pour atteindre la destination.

Quand un système désire envoyer un datagramme, il commence par voir si l'adresse destination correspond à une adresse de son réseau. Dans ce cas il emet directement le message. Dans le cas inverse le système cherche dans sa table la passerelle à laquelle il doit envoyer le message.

Plusieurs stratégies ont été développées pour minimiser la taille de la table de routage.

Une de ces statégies désigne une passerelle par défaut qui recevra le message si aucune entrée dans la table de routage ne permet d'identifier une passerelle de sortie.

Certaines passerelles à qui on s'adresse répondent " I'm not the best gateway, use this one instead". Certains systèmes utilisent ce message pour mettre à jour leur table de routage dynamique.

Sur ensisun la table de routage est obtenue en tapant la commande netstat -nr. On obtient en sortie le fichier suivant:

Routing Table:

Destination	Gateway
127.0.0.1	127.0.0.1
192.33.175.0	192.33.174.36
192.33.174.128	192.33.174.33
192.33.174.160	192.33.174.33
192.33.174.96	192.33.174.35
152.77.0.0	192.33.174.62
129.88.0.0	192.33.174.62
130.190.0.0	192.33.174.62
192.33.174.32	192.33.174.34
192.33.174.64	192.33.174.65
224.0.0.0	192.33.174.34
default	192.33.174.62

On remarquera un chemin de sortie par défaut default qui sera utilisé dans le cas où l'adresse de la destination ne figure pas dans la table.

2- LA COUCHE ETHERNET

Cette couche correspond aux couches 1 et 2 du modèle OSI.

La plupart des réseaux actuels utilisent Ethernet. Nous allons donc décrire l'entête Ethernet. Lorsqu'une donnée est envoyée sur Ethernet, toutes les machines sur le réseau voient le paquet. Il faut donc une information pour que seule la machine concernée par le message l'enregistre. Comme vous l'avez deviné, c'est un entête Ethernet qui est ajoutée.

Chaque paquet Ethernet contient un entête de 14 octets qui contient les adresses source et destination.

Chaque machine est supposée reconnaître les paquets contenant son adresse Ethernet dans le champ destinataire. Il y a donc des moyens de tricher et de lire les données du voisins, c'est une des raisons pour lesquelles Ethernet manque de sécurité.

Notons qu'il ny'a pas de relation entre l'adresse Ethernet et l'adresse Internet. Chaque machine doit posséder une table de correspondance entre adresse Ethernet et Internet.

Le checksum est calculé sur le paquet entier et ne fait pas partie de l'entête. Il est mis à la fin du paquet. Si on note E l'entête Ethernet et C le checksum, notre fichier initial ressemblera à :
EIT**...**C EIT**...**C EIT**...**C EIT**...**C EIT**...**C EIT**...**C EIT**...**C

Quand ces paquets seront reçus à l'autre extrémité, toutes les entêtes sont extraites. L'interface Ethernet extrait l'entête Ethernet et le cheksum puis passe le datagramme à la couche IP. IP retire l'entête IP, puis passe le datagramme à la couche TCP. TCP vérifie alors le numéro de séquence, et combine tous les datagrammes pour reconstituer notre fichier initial.

3 - LA COUCHE APPLICATION

Cette couche est au dessus de TCP et IP. Par exemple lorsque vous envoyez un message cette couche le délivre à TCP qui se charge de le livrer à sa destination.

Comme TCP et IP masquent tous les détails du mécanisme d'acheminement et de réemission des données, la couche application peut voir une connexion réseau comme un simple flux de données.

Notons qu'une connexion est identifiée par 4 nombres: l'adresse internet et le numéro de port de chaque extrémité de la connexion. Chaque datagramme contient ces 4 numéros. L'adresse internet est contenue dans l'entête IP et les numéros de ports dans l'entête TCP.

Par exemple on peut avoir les deux connexions suivante simultanement:

	adresses internet		ports TCP
connexion1	128.14.5.129	128.14.5.130	1234, 21
connexion2	128.14.5.129	128.14.5.130	1235, 21

La seule différence entre ces deux connexions est le numéro de port, et ceci est suffisant pour les différencier.

encapsulation des données

Les données sont transférées verticalement d'une couche à une autre en y rajoutant un entête. Cet entête permet de rajouter des informations identifiant le type de données, le service demandé, le destinataire, l'adresse source ... : c'est l'encapsulation des données.

La figure 4 illustre le mécanisme d'encapsulation.

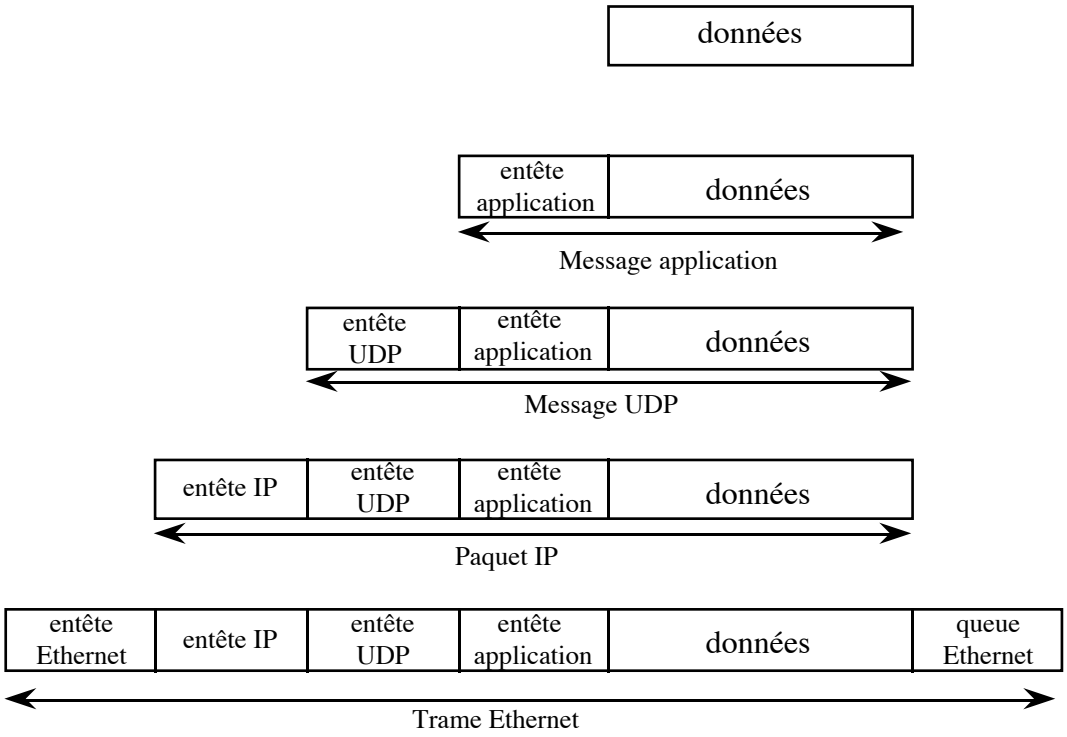

Fig 4 ENCAPSULATION DE DONNEES

La communication entre deux machines utilisant le protocole TCP/IP ou UDP/IP se résume par la suivante (figure 5).

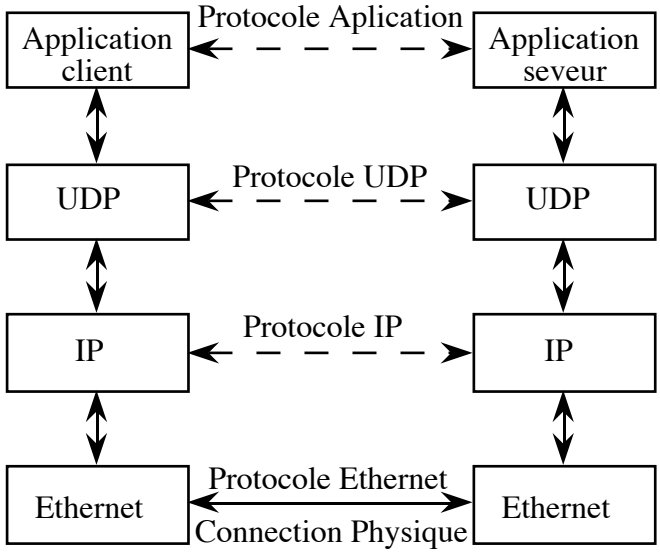

Fig 5 Connexion de deux applications sur un réseau Ethernet, UDP/IP

Chaque couche communique avec son correspondant du même niveau. Aucune donnée n'est transférée de la couche N d'une machine à la couche N d'une autre machine, mais chaque couche passe les données et le contrôle à la couche immédiatement inférieure, jusqu'à la plus basse. En dessous de la couche 1 se trouve le support physique qui véhicule réellement la communication.

III - PRESENTATION DES UNITES DE DONNEES DE PROTOCOLE CIRCULANT ENTRE LES COUCHES

1- Le datagramme IP

0	4	8	12	16	20	24	28	31
Version	I.H.L.	Type de service		Longueur totale				
Identificateur				drapeaux	offset de fragmentation			
Durée de vie		Protocol		Checksum de l'entête				
ADRESSE SOURCE								
ADRESSE DESTINATION								
OPTIONS							Rembourrage	
DONNEES								

Fig 6 DATAGRAMME IP

Version :

Ce champ indique la version du format d'en-tête utilisé. La version utilisée actuellement est la version 4.

I.H.L : (Internet Header Length)
` Longueur de l'entête Internet ; nombre de mots de 32 bits utilisés par l'entête : ce champs pointe donc sur le début de la zone données (au sens IP). La valeur minimum utilisable est 5.

Type de service :
Ce champ permet de sélectionner le type de service dont on souhaite faire bénéficier le datagramme. Ce paramètre fournit des indications abstraites sur le type de service souhaité et permet de guider la sélection des paramètres effectifs utilisés pour la transmission d'un datagramme sur un réseau donné. un des services offerts sur certains réseaux est la priorité de traitement.

Dans ce dernier cas, et selon le réseau, les datagrammes sont traités différemment selon leur niveau de priorité (par exemple, à un moment de très forte charge sur certains réseaux, seul les datagrammes de haute priorité sont acceptés).

Il y a quatre paramètres disponibles réglant le temps de transmission, le débit, la fiabilité, et la priorité. Il va de soi que ces paramètres sont relativement couplés entre eux.

format de l'octet du type de service

0 2	3	4	5	6	7
priorité	d	D	F	X	X

Bits 0 à 2 : niveau de priorité
111 contrôle réseau
110 contrôle inter-réseaux
010 Immédiat
001 Prioritaire
000 Normal.

Bit 3 : 0 = délai normal, 1 = délai réduit
Bit 4 : 0 = débit normal, 1 = haut débit
Bit 5 : 0 = fiabilité normale, 1 = haute fiabilité
Bit 6 & 7 : réservés à un futur usage

Longueur totale :
Ce champ désigne la longueur du segment IP en incluant l'entête Internet et la zone de données. Chaque machine travaillant sur TCP/IP doit être capable d'accepter des segments allant au moins jusqu'à 576 octets.

Remarque Avant de définir les champs immédiats qui suivent, il faut préciser qu'une trame IP peut être divisée dans certains réseaux lors du chemin de transmission (passage par les différents routeurs et passerelles) avant d'être réassemblée pour la réception de la machine destinataire.

Identificateur :
Pointeur utilisé pour réassembler les fragments d'un segment.

drapeaux : 3bits
Bit 0 : réservé, doit être à zéro.
Bit 1 : 0 = fragmentation autorisée, 1 = fragmentation interdite

Bit 2 : 0 = Dernier fragment, 1 = fragment à suivre.

offset de fragmentation :
Ce champ indique la portion datagramme auquel ce fragment appartient.

Durée de vie :
Ce champs indique la durée de vie maximale du datagramme dans le système internet. Une valeur nulle indique que ce datagramme doit être détruit.

Protocole :
indique le protocole utilisé dans la portion de données

Checksum de l'entête :
Vérification de la validité de l'entête IP.

ADRESSE SOURCE :
Adresse Internet de la machine émettrice du datagramme.

ADRESSE DESTINATION :
Adresse Internet de la machine destinataire du datagramme.

OPTIONS :
Ce champ a une taille variable. Il n'est pas obligatoire et peut ne pas apparaître dans le datagramme.

Rembourrage :
Utilisé en cas d'option pour amener, si besoin, la longueur de l'entête internet à un multiple de 32 bits.

2 - Le paquet UDP

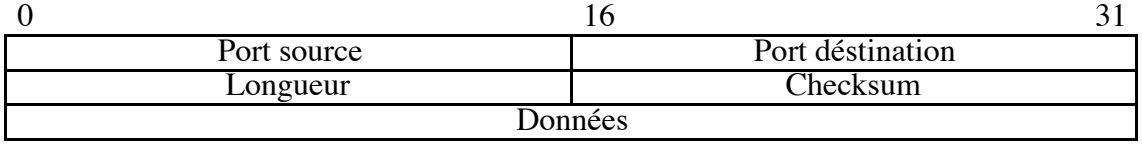

Fig 7 Entête UDP

Port source :
Le numéro du port source.

Port destination :
Le numéro du port destinataire.

Longueur :
longueur du datagramme.

Checksum :
Vérification de la validité du segment TCP.

3 - Le paquet TCP

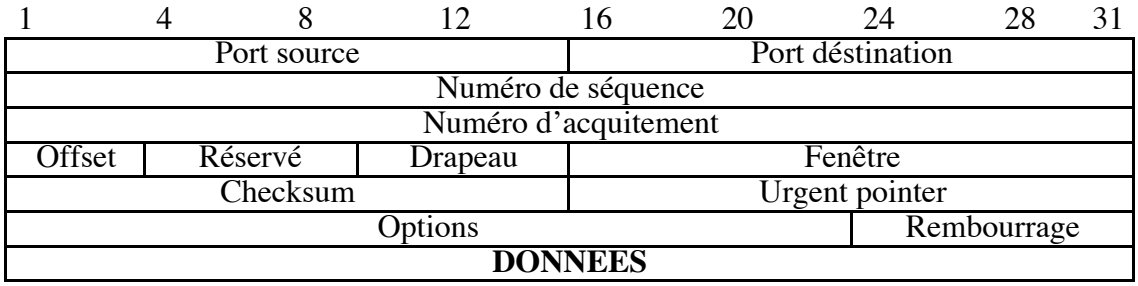

Fig 8 Entête TCP

Port source :
Le numéro du port source.

Port destination :
Le numéro du port destinataire.

Numéro de séquence :
Numéro de séquence du premier octet de données contenu dans le présent segment sauf dans le cas où on a une ouverture de connexion. Dans ce cas là, le champ présente la valeur ISN (Initial Séquence Number) et le premier octet de données a pour valeur ISN+1.

Numéro d'acquittement :
Ce champ contient le numéro de séquence du prochain octet que l'émetteur du présent segment est prêt à recevoir. une fois que la connexion est établie , ce champ est toujours envoyé.

Offset :
Ce champ donne le nombre de mots de 32 bits dans l'entête TCP. Il indique donc le début de la zone données.

Réservé :
Champ de 6 bits réservé à un usage futur

Drapeau ou bits de contrôle :
Bit 1 : Indication de la présence de données urgentes.
Bit 2 : Validation du champ acquittement.
Bit 3 : Fonction Push (concerne la transmission élémentaire de données)
Bit 4 : Réinitialisation de la connexion (Reset)
Bit 5 : Synchronisation des numéros de séquence initiaux à l'établissement de la connexion.

Fenêtre :
Nombre d'octets que l'émetteur du présent segment est prêt à recevoir. En concordance avec le champ Numéro d'acquittement, cela indique les numéros de séquence acceptables pour les prochains octets à recevoir (ils devront être distincts et compris entre le numéro d'acquittement et le numéro d'acquittement + la fenêtre.

Checksum :

Vérification de la validité du segment TCP.

Urgent pointer :

Ce champ fonctionne en concordance avec le bit 1 des drapeaux. Lorsqu'il y a des données urgentes ce pointeur indique le numéro de séquence du premier octet qui suit les données urgentes.

Options :

Ce champ a une taille variable. Il n'est pas obligatoire et peut ne pas apparaître dans le datagramme.

Rembourrage :

Utilisé en cas d'option pour amener si besoin est la longueur de l'entête internet à un multiple de 32 bits.

2ème PARTIE

LES SOCKETS

I - LE MODELE CLIENT SERVEUR

1- définitions

Les protocoles de communication comme TCP/IP permettent la communication point à point entre deux applications s'éxécutant eventuellement sur deux machines différentes.

Le détail du transfert effectif des données entre deux applications est spécifié par le protocole de communication de la couche transport, mais le moment et la façon dont les applications interagissent entre elles sont laissés à la charge du programmeur.

L'architcture client/serveur est devenue la méthode incontournable pour la communication point à point au niveau applicatif, quand le protocole utilisé est de la famille TCP/IP.

Ce modéle est motivé par le fait que TCP/IP ne fournit aucun mécanisme permettant l'exécution automatique d'un programme à l'arrivée d'un message si bien que dans une communication point à point, l'une des applications doit attendre l'initiative de la communication de la part de l'autre application.

Les applications peuvent être classées en deux catégories:

- les **clients**: applications qui prennent l'initiative du lancement de la communication, c'est à dire demande l'ouverture de connexion, l'envoi d'une requête, l'attente de la réponse à la requête, reprise de l'exécution du programme.
- les **serveurs**: applications qui attendent la communication, c'est à dire l'attente d'une demande d'ouverture de connexion, la reception d'une requête et l'envoi d'une réponse.

2- applications orientées connexion ou sans connexion

Une application est dite orientée connexion si le protocole sous-jacent utilisé est en mode connecté (TCP/IP par exemple).

Une application est dite orientée sans connexion si le protocole sous-jacent utilisé est en mode non connecté (UDP/IP par exemple).

L'avantage de l'utilisation d'un protocole comme TCP/IP est la fiabilité: la couche transport effectue elle même le "cheksum", la réémission de morceaux de messages perdus, l'élimination des morceaux duppliqués, l'adaptation du débit.

Un protocole comme UDP/IP n'effectue pas ces vérifications qui doivent alors être faites par le protocole de communication de niveau applicatif. Pour cette raison, la programmation de clients ou serveurs en mode non connecté est plus complexe qu'en mode connecté. Cependant, en réseau local où le transport est fiable, il peut être avantageux d'utiliser UDP car ce mode de communication demande moins d'opérations que TCP.

II - LES SOCKETS

INTRODUTION

LES SOCKETS RACONTEES AUX ENFANTS

Le concept de socket est assez difficile à appréhender. C'est BSD qui les a choisis pour accomplir les communications inter-processus (IPC). Cela veut dire qu'un socket est utilisé pour permettre aux processus de communiquer entre eux de la même manière que le téléphone nous permet de communiquer entre nous. L'analogie entre le concept de socket et le téléphone est assez proche, et sera utilisée pour éclaircir la notion de socket.

Pour recevoir des coups de téléphone, vous devez d'abord installer le téléphone chez vous. De la même façon vous devez commencer par créer un socket avant d'attendre des demandes de communications. La commande socket() est alors utilisée pour créer un nouveau socket.

Seulement il faut créer un socket avec les bonnes options. Vous devez spécifier le type d'adressage du socket. Les deux types d adressage les plus répandus sont AF_UNIX (famille d'adresse UNIX) et AF_INET (famille d'adresse Internet). AF_INET utilise les adresses Internet qui sont du format suivant xx.xx.xx.xx (ex: 178.33.174.34). En plus des adresses Internet, on a besoin aussi d'un numéro de port sur la machine pour pouvoir utiliser plusieurs socket simultanément.

Une autre option a spécifier lors de la création d'un socket est son type. les deux types les plus répandus sont SOCK_STREAM et SOCK_DGRAM. SOCK_STREAM sont spécifiques au mode connecté alors que SOCK_DGRAM sont spécifiques au mode déconnecté.

De la même façon qu'on vous attribue un numéro de téléphone pour recevoir vos appels, on doit spécifier au socket une adresse à laquelle il doit recevoir les messages qui lui sont destinés. Ceci est réalisé par la fonction bind() qui associe un numéro au socket.

Les sockets de type SOCK_STREAM ont la possibilité de mettre les requêtes de communication dans une file d'attente , de la même façon que vous pouvez recevoir un appel pendant une conversation téléphonique. C'est la fonction listen() qui permet de définir la capacité de la file d'attente (jusqu'à 5) . Il n est pas indispensable d'utiliser cette fonction mais c'est plutôt une bonne habitude que de ne pas l'oublier.

L'étape suivante est d'attendre les demandes de communication. C'est le rôle de la fonction accept(). Cette fonction retourne un nouveau socket qui est connecté à l appelant. Le socket initial peut alors se remettre à attendre les demandes de communication. C'est pour cette raison qu'on exécute un fork à chaque demande de connexion.

On sait maintenant comment créer un socket qui reçoit des demandes de communication, mais comment l'appeler ?

Pour le téléphone vous devez d'abord avoir le numéro avant de pouvoir appeler. Le rôle de la fonction connect() est de connecter un socket a un autre socket qui est en attente de demande de communication. Maintenant qu'une connexion est établie entre deux sockets, la conversation peut commencer. C'est le rôle des fonction read() et write(). A la fin de la communication on doit raccrocher le téléphone ou fermer le

socket qui a servi à la communication. C'est le rôle de la fonction close().

Nous attendons avec impatience le transfert d'appel avec un socket. La partie suivante se propose d'approfondir les points survolés par cette brève introduction sur les sockets.

Sous UNIX l'implémentation est de la forme suivante:

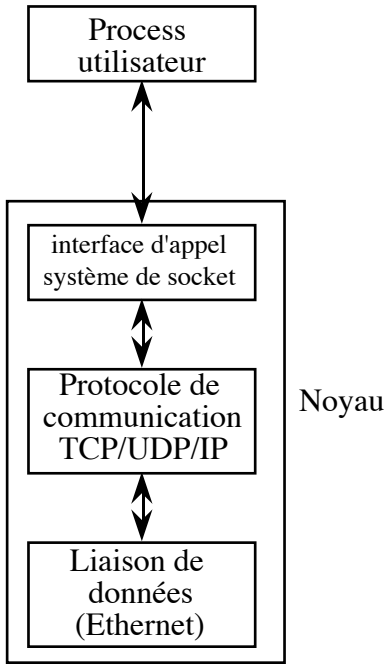

Fig 9 Implémentation du réseau sous UNIX

III - ADRESSES DE SOCKETS

1- les familles d'adresse

Il existe plusieurs familles d'adresses, chacune correspondant à un protocole particulier. Les familles les plus répandues sont:

AF_UNIX	Protocoles internes de UNIX
AF_INET	Protocoles Internet
AF_NS	Protocols de Xerox NS
AF_IMPLINK	Famille spéciale pour des applications particulières auxquelles nous ne nous intéresserons pas.

2- les structures d'adresse

Plusieurs appels systèmes réseaux sous unix nécessitent un pointeur sur une structure d'adresse de socket. La définition de cette structure se trouve dans <sys/socket.h> ;

```
struct sockaddr {
        u_short         sa_family ;
```

```
        char            sa_data[14] ;
} ;
```

sa_family : adresse de famille : prend la valeur AF_xxx
sa_data : peut contenir jusqu'à 14 octets de protocole spécifique d'adresse interprété selon le type d'adresse.

Pour la famille internet les structures suivantes sont définis dans le fichier <netinet/in.h>.

```
struct in_addr {
        u_long          s_addr ;
} ;
```

s_addr : 32 bits constituants l'identificateur du réseau et de la machine hôte ordonnes selon l'ordre réseau.

```
struct sockaddr_in {
        short           sin_family ;
        u_short         sin_port ;
        struct  in_addr  sin_addr ;
        char            sin_zero[8] ;
} ;
```

sin_family : AF_INET ;
sin_port : 16 bits de numéro de port (ordonnancement réseau) ;
sin_addr : 32 bits constituants l'identificateur du réseau et de la machine hôte ordonnes selon l'ordre réseau.
sin_zero[8] : inutilisés ;

Le fichier <sys/types.h> fourni des définitions de C et de types de données qui sont utilisés dans le système. Ainsi nous verrons apparaître les définitions suivantes qui sont malheureusement différentes entre les version 4.3BSD et le système V.

TYPE en C	**4.3BSD**	**Système 5**
unsigned char	u_char	unchar
unsigned short	u_short	ushort
unsigned int	u_int	uint
unsigned long	u_long	ulong

VI - LES APPELS SYSTEME

1- L'appel système socket

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

int socket (int family, int type, int protocole) ;

La variable family peut prendre 4 valeurs dont les préfixes commencent par AF comme Famille d'Adresse :

AF_UNIX	Protocoles internes de UNIX
AF_INET	Protocoles Internet
AF_NS	Protocols de Xerox NS
AF_IMPLINK	Famille spéciale pour des applications particulières auxquelles nous ne nous intéresserons pas.

La variable type peut prendre 5 valeurs :

SOCK_STREAM	utilisé en mode connecté au dessus de TCP.
SOCK_DGRAM	utilisé en mode déconnecté avec des datagrammes au dessus de UDP.
SOCK_RAW	utilisé au dessus de IP
SOCK_SEQPACKET	Sequenced Packet socket. Ce type n'est pas utilisable avec TCP/IP ni avec UDP/IP.

Le mode raw accessible par l'option SOCK_RAW lors de la création d'un socket, permet l'accès aux interfaces internes du réseau. Ce mode n'est accessible qu'au super-utilisateur et ne sera pas détaillé.

L'argument protocole dans l'appel système socket est généralement mis à 0. Dans certaines applications spécifiques, il faut cependant spécifier la valeur du protocole. Les combinaisons valides pour la famille AF_INET sont les suivantes :

TYPE	PROTOCOLE	PROTOCOLE ACTUEL
SOCK_DGRAM	IPPROTO_UDP	UDP
SOCK_STREAM	IPPROTO_TCP	TCP
SOCK_RAW	IPPROTO_ICMP	ICMP
SOCK_RAW	IPPROTO_RAW	(raw)

Les constantes IPPROTO_xxx sont définies dans le fichier <netinet/in.h>.

L'appel système socket retourne un entier dont la fonction est similaire à celle d'un descripteur de fichier. Nous appellerons cette entier descripteur de sockets (sockfd).

2 - L'appel système bind

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

*int bind (int sockfd, struct sockaddr *myaddr , int addrlen) ;*

Le premier argument est le descripteur de socket retourné par l'appel socket.
Le second argument est un pointeur sur une adresse de protocole spécifique et le troisième est la taille de cette structure d'adresse. Il y a 3 utilisations possibles de bind :

1 - Le serveur enregistre sa propre adresse auprès du système. Il indique au système que tout message reçu pour cette adresse doit lui être fourni. Que la liaison soit avec ou sans connexion l'appel de bind est nécessaire avant l'acceptation d'une requête d'un client.

2 - Un client peut enregistrer une adresse spécifique pour lui même.

3 - Un client sans connexion doit s'assurer que le système lui a affecté une unique adresse que ses correspondants utiliseront afin de lui envoyer des messages.

L'appel de bind rempli l'adresse locale et celle du process associés au socket.

3 - L'appel système connect

Un socket est initialement créé dans l'état non connecté, ce qui signifie qu'il n'est associé à aucune destination éloignée. L'appel système connect associe de façon permanente un socket à une destination éloignée et le place dans l'état connecté.

Un programme d'application doit invoquer connect pour établir une connexion avant de pouvoir transférer les données via un socket de transfert fiable en mode connecté.

Les sockets utilisées avec les services de transfert en mode datagramme n'ont pas besoin d'établir une connexion avant d'être utilisés, mais procéder de la sorte interdit de transférer des données sans mentionner à chaque fois, l'adresse de destination.

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

*int connect (int sockfd, struct sockaddr *servaddr , int addrlen) ;*

sockfd est le descripteur de socket retourné par l'appel socket.
servaddr est un pointeur sur une structure d'adresse de socket qui indique l'adresse de destination avec laquelle le socket doit se connecter.
addrlen taille de la structure d'adresse.

4 - L'appel système listen

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

int listen (int sockfd, int backlog) ;

Cet appel est généralement utilisé après les appels socket et bind et juste avant les appels l'appel accept. L'argument backlog spécifie le nombre de connections à établir dans une file d'attente par le système lorsque le serveur exécute l'appel accept. Cet argument est généralement mis à 5 qui est la valeur maximale utilisée.

5 - L'appel système accept

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

*int accept (int sockfd, struct sockaddr *peer , int *addrlen) ;*

L'argument sockfd désigne le descripteur de socket sur lequel seront attendues les connexions. Peer est un pointeur vers une structure d'adresse de socket.

Lorsqu'une requête arrive, le système enregistre l'adresse du client dans la stucture *peer et la longueur de l'adresse dans *addrlen. Il crée alors un nouveau socket connecté avec la destination spécifiée par le client, et renvoit à l'appelant un descripteur de socket. Les échanges futurs avec ce client se feront donc par l'intermédiaire de ce socket.

Le socket initiale sockfd n'a donc pas de destination et reste ouvert pour accepter de futures demandes.

Tant qu'il n'y a pas de connexions le serveur se bloque sur cette appel. Lorsqu'une demande de connexion arrive, l'appel système accept se termine. Le serveur peut gérer les demandes itérativement ou simultanément :

Dans l'approche itérative, le serveur traite lui même la requête, ferme le nouveau socket puis invoque de nouveau accept pour obtenir la demande suivante.

Dans l'approche parallèle, lorsque l'appel système accept se termine le serveur crée un serveur fils chargé de traiter la demande (appel de fork et exec). Lorsque le fils a terminer il ferme le socket et meurt. Le serveur maître ferme quand à lui la copie du nouveau socket après avoir exécuté le fork. Il appel ensuite de nouveau accept pour obtenir la demande suivante.

Exemple de serveur itératif

```
int sockfd, newsockfd ;

if ( ( sockfd = socket (.....)) < 0 )
        err_sys("erreur de socket") ;
if ( bind ( sockfd, ....) < 0 )
        err_sys ("erreur de bind")
if ( listen ( sockfd , 5) < 0 ) ;
        err_sys (" erreur de listen" ) ;
```

```
for ( ; ; ) {
        newsockfd = accept ( sockfd, .....) ;
        if ( newsockfd < 0)
                err_sys( "erreur de accept") ;

        execute_la_demande( newsockfd ) ;
        close ( newsockfd ) ;
}
```

Exemple de serveur à accès concurrent

```
int sockfd, newsockfd ;

if ( ( sockfd = socket (.....)) < 0 )
        err_sys("erreur de socket") ;
if ( bind ( sockfd, ....) < 0 )
        err_sys ("erreur de bind")
if ( listen ( sockfd , 5) < 0 ) ;
        err_sys (" erreur de listen" ) ;

for ( ; ; ) {
        newsockfd = accept ( sockfd, .....) ;
        if ( newsockfd < 0)
                err_sys( "erreur de accept") ;

        if ( fork() == 0 ) {
                close ( sockfd ) ;
                execute_la_demande( newsockfd ) ;
                exit (1) ;
                }
        close ( newsockfd ) ;
}
```

V - ÉCHANGES D'INFORMATIONS SUR UN SOCKET

1- EMISSION D'INFORMATION

Une fois que le programme d'application dispose d'un socket, il peut l'utiliser afin de transférer des données. Cinq appels système sont utilisables : send, sendto, sendmsg, write et writev. Send, write et writev ne sont utilisables qu'avec des sockets en mode connecté car ils ne permettent pas d'indiquer d'adresse de destination. Les différences entre ces trois appels sont mineures :

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

write ( int sockfd, char *buff, int nbytes ) ;
writev ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int send (int sockfd, char *buff, int nbytes, int flags ) ;
```

socket	contient le descripteur de socket .
buff	est un pointeur sur un tampon où sont stockées les données à envoyer.
nbytes	est le nombre d'octets ou de caractères que l'on désire envoyer
vect_E/S	est un pointeur vers un tableau de pointeurs sur des blocs qui constituent

le message à envoyer.
flags drapeau de contrôle de la transmission :

Pour le mode non connecté on a deux appels sendto et sendmsg qui imposent d'indiquer l'adresse de destination :

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

int sendto (int sockfd, char *buff, int nbytes, int flags, struct sockaddr *to, int addrlen) ;
```

Les quatres premiers arguments sont les mêmes que pour send, les deux derniers sont l'adresse de destination et la taille de cette adresse.

Pour les cas où on utiliserait fréquemment l'appel sendto qui nécessite beaucoup d'arguments et qui serait donc d'une utilisation trop lourde on définit la structure suivante:

```
struct struct_mesg {
        int *sockaddr ;
        int   sockaddr_len ;
        iovec *vecteur_E/S
        int    vecteur_E/S_len
        int   *droit_d'acces
        int    droit_d'acces_len
 }
```

Cette structure sera utilisée par l'appel sendmsg :
int sendmsg (int sockfd, struct struct_mesg, int flags) ;

2 - RECEPTION D'INFORMATION

On distingue 5 appels système de réception d'information qui sont symétriques au appels d'envoi. Pour le mode connecté on a les appelles read, readv et recv et pour le mode sans connexion on a les appelles recvfrom et recvmsg.

```
int read ( int sockfd, char *buff, int nbytes ) ;
int readv ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int recv (int sockfd, char *buff, int nbytes, int flags ) ;
```

sockfd est le descripteur sur lequel les données seront lues .
buff est un pointeur sur un buffer où seront stockées les données lues
nbytes est le nombre maximal d'octets ou de caractères qui seront lus.

readv permet de mettre les données lues dans des cases mémoire non contiguës. Ces cases mémoires sont pointées par un tableau de pointeurs qui lui même est pointé par vect_E/S.
lgr_vect_E/S est la longueur de ce tableau.

Pour le mode sans connexion, il faut préciser les adresses des correspondants desquels on attend des données.

*int recvfrom (int sockfd, char *buff, int nbytes, int flags, struct sockaddr *from*
int addrlen) ;

Pour les mêmes raisons que pour sendto et sendmsg, et pour des raison de symétrie on a définie l'appel recvmsg qui utilise la même structure que sendmsg.

int recvmsg (int sockfd, struct struct_mesg, int flags) ;

DETAIL DES ARGUMENTS FLAGS

Les arguments flags sont soit zéro soit le résultat d'un or entre les trois constantes suivantes :

MSG_OOB	émission ou réception de données hors bande
MSG_PEEK	peek at incoming message (recv or recvfrom)
MSG_DONTROUTE	bypass routing (send or sendto)

The MSG_PEEK flag lets the caller look at the data that's available to be read, without having the système discard the data after the recv or recvfrom returns.

MSG_DONTROUTE sera detaillé dans les options de socket.

MSG_OOB : les données hors bande

Les données hors bande sont des données urgentes.

Les données hors bande sont des transmissions indépendantes entre une paire de socket connectés. Les données hors bande sont délivrées indépendamment des données normales. Pour les protocoles de communication qui ne supportent pas les données hors bande, les messages urgents sont extraits du flux de données et stockés séparément. Ceci laisse le choix à l'utilisateur entre recevoir les données urgentes dans l'ordre ou pas, sans avoir à les stocker dans un buffer.

Si un socket possède un process-group, un signal SIGURG est généré quand le protocole prend connaissance de l'arrivée d'une donnée hors bande. Le process-group ou le process id qui devra être informé de l'arrivée d'une donnée urgente peut être initialisé par la fonction fcntl().

si plusieurs sockets sont susceptibles de recevoir des données hors bande, l'appel de la fonction select() permet de sélectionner le socket devant recevoir la donnée urgente. Pour envoyer des données hors bande, l'option MSG_OOB doit être utilisée lors de l'appel de send() ou sendto().

LE MULTIPLEXAGE

L'appel système qui permet le multiplexage des sockets est select().

select() examine les descripteurs qui lui sont passés en paramètre et teste si certains sont prêts à lire, écrire ou ont une condition d'exception. Les données hors bande sont les seules conditions d'exception que peut recevoir un socket.
description de l'appel système select():

*int select (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)*

On ne s'attardera pas sur les structures passées en paramètre, seuls les fonctions

qui permettront leur manipulation seront détaillées. Les arguments 2, 3 et 4 de select sont des pointeurs sur des structures qui contiendront les descripteurs de sockets à tester en lecture, écriture et en réception d'exception par le select.

Le dernier argument timeout, s'il est non nul , spécifie l'intervalle de temps maximal à attendre avant de sortir de select. Si ce pointeur est nulle select() bloquera indéfiniment. Pour réaliser un polling sur select() (i.e. select ne rendra la main que lorsqu'au moins un descripteur aura été sélectionné) le pointeur timeout doit être non nul mais pointant sur une STRUCTURE timeval nulle.

A la fin de select les structures passées en arguments contiennent les descripteurs sélectionnés par la fonction select

Avant d'appeler select() , il faut initialiser ses arguments. On commencera toujours par les initialiser à une structure nulle par la fonction FD_ZERO (fd_set &fdset).

void FD_SET (int fd, fd_set &fdset) ajoute le descripteur fd à fdset.

void FD_CLR (int fd, fd_set &fdset) retire le descripteur fd a fdset.

A la fin de select, la fonction int FD_ISSET (int fd, fd_set &fdset) nous permet de savoir si le descripteur fd est sélectionné dans fdset ou non. cette fonction est à appeler après select().

VI- ENCHAINEMENT DES APPELS SYSTEMES

1- mode connecté

Deux approches de programmation de serveurs se présentent: l'approche itérative et l'approche parallèle.

Rappelons que dans l'approche itérative, le serveur traite lui même la requête, ferme le nouveau socket puis invoque de nouveau accept pour obtenir la demande suivante.

L'enchainement dans le temps des appels système pour un serveur itératif se résume par la figure suivante:

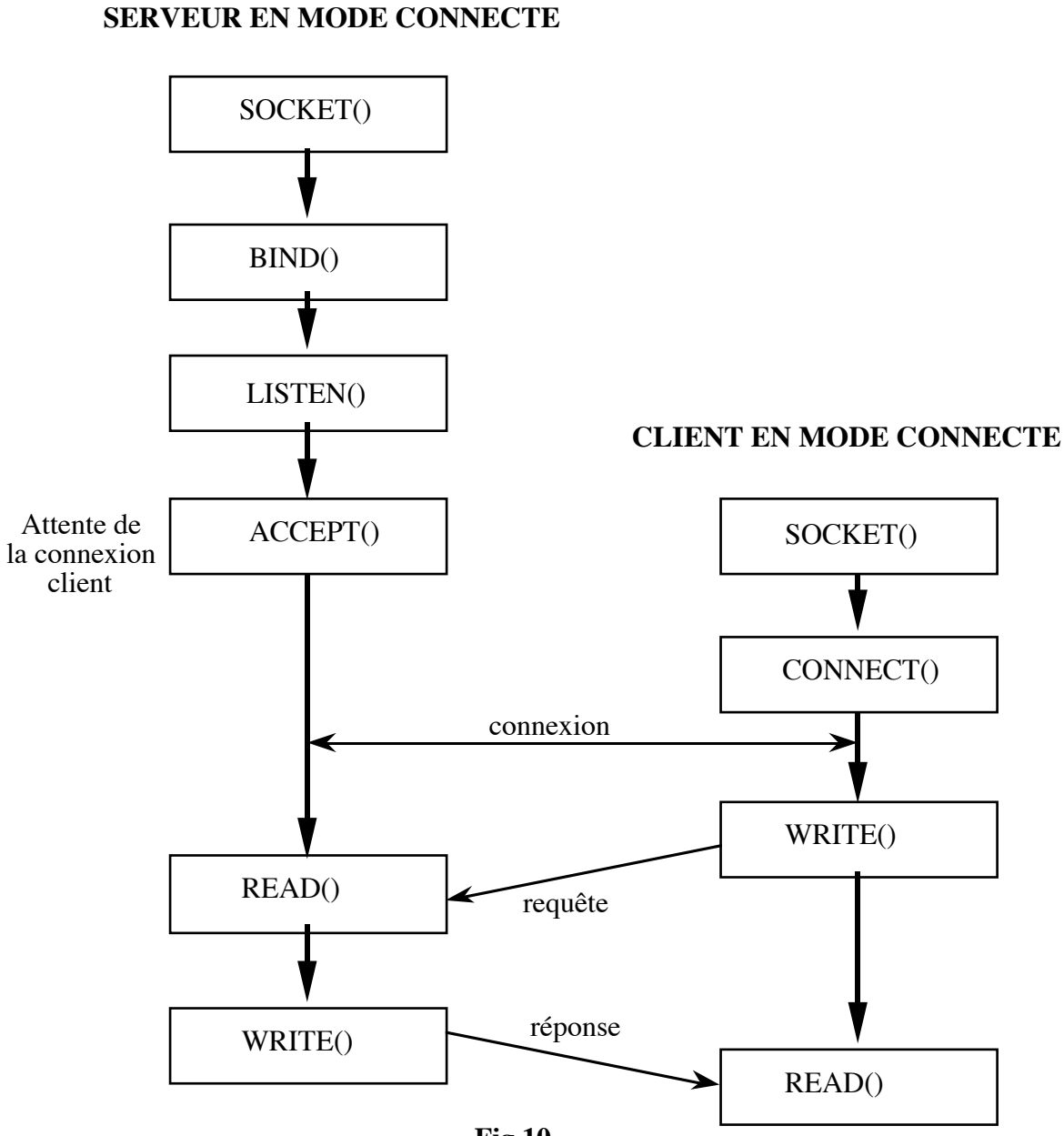

Fig 10

Dans l'approche parallèle, lorsque l'appel système accept se termine le serveur crée un serveur fils chargé de traiter la demande (appel de fork et exec). Lorsque le fils a terminer il ferme le socket et meurt. Le serveur maître ferme quand à lui la copie du nouveau socket après avoir exécuté le fork. Il appel ensuite de nouveau accept pour obtenir la demande suivante.

L'enchainement des appels systèmes pour un serveur parallèle se résume par la figure suivante:

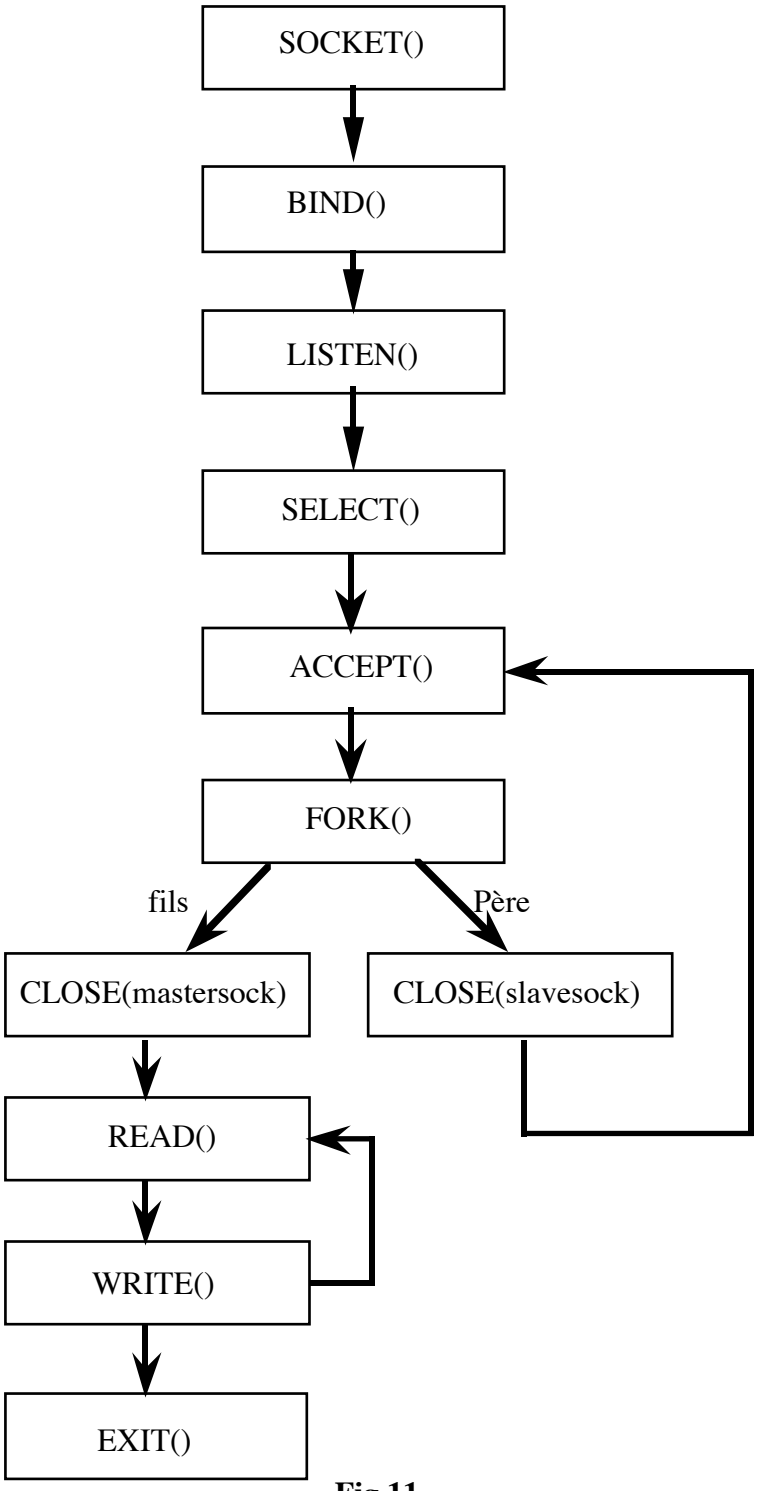

Fig 11

2- mode non connecté

Dans le cas d'un protocole sans connexion, les appels système sont différents. Le client n'établit pas de connexion avec le serveur. Le client envoit un datagramme au serveur en utilisant l'appel système sendto(). Symétriquement le serveur n'accepte pas de connexion d'un client, mais attend un datagramme d'un client avec l'appel système recvfrom(). Ce dernier revoie l'adresse réseau du client, avec le datagramme, pour que le serveur puisse répondre à la bonne adresse.

l'enchainement dans le temps des appels systèmes pour une communication en mode non connecté et pour un serveur itératif se résume par la figure suivante:

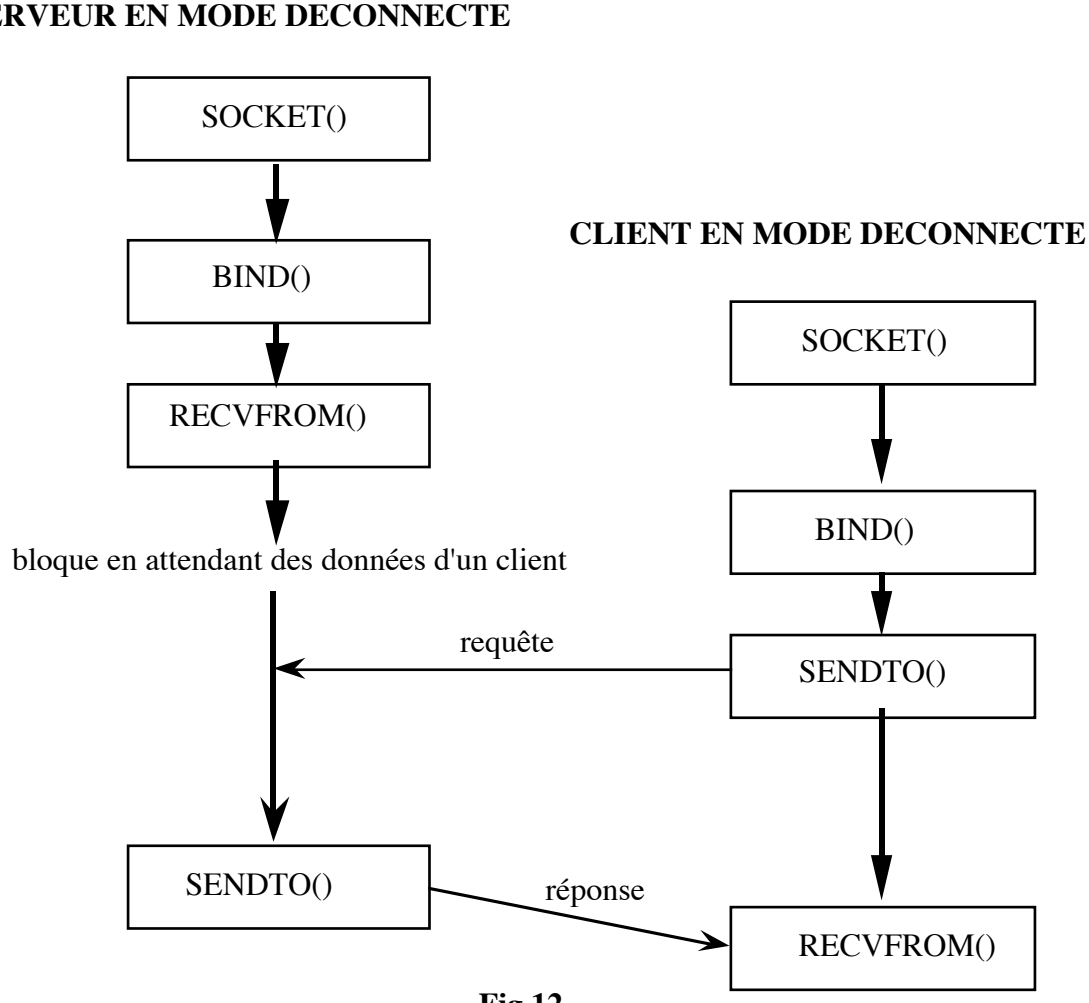

Fig 12

L'enchainement dans le temps des appels systèmes pour une communication en mode non connecté et pour un serveur parallèle se résume par la figure suivante:

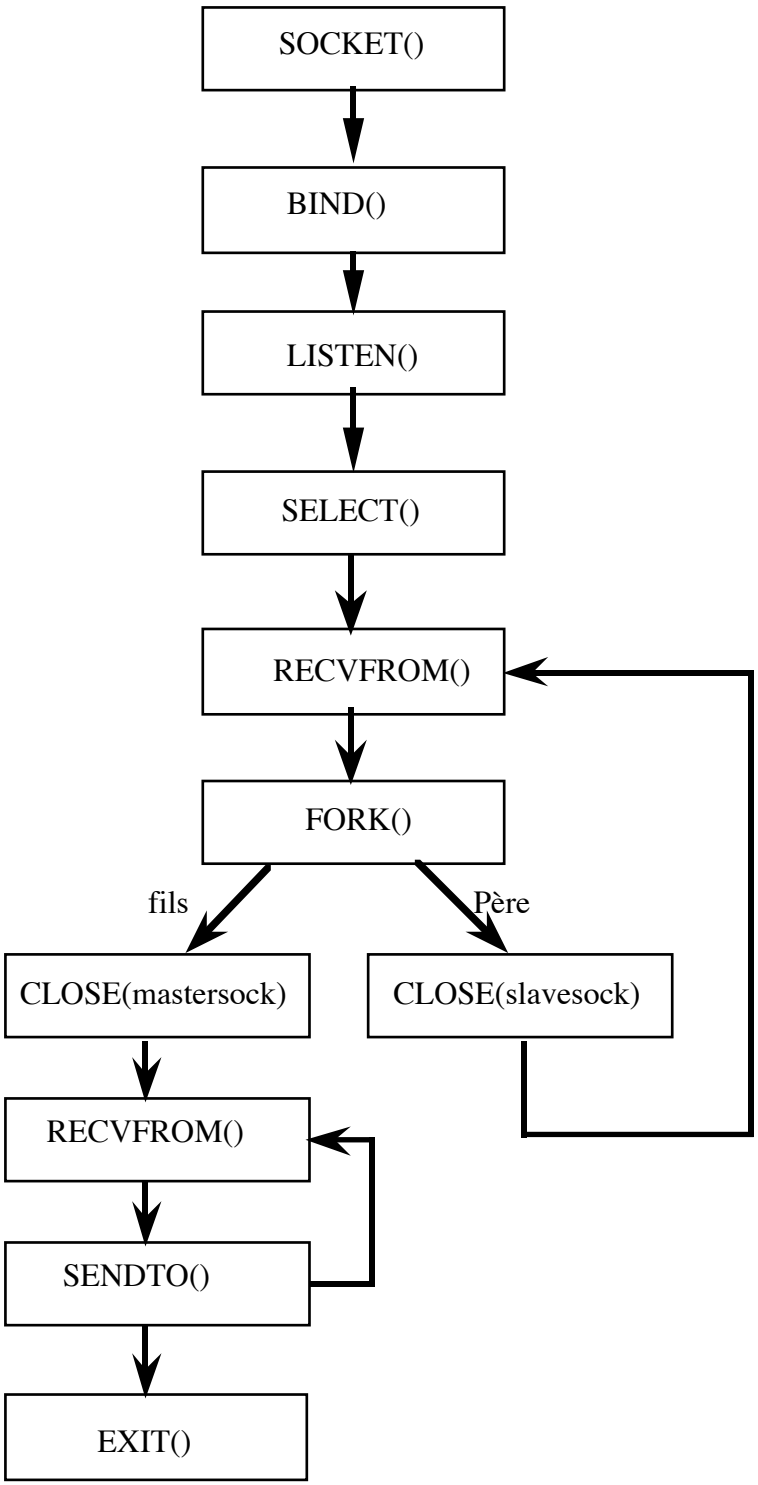

Fig 13

VII - DES PROCEDURES BIEN UTILE !

1- PROCEDURES RÉSEAU DE CONVERSION DE L'ORDRE DES OCTETS

Les machines ont des façons différentes d'enregistrer les entiers et les protocoles TCP/IP ont une représentation des octets normalisée et indépendante des machines.

Pour que les programmes soient portables sur toutes les machines il faut pouvoir convertir les représentations réseaux en représentation machine a chaque de paquet et vice et versa.

Les fonction de conversion sont les suivantes :

htonl (val)
host to network long : convertit une valeur sur 32 bits de la représentation machine vers la représentation réseau.

htons (val)
host to network short : convertit une valeur sur 16 bits de la représentation machine vers la représentation réseau.

ntohl (val)
network to host long : convertit une valeur sur 32 bits de la représentation réseau vers la représentation machine.

ntohs (val)
network to host short : convertit une valeur sur 16 bits de la représentation réseau vers la représentation machine.

2- LES OPERATIONS SUR LES OCTETS

Pour initialiser une structure où pour la mettre à jour on dispose de deux fonctions:

Tout d'abord la fonction bzero() qui initialise à zéro tous les champs de la structure passée en paramètre. Cette fonction doit être appelée avant l'utilisation de la structure.

Il y a aussi la fonction bcopy() qui a pour paramètre deux structures et un entier n qui indique le nombre d'octets à copier. Cette fonction copie les n premiers octets de la structure 1 dans la structure 2.

Les listings de ces deux fonctions sont donnés à la fin de cet ouvrage.

3 -DEMANDER ET DEFINIR LES NOMS DE MACHINE

Le système d'exploitation UNIX gère en interne un nom de machine. Pour les machines reliées à l'Internet , le nom interne coïncide généralement avec celui du domaine qui correspond avec l'interface réseau principale. L'appel système "rechercher le nom de la machine" (gethostname) permet aux processus utilisateurs d'accéder au nom de la machine. L'appel système "définir le nom de la machine" (sethostname) permet à des processus privilégiés de définir le nom de la machine.

Ces appels ont la forme suivante :

gethostname (nom, longueur) ;
sethostname (nom, longueur) ;

nom	est un pointeur sur une chaîne de caractères contenant le nom de la machine
longueur	est la taille du nom.

4- OBTENIR DES INFORMATIONS RELATIVES AUX MACHINES

Il existe des bibliothèques de procédures qui permettent de retrouver l'information relative à une machine à partir de son nom de domaine ou de son adresse IP. Lorsqu'elles sont utilisées sur des machines qui ont accès à un serveur de noms de domaines. Elles émettent une requête et attendent la réponse. Lorsqu'elles sont utilisées sur des machines qui n'ont pas accès à un serveur de nom, les machines obtiennent les informations à partir d'une base de données stockée sur mémoire secondaire.

La fonction gethostbyname (rechercher une machine par son nom) accept un nom de domaine et renvoie un pointeur vers une structure qui contient l'information relative à cette machine. L'appel est le suivant :

*ptr = gethostbyname (char *chaîne_nom) ;*

la structure où gethostbyname met les information est la suivante :

```
struct hostent {
        char        *h_name ;        /*nom officiel de la machine*/
        char        *h_aliases ;     /*liste des surnoms de cette machine*/
        int         h_addrtype ;     /*type d'adresse (exple adresse IP) */
        int         h_length ;       /*longueur de l'adresse */
        char        **h_addr_list ;  /*liste des adresses de la machine */
                                     /*une machine peut avoir plusieurs
adresses                                    comme c'est le cas pour les routeurs
*/
 }

#define h_addr a_addr_list[0] ;
```

La fonction gethostbyaddr (rechercher une machine par son adresse) produit le même résultat . La différence entre les deux est que cette fonction a pour argument l'adresse de la machine.

*ptr = gethostbyaddr (char *addr,int lgr,int type) ;*

addr	pointe vers l'adresse de la machine.
lgr	longueur de l'adresse.
type	indique le type d'adresse (adresse IP par exemple).

5- OBTENIR DES INFORMATIONS RELATIVES AUX RÉSEAUX

De la même façon que pour l'accès aux informations sur les machines à travers le réseau ou une base de données, on peut accéder aux informations sur le réseau par les procédures "rechercher-nom-du réseau-par-nom" (getnetbyname). Cette procédure prend comme paramètre un nom de réseau et rempli une structure dont elle retourne le pointeur.

*ptr = getnetbyname (char *nom) ;*

la structure remplie est la suivante :

```
struct netent {
        char        *n_name ;       /*nom officiel de réseau *:
        char        **n_aliases ;   /*surnom du réseau */
        int         n_addrtype ;    /*type d'adresse */
        int         n_net   ;       /*adresse du réseau sur 32 bits */
                                    /*une adresse IP par exple où le numéro de
                                    machine sera zéro */
 } ;
```

La fonction getnetbyaddr (rechercher-nom-de réseau-par-adresse) produit le même résultat. La différence entre les deux est que cette fonction a pour argument l'adresse de la machine.

ptr = getnetbyaddr (addr_net, type) ;

addr_net est une adresse sur 32 bits.
type indique le type d'adresse.

6 - OBTENIR DES INFORMATIONS RELATIVES AUX PROTOCOLES

Dans la base de données des protocoles disponibles sur la machine, chaque protocole a un nom officiel, des alias (surnoms) officiels et un numéro de protocole officiel. La procédure getprotobyname permet d'obtenir des informations sur le protocole en donnant son nom.

*ptr = getprotobyname(char *proto_name) ;*

la fonction renvoie un pointeur sur la structure d'information suivante :

```
struct protoent {
        char        *p_name ;       /*nom officiel du protocole*/
        char        **p_aliases ;   /*surnoms du protocoles*/
        int         p_proto ;       /*numéro du protocole*/
 } ;
```

La fonction getprotobynumber produit le même résultat mais fournie comme argument le numéro du protocole.

ptr = getprotobynumber (int num_proto) ;

7- OBTENIR DES INFORMATIONS RELATIVES AUX SERVICES RÉSEAUX

Certains numéros de ports sont réservés pour TCP comme pour UDP, ainsi par exemple le port 43 correspond au service whois (qui_est_ce) qui permet de donner des informations sur les utilisateurs d'une machine serveur.
la procédure getservbyname permet d'obtenir des informations sur le service en donnant comme argument une chaîne de caractères contenant le numéro du port, et une chaîne contenant le protocole utilisé.

*ptr = getservbyname (char *num_port, char *proto) ;*

ptr pointe sur la structure suivante :

```
struct servent {
        char        *s_name ;        /*nom du service */
        char        **s_aliases ;    /*surnoms du service*/
        int         s_port ;         /*numéro du port */
        char        *s_proto ;       /*nom du protocole*/
}
```

VIII- EXEMPLE DE PROGRAMME DE CLIENT/SERVEUR

L'exemple que nous allons développer concerne une application de transfert de fichier. Le client après l'établissement de la connexion, fournit au serveur un nom de fichier (qui correspondra dans le cas de cette application à un signe astrologique) et le serveur envoi au client le contenu de son fichier (qui sera donc l'horoscope de la semaine).

1 - exemple de client

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

/* programme client */

main(argc,argv)
int argc ;
char *argv[] ;

{
int sd,lon ;
struct sockaddr_in adsock;
struct hostent *hptr ;          /* infos recuperes sur le host */
struct servent *sptr ;          /* infos recuperes sur le service */
char buf[256] ;                 /*buffer de lecture des infos */
char *prog ;                    /*nom de ce programme */
char *host ;                    /*pointeur sur le nom du host distant */
char *mesg ;
char *resg ;                    /*pointeur sur l'utilisateur distant */

prog = argv[0] ;

/* verification du nombre d'arguments de la ligne de commande */
if (argc != 3) {
        printf("il faut trois arguments\n") ;
        exit(1) ;
        }

host = argv[1] ;
mesg = argv[2] ;

/* prendre les infos sur le host */
if((hptr = gethostbyname(host)) == NULL) {
        printf("probleme d'infos sur le host \n") ;
        exit(1) ;
        }

/* initialiser la struct adsock avec les infos recuperes */
/* c a d l'adresse du host et le domaine internet */
bcopy((char *)hptr->h_addr,(char*)&adsock.sin_addr,hptr->h_length) ;
adsock.sin_family = hptr->h_addrtype ;
```

```
/* prendre les infos sur le service */
if((sptr = getservbyname("2223","tcp")) == NULL ) {
        printf("probleme d'infos sur le service \n") ;
        exit(1) ;
        }

/* recupere le numero du port */
adsock.sin_port = sptr->s_port ;

/* creer maintenant un socket */
if((sd = socket(AF_INET,SOCK_STREAM,0))<0) {
        printf("probleme lors de la creation de socket \n") ;
        exit(1) ;
        }

/* etablir la connexion avec le serveur */
if((connect(sd,( struct sockaddr * ) &adsock,sizeof(adsock))) < 0 ) {
        printf("probleme de connexion \n") ;
        exit(1) ;
        }

/* on va envoyer la requete */
if((write(sd, mesg, strlen(mesg))) < 0 ) {
        printf("erreur sur le write \n") ;
        exit(1) ;
        }

/* lecture de la reponse du serveur */
while((lon =read(sd, buf, sizeof(buf))) > 0)
        /* affichage de la reponse sur l'ecran */
        write(1, buf, lon) ;

/* on ferme le socket */
close(sd) ;
exit(0) ;
}
```

2 - Exemple de serveur

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define MAXNOM 32

void renvoi(int s) ;
/* programme serveur */

main(argc,argv)
int argc ;
char *argv[] ;

{
int sd,lacc, nsd ;
struct sockaddr_in adsock, adacc; /* struc adresse internet */
struct hostent *hptr ;                  /* infos recuperes sur le host */
struct servent *sptr ;                          /* infos recuperes sur le service */
char machine[MAXNOM+1] ;          /* nom de la machine locale */
char *prog ;                                    /*nom de ce programme */
char *mesg ;

prog = argv[0] ;

/* prendre les infos sur le service */
if((sptr = getservbyname("2223","tcp")) == NULL ) {
          printf("probleme : service inconnu\n") ;
          exit(1) ;
          }

/* recuperer le nom de la machine locale */
gethostname(machine,MAXNOM) ;

/* prendre les infos sur la machine locale */
if((hptr = gethostbyname(machine)) == NULL) {
          printf("probleme : host inconnu\n") ;
          exit(1) ;
          }
bzero((char*)&adsock, sizeof(adsock));

/*initialiser la struct adsock avec les infos recuperes */
bcopy((char *)&hptr->h_addr,(char*)&adsock.sin_addr,hptr->h_length) ;

adsock.sin_family = AF_INET ;


/* recupere le numero du port */
adsock.sin_port = sptr->s_port ;

adsock.sin_addr.s_addr = INADDR_ANY ;
```

```
/* creer maintenant un socket */
if((sd = socket(AF_INET,SOCK_STREAM,0))<0) {
        printf("probleme lors de la creation de socket \n") ;
        exit(1) ;
        }

/* on va fair un bind sur le socket */
if(bind(sd,(struct sockaddr *) &adsock ,sizeof(adsock) ) == -1 ) {
        printf("probleme avec le bind \n") ;
        exit(1) ;
        }

/*initialiser la queue d'ecoute */
listen(sd,5) ; /* backlog = 5 */

/* bouble d'attente de nouvelles connexions */
for(;;){
        lacc = sizeof(adacc) ;
        /* boucle d'accept */
        if((nsd = accept(sd, ( struct sockaddr * ) &adacc, &lacc)) < 0 )
                {
                /* adacc -> infos du client */
                printf("erreur sur l'accept \n") ;
                exit(1) ;
                }
renvoi(nsd) ;
close(nsd);
}
}

/* fonction renvoi */
void renvoi(s)
int s ;
{
char buf[256];
char *fichier;
char f[20];
char *mesgptr;
char mesg[250];
int l, c;
FILE *fp ;
char line[250] ={0} ;
char *lineptr = NULL;

/* lire une requete */
if((l=read(s, buf, sizeof(buf))) <= 0)
        return ;
buf[l] = '\0' ;         /*fin de chaine */

fichier=f;
if (!strcmp(buf,"lion"))  fichier = "lion" ;
if (!strcmp(buf,"cancer")) fichier = "cancer" ;
if (!strcmp(buf,"scorpion")) fichier = "scorpion" ;
if (!strcmp(buf,"taureau")) fichier = "taureau" ;
```

```
if (!strcmp(buf,"vierge")) fichier = "vierge" ;
if (!strcmp(buf,"poisson")) fichier = "poisson" ;
if (!strcmp(buf,"capricorne"))  fichier = "capricorne" ;
if (!strcmp(buf,"balance"))  fichier = "balance" ;
if (!strcmp(buf,"belier"))  fichier = "belier" ;
if (!strcmp(buf,"verseau"))  fichier = "verseau" ;
if (!strcmp(buf,"gemeaux"))  fichier = "gemeaux" ;


if((fp=fopen(fichier,"r"))==NULL) {
        printf("erreur lors de l'ouverture du fichier\n");}

while(fgets(line,240,fp)){
                write(s, line, strlen(line)) ;
                }
return ;

}
```

IX - LES APPELS SYSTEME SOCKET AVANCES

1- L'appel système getpeername

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

*int getpeername (int sockfd, struct sockaddr *peer, int *addrlen) ;*

Cette appel système rend le "nom" du processus correspondant auquel le socket est connecté. Le nom rendu est en fait l'adresse de la machine correspondante et le processus correspondant.

2- L'appel système getsockname

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

*int getsockname (int sockfd, struct sockaddr *peer, int *addrlen) ;*

Cette appel rend le nom associé au socket c'est à dire l'adresse de la machine locale et le processus local.

3- L'appel système shutdown

La façon normale de clore une session de connexion réseau est de faire l'appel système close. Cette appel a la propriété au moment de son appel de tenter de délivrer toutes les donner qui sont encore à envoyer. L'appel shutdown fourni plus de moyens de contrôle de la connexion au moment de sa fermeture.

int shutdown (int sockfd, int howto) ;

Si howto = 0 aucune donnée ne peut être reçue sur le socket.
Si howto = 1 aucune donnée ne peut être envoyée sur le socket.
Si howto = 2 aucune donnée ne peut être reçue ou envoyée sur le socket.

4 - Les ports réservés

Il y a deux façons pour un process d'avoir un numéro de port :

• Le process demande un numéro de port spécifique. Ce qui est le cas pour les serveurs qui ont besoin d'un numéro de port bien connu.
• Le process peut aussi laisser le système lui allouer un numéro de port automatiquement. En pratique, cela se fait en faisant un appel bind avec comme argument de port zéro.

Pour TCP comme pour UDP dans le domaine Internet, il y a des ports réservés. Ces derniers vont de 1 à 1023. Le fichier etc/services contient les numéros de ports réservés et les services associés. Voici une partie de ce fichier:

```
#
# Network services, Internet style
#
echo            7/tcp
echo            7/udp
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat                 15/tcp
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
name            42/udp          nameserver
whois           43/tcp          nicname
domain          53/udp
domain          53/tcp
hostnames       101/tcp         hostname

# Ajout Serge.Rouveyrol@imag.fr pour Wais
z3950           210/tcp         wais            # Wide Area Information Server
...
...
...
```

L'instruction rresvport () fournie à son appelant un socket TCP réservé.

```
int rresvport ( int *aport ) ;
```

Cette fonction crée un socket stream Internet et fait un bind avec un port réservé. Elle retourne le descripteur de sockets ou -1 s'il y a eu un problème. L'argument aport est un pointeur sur un entier qui est le premier port à partir duquel le bind sera fait.

	Internet
ports réservés	1-1023
ports automatiquement alloués par le système	1024-5000
ports alloués par l'instruction rresvport()	512-1023

résumé des allocations de port

X - DEMANDER DES OPTIONS DE SOCKET

En plus de l'association d'un socket à une adresse locale ou de sa connexion a une adresse de destination, apparaît la nécessité d'un mécanisme qui permette aux programmes d'application de piloter les sockets.

Ainsi lors de l'utilisation de protocoles utilisants temporisateurs et retransmission, les programmes d'applications peuvent souhaiter connaître ou définir les valeurs de temporisateur. Il peuvent ainsi contrôler l'allocation de la mémoire tampon, vérifier si le socket autorise la diffusion ou la gestion des données hors bande. Plutôt que d'ajouter des primitives pour chaque opération de contrôle, les concepteurs ont choisi de construire un mécanisme unique. Ce dernier ne comporte que deux opérations : getsockopt et setsockopt.

L'appel système getsockopt permet à l'application de demander les informations relatives au socket. L'appelant indique le socket, les options intéressantes et un tampon où enregistrer les informations demandées. Le système d'exploitation analyse ses structures de données internes relatives au socket et transmet l'information à l'appelant. L'appel est le suivant :

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

*int getsockopt (int sockfd, int level, int optname, char *optval, int *optlen) ;*

sockfd désigne le socket sur lequel les informations sont demandées.
level indique si l'opération s'applique au socket lui même ou à un protocole sous-jacent (TCP IP ...)
optname indique l'option unique à laquelle s'applique l'opération
les deux derniers argument sont respectivement le tampon où seront mis les informations, et la taille des informations mises dans le tampon.

L'appel setsockopt permet à l'application de définir les options de socket :

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

*int setsockopt (int sockfd, int level, int optname, char *optval, int *optlen) ;*

Les définitions des arguments sont identiques à ceux de getsockopt.
Le tableau suivant résume les options de socket

level	optname	get	set	Description	flag	type de données
IPPROTO_IP	IP_OPTIONS	•	•	option de l'entête IP		
IPPROTO_TCP	TCP_MAXSEG	•		donne la taille max dun segment tcp	•	int
	TCP_NODELAY	•	•	ne pas retarder l'envoie pour unir des paquets		int
SOL_SOCKET	SO_DEBUG	•	•	permet des infos de debuggage	•	int
	SO_DONTROUTE	•	•	utilise juste les adresses d'interface	•	int
	SO_ERROR	•		rend le status de l'erreur	•	int
	SO_LINGER	•	•	contrôle de l'envoie des données après close		struct linger
	SO_OOBINLINE	•	•	concerne la réception de données hors dande	•	int
	SO_RCVBUF	•	•	taille du buffer de réception		int
	SO_SNDBUF	•	•	taille du buffer d'envoi		int
	SO_RCVTIMEO	•	•	timeout de réception		int
	SO_SNDTIMEO	•	•	timeout d'emission		int
	SO_REUSEADDR	•	•	autorise la réutilisabilité de l'adresse locale	•	int
	SO_TYPE	•		fournit le type de socket		int

Les options des sockets

IP_OPTIONS : permet au programmeur de spécifier des options dans l'entête des trames IP. Ce qui nécessite une bonne connaissance de l'entête IP

TCP_MAXSEG : Retourne la taille maximum de segment utilisé pour le socket. La valeur typique est 1024 octets. Notons que cette valeure affectée par le système ne peut être modifiée par le programmeur.

TCP_NODELAY : Cette option est utilisée dans les réseaux lents où le système bufferise les commandes à envoyer en attendant la réponse du serveur des requêtes précédentes pour afficher l'echo.
Lorsque le clients se déroule sur un terminal à fenêtre l'echo n'est pas nécessaire et l'option TCP_NODELAY court-circuite l'algorithme de mise dans le buffer pour que les données soient transmises le plus tôt possible c'est à dire aussi tôt que le réseau le permet.

SO_DEBUG : Autorise ou pas de debugage au niveau bas dans le noyau. Cette option autorise le noyau a maintenir un historique des récents paquets envoyés ou reçus.

SO_DONTROUTE : Spécifie que les messages sortant doivent passer par les mécanismes de routage normaux des couches basses au lieu d'être dirigés vers l'interface réseau appropriée comme spécifié dans la portion réseau de l'adresse de destination.

SO_ERROR : Retourne à l'appelant le contenu de la variable so_error, qui est définie dans <sys/socketvar.h>. Cette variable contient les valeurs standards des numéros d'erreurs UNIX.

SO_LINGER : Cette option détermine ce qu'il faut faire quand il existe des messages non envoyés pour un socket quand le processus a exécuté close sur le socket. Par défaut, close s'exécute immédiatement et le système tente de livrer les données non envoyées.
Si l'option linger est activée l'action dépend de la variable linger time spécifiée par l'utilisateur : si elle est égal à zéro, toute
données à envoyer après le close ne sera pas transmise. Sinon elle sera transmise.

SO_OOBINLINE : Cette option spécifie que les données hors band doivent aussi être placer dans la queue d'entrée. Quand les données hors bande sont réception n'est pas nécessaire pour lire les données hors bande.

SO_RCVBUF et SO_SNDBUF : Spécifie la taille du buffer de réception et d'emission pour le socket. Cette option n'est nécessaire que lorsque l'on désir une taille de buffer supérieure à celle que l'on a par défaut.

SO_RCVTIMEO et SNDTIMEO : Ces deux options spécifie les valeur de timeout pour la et l'envoie de messages. Cette option est rarement utilisée dans les applications.

SO_REUSEADDR : Informe le système pour qu'il autorise l'adresse locale à être réutilisée. Normalement le système n'autorise pas la réutilisation
de l'adresse locale; quand connect est appelé il requiert que l'adresse locale soit unique.

SO_TYPE : retourne le type de socket. La valeure entière retournée est une valeure comme SOCK_STREAM ou SOCK_DGRAM. Cette option est typiquement appelée par un process qui hérite un
socket quant il commence à s'exécuter.

TROISIEME PARTIE

LES PROGRAMMES DEVELOPPES

L'application développée consiste en un jeu où le client essaye de deviner un mot caché par le serveur. Le client propose successivement des lettres et le serveur lui répond en lui signalant l'existence ou non de la lettre dans le mot caché. Le client a un nombre d'essais limité pour deviner ce mot.

Programme Client.c

```
#include <stdio.h>

/* machine, service et protocole par defaut */
#define DEFHOST "localhost"
#define DEFSERVICE "daytime"
#define DEFPROTOCOL "tcp"


/*************************************************************/
/*                                                           */
/* Ce programme appelle la fonction setsock avec les arguments passes  */
/* par l'utilisateur.                                        */
/*                                                           */
/* Exemlple d'utilisation : client ensisun 1345 udp                    */
/* Les arguments a passer sont:                              */
/*                                                           */
/*      1) la machine distante sur laquelle s'excecute le serveur.     */
/*         par defaut c'est la machine locale.                         */
/*                                                           */
/*      2) le port avec lequel on peut communiquer avec le serveur.    */
/*         par defaut c'est le port correspondant au service daytime.  */
/*                                                           */
/*      3) le protocole de communication ( TCP ou UDP ).               */
/*         par defaut c'est le protocole TCP.                          */
/*                                                           */
/*                                                           */
/*      AUTEURS:                                             */
/*              JAZOULI     Abdelillah                       */
/*              RADI        Nour-eddine                      */
/*              ZGHAL       Tarek                            */
/*                                                           */
/*      DATE:       JUIN 1994                                */
/*                                                           */
/*                                                           */
/*************************************************************/


int main(argc,argv)

int argc;       /* nombre d arguments passes par l'utilisateur      */
char *argv[];   /* pointeur sur les arguments                       */
                /* argv[0] contient le nombre d'arguments           */

{

/* nom ou numero de la machine distante supportant le serveur */
char *host = DEFHOST;
```

```
/* numero de port ou nom d'un service connu */
char *service = DEFSERVICE;

/* nom du protocole utilise TCP ou UDP */
char *protocol = DEFPROTOCOL;

/* descripteur du socket */
int sock;

/* ce switch initialise les varibles host, port et protocole    */
/* suivant le nombre d'arguments passes par l'utilisateur    */

switch (argc)
        {
         case 1:        /* 0 arguments passes en parametre */
                break;
         case 2:        /* 1 argument passe en parametre   */
                host = argv[1];
                break;
        case 3:         /* 2 arguments passes en parametre */
                host = argv[1];
                service = argv[2];
                break;
        case 4:         /* 3 arguments passes en parametre */
                host = argv[1];
                service = argv[2];
                protocol = argv[3];
                break;
        default:        /* probleme dans le nombre d'arguments */
                printf("erreur: usage:connection[serveur-port-protocol]\n");
                exit(1);
}

/*****************************************************************/
/*                                                               */
/* appel de la fonction setsock qui va allouer un socket au client.  */
/* la valeur rendue par setsock est le descripteur de socket alloue.
        */
/*                                                               */
/*****************************************************************/

sock = setsock(host, service, protocol);

/*****************************************************************/
/*                                                               */
/* on appelle la fonction de communication correspondant
        */
/* au protocole choisi                                           */
/*                                                               */
/*****************************************************************/

if ( strcmp(protocol,"tcp") == 0 )
        dialogueTCP(sock);
else if ( strcmp(protocol,"udp") == 0 )
        dialogueUDP(sock);
```

```
else
        printf("protocole inconnu\n");
/* on ferme le socket qui nous a ete alloue */
close(sock);
}
```

Programme setsock.c

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

#ifndef INADDR_NONE
#define INADDR_NONE     0xffffffff
# define DEFFLAG 0
#endif


/************************************************************/
/*                                                          */
/* Ce programme alloue un socket correspondant aux arguments passes  */
/* par le programme appelant.                               */
/*                                                          */
/* Les arguments a passer sont:                             */
/*                                                          */
/*      1) la machine distante sur laquelle s'excecute le serveur.    */
/*                                                          */
/*      2) le port avec lequel on peut communiquer avec le serveur.   */
/*                                                          */
/*      3) le protocole de communication ( TCP ou UDP ).    */
/*                                                          */
/*                                                          */
/*      AUTEURS:                                            */
/*              JAZOULI Abdelillah                                  */
/*              RADI  Nour-eddine                           */
/*              ZGHAL         Tarek                         */
/*                                                          */
/*      DATE:           JUIN 1994                           */
/*                                                          */
/*                                                          */
/************************************************************/



int setsock(host,service,protocole)

/* nom ou numero de la machine distante supportant le serveur */
char *host ;

/* numero de port ou nom d'un service connu */
char *service ;

/* nom du protocole utilise TCP ou UDP */
char *protocole ;

{
struct hostent *phe;   /* pointeur sur la structure d info du host      */
struct servent *pse;   /* pointeur sur la structure d info du service   */
struct protoent *ppe;  /* pointeur sur la structure d info du protocole */
```

```
struct sockaddr_in server_in; /* structure d info sur l adresse du serveur  */
int sock;               /* descripteur de socket                          */
int sock_type;          /* type associe au socket: STREAM ou DATAGRAM */

char bufreq[1];         /* ce message permet de debloquer le serveur      */
                        /* en mode deconnecte (DATAGRAM).                 */


/*************************************************************/
/*                                                           */
/* mise en place de la structure d information sur le serveur: server_in    */
/*                                                           */
/*************************************************************/

/* initialise la structure server_in a zero */
bzero ( (char *)&server_in, sizeof(server_in));

/* mise en place de la famille d'adresse */
server_in.sin_family = AF_INET;

/* mise en place du numero de port du service */
if ( pse = getservbyname(service,protocole) )
        {
        /* cas ou le service est designe par son nom */
        server_in.sin_port = htons(u_short)pse->s_port;
        }
        /* cas ou le service est designe par un numero de port */
else if  ( (server_in.sin_port = htons((u_short)atoi(service)) ) == 0 )
        {
          /* cas d'erreur sur le service */
          printf ("service %s impossible \n",service);
          exit(1);
        }

/* mise en place du numero du host */
if ( phe = gethostbyname(host) )
        /* cas ou le host est designe par un nom (ex:ensisun) */
        bcopy ( phe->h_addr, (char *)&server_in.sin_addr , phe->h_length);
        /* cas ou le host est designe par un numero (ex:xx.xx.xx.xx) */
else if ((server_in.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE)
        {
        /* cas d'erreur sur le host */
        printf ("entree de host %s impossible \n",host);
        exit(1);
        }



/*************************************************************/
/*                                                           */
/* mise en place de la structure d information sur le protocole             */
/*                                                           */
/*************************************************************/

if ( (ppe = getprotobyname(protocole)) == 0)
```

```
        {
        printf ("entree de protcole %s impossible \n",protocole);
        exit(1);
        }


/*****************************************************************/
/*                                                               */
/*                      allocation de socket                     */
/*                                                               */
/*****************************************************************/

/* mise en place du type de socket */
if ( strcmp(protocole,"udp") == 0 )
        sock_type = SOCK_DGRAM;
else
        sock_type = SOCK_STREAM;

/* creation du socket avec le protocole choisi */
sock = socket(PF_INET, sock_type, ppe->p_proto);
if ( sock < 0 )
        {
        printf ("impossible de creer le socket \n");
        exit(1);
}

/* CAS D'UN STREAM SOCKET */
/* connecter le socket au serveur */
if ( sock_type == SOCK_STREAM ){
  if ( connect(sock,(struct sockaddr *)&server_in,sizeof(struct sockaddr)) < 0 )
        {
        printf ("connection a %s,%s impossible \n",host,service);
        exit(1);
        }
}

/* CAS D'UN DATAGRAM SOCKET */
/* envoyer un message pour communiquer l'adresse du client au serveur */
if ( sock_type == SOCK_DGRAM )
      sendto(sock,bufreq,strlen(bufreq),DEFFLAG,(struct
sockaddr*)&server_in,sizeof(server_in)) ;

/* on renvoie le descripteur du socket alloue */
return(sock);

}               /* fin de setsock */
```

DialogueTCP

```
#include <stdio.h>
#define DEFLINELEN 128

/*****************************************************************/
/*                                                               */
/* Ce programme represente l'application du cote client.         */
/*                                                               */
/* L'application developpee consiste en un jeu ou le client essaye de  */
/* deviner un mot cache par le serveur .                         */
/* Le client propose successivement des lettres et le serveur lui */
/* repond en lui signalant l'existance ou nonde la lettre dans   */
/* le mot cache. Le client a un nombre d'essais limite par le serveur */
/* pour deviner le mot cache.                                    */
/*                                                               */
/* Le serveur envoi la valeur 1 pour signaler qu'il va envoyer   */
/* son dernier message. C'est le cas ou on a trouve le mot cache */
/* ou on a depasse le nombre d'essais autorise.                  */
/*                                                               */
/* Cette application est developpe pour un protocole TCP
        */
/*                                                               */
/* L'argument a passer est:                                      */
/*                                                               */
/*      1) Le descripteur de socket avec lequel on va communiquer */
/*         avec le serveur.                                      */
/*                                                               */
/*                                                               */
/*      AUTEURS:                                                 */
/*              JAZOULI Abdelillah
        */
/*              RADI  Nour-eddine                                */
/*              ZGHAL         Tarek                              */
/*                                                               */
/*      DATE:         JUIN 1994                                  */
/*                                                               */
/*                                                               */
/*****************************************************************/


void dialogueTCP(sock)

int sock;
{
/* buffer qui contiendra la requete du client */
char bufreq[DEFLINELEN+1];

/* buffer qui contiendra la reponse du serveur */
char bufans[DEFLINELEN+1];


int n ;
int PAS_TROUVE = 1;       /* vaut 1 si le client n'a pas encore trouve */
```

```
                                /* le mot cache, 0 s'il a trouve              */

/* on lit le message de bienvenue envoye par le serveur */
if ( (n = read(sock,bufans,DEFLINELEN) ) < 0 )
        printf("erreur sur le read \n");
else {
        /* on affiche ce message s'il n'ya pas eu d'erreur sur le read*/
        bufans[n]='\0' ;
        fputs(bufans,stdout) ;
        fflush(stdout);
}

/* on boucle indefiniment jusqu'a ce que l'on trouve le mot */
while (PAS_TROUVE) {

        /* on vide les buffer de requete et de reponse */
        n=0;
        bzero(bufans,sizeof(bufans));
        bzero(bufreq,sizeof(bufreq));

        /* on propose une lettre au serveur */
        printf ("proposer une lettre: ");
        fflush(stdin);
        scanf ("%c",bufreq );

        /* on envoi la lettre proposee au serveur */
        write (sock,bufreq,strlen(bufreq));

        printf ("reponse du serveur: ");

        /* on attend la reponse du serveur */
        if ( (n=read(sock,bufans,DEFLINELEN)) < 0 )
                printf("erreur sur le read \n");
        else {  /* cas ou on a recu la reponse du serveur */

                if (bufans[0]=='1') {
                /* cas ou le serveur nous signale qu'il */
                /* va envoyer son dernier message   */
                        fflush(stdout);
                        /* on lit le dernier message */
                        n=read(sock,bufans,DEFLINELEN) ;
                        /* on va arreter la communication */
                        PAS_TROUVE = 0 ;
                }
                bufans[n]='\0' ;
                /* on affiche le message du serveur */
                fputs(bufans,stdout) ;
                fflush(stdout);
        }
        printf("\n");
}  /* fin du while */

return ;
}               /* fin de dialogueTCP */
```

Dialogue UDP

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


#define DEFLINELEN 128
#define DEFFLAG 0

/*****************************************************************/
/*                                                               */
/* Ce programme represente l'application du cote client.         */
/*                                                               */
/* L'application developpee consiste en un jeu ou le client doit */
/* deviner un mot que le serveur lui a cache.                    */
/* Le client propose successivement des lettres et le serveur lui */
/* repond en lui signalant l'existance ou pas de la lettre dans
        */
/* le mot cache. Le client a un nombre d'essais determine par le serveur */
/* pour deviner le mot cache.                                    */
/*                                                               */
/* Le serveur envoi la valeur 1 pour signaler qu'il va envoyer   */
/* son dernier message. C'est le cas ou on a trouve le mot cache */
/* ou on a depasse le nombre d'essais autorise.                  */
/*                                                               */
/* Cette application est developpe pour un protocole UDP         */
/*                                                               */
/* L'argument a passer est:                                      */
/*                                                               */
/*     1) Le descripteur de socket avec lequel on va communiquer */
/*        avec le serveur.                                       */
/*                                                               */
/*                                                               */
/*     AUTEURS:                                                  */
/*           JAZOULI Abdelillah                                  */
/*           RADI    Nour-eddine                                 */
/*           ZGHAL   Tarek                                       */
/*                                                               */
/*     DATE:   JUIN 1994                                         */
/*                                                               */
/*                                                               */
/*****************************************************************/


void dialogueUDP(sock)

int sock;
{
/* buffer qui contiendra la requete du client */
char bufreq[DEFLINELEN+1];
```

```
/* buffer qui contiendra la reponse du serveur */
char bufans[DEFLINELEN+1];

char *bravo = "vous avez trouve le mot !" ;
char *TROUVE= "1";
struct sockaddr_in serv_fils;
int serv_addr_len ;
int n;
int PAS_TROUVE = 1;     /* vaut 1 si le client n'a pas encore trouve */
                                /* le mot cache, 0 s'il a trouve             */


serv_addr_len = sizeof(serv_fils) ;


/* on lit le message de bienvenue envoye par le serveur */
if ( (n=recvfrom(sock,bufans,sizeof(bufans),0,(struct sockaddr
*)&serv_fils,&serv_addr_len)) <0 )
        printf("erreur sur le read \n");
else {
      /* on affiche ce message s'il n'ya pas eu d'erreur sur le read*/
        bufans[n]='\0' ;
        fputs(bufans,stdout) ;
        fflush(stdout);
}


/* on boucle indefiniment jusqu'a ce que l'on trouve le mot */
while (PAS_TROUVE) {

      /* on vide les buffer de requete et de reponse */
        n=0;
        bzero(bufans,sizeof(bufans));
        bzero(bufreq,sizeof(bufreq));

      /* on propose une lettre au serveur */
        printf ("proposer une lettre: ");
        fflush(stdin);
        scanf ("%c",bufreq );

     /* on envoi la lettre proposee au serveur */
        sendto(sock,bufreq,strlen(bufreq),DEFFLAG,(struct
sockaddr*)&serv_fils,sizeof(serv_fils)) ;

        printf ("reponse du serveur: ");

      /* on attend la reponse du serveur */
        if ((n=recvfrom(sock,bufans,sizeof(bufans),0,(struct sockaddr
*)&serv_fils,&serv_addr_len))<0)
                printf("erreur sur le read \n");
        else {  /* cas ou on a recu la reponse du serveur */

                if (bufans[0]=='1') {
                /* cas ou le serveur nous signale qu'il */
```

```
                /* va envoyer son dernier message       */
                        fflush(stdout);
                        /* on lit le dernier message */
                        n=recvfrom(sock,bufans,sizeof(bufans),0,(struct sockaddr
*)&serv_fils,&serv_addr_len);
                        /* on va arreter la communication */
                        PAS_TROUVE = 0 ;
                }
                bufans[n]='\0' ;
                /* on affiche le message du serveur */
                fputs(bufans,stdout) ;
                fflush(stdout);
        }
        printf("\n");
}  /*fin du while */

return ;
} /* fin de dialogueUDP */
```

Serveur

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

/* service et protocole par defaut */
#define DEFSERVICE "1244"
#define DEFPROTOCOL "tcp"
/* nombre max de clients en attente */
#define DEFQUEUELEN 3

extern void serverUDP() ;
extern void serverTCP() ;

/*************************************************************/
/*                                                           */
/* Ce programme appelle la fonction setsock avec les arguments passes
        */
/* par l'utilisateur.                                        */
/*                                                           */
/* Les arguments a passer sont:                              */
/*                                                           */
/*      1) le port avec lequel on peut communiquer avec le serveur.  */
/*         par defaut c'est le port 1234.                    */
/*                                                           */
/*      2) le protocole de communication ( TCP ou UDP ).     */
/*         par defaut c'est le protocole TCP.                */
/*                                                           */
/*         3) le nombre maximum de clients en attente.       */
/*            par defaut cette valeur vaut 3                 */
/*                                                           */
/*      AUTEURS:                                             */
/*            JAZOULI Abdelillah                             */
/*            RADI    Nour-eddine                            */
/*            ZGHAL   Tarek                                  */
/*                                                           */
/*      DATE:   JUIN 1994                                    */
/*                                                           */
/*                                                           */
/*************************************************************/


int main(argc,argv)

int argc;       /* nombre d arguments passes par l'utilisateur  */
char *argv[];   /* pointeur sur les arguments                   */
                /* argv[0] contient le nombre d'arguments       */

{

/* numero de port ou nom d'un service connu */
char *service = DEFSERVICE;

/* nom du protocole utilise TCP ou UDP */
```

```
char *protocol = DEFPROTOCOL;

/* file d'attente des clients */
int queue_len = DEFQUEUELEN ;

/* descripteur de socket */
int sock ;


/* ce switch initialise les varibles port et protocole    */
/* suivant le nombre d'arguments passes par l'utilisateur      */

switch( argc)
        {
        case 1 :        /* 0 arguments passes en parametre */
                break ;
        case 2 :        /* 1 arguments passes en parametre */
                service = argv[1] ; break ;
        case 3 :        /* 2 arguments passes en parametre */
                service = argv[1] ;
                protocol = argv[2] ; break ;
        case 4 :        /* 3 arguments passes en parametre */
                service = argv[1] ;
                protocol = argv[2] ;
                queue_len = atoi( argv[3]); break ;
        default :       /* probleme dans le nombre d'arguments */
                printf("USAGE : service protocol queue_len\n");
                exit(1) ;
} /* fin switch */


/*****************************************************************/
/*                                                               */
/* appel de la fonction setsock qui va allouer un socket au client.  */
/* la valeur rendue par setsock est le descripteur de socket alloue. */
/*                                                               */
/*****************************************************************/

sock = setsock( service,protocol,queue_len) ;

/*****************************************************************/
/*                                                               */
/* on appelle la fonction de communication correspondant         */
/* au protocole choisi                                           */
/*                                                               */
/*****************************************************************/

if ( strcmp( protocol,"tcp") == 0 )
        serverTCP( sock) ;
else  if ( strcmp( protocol,"udp") == 0 )
        serverUDP( sock,protocol) ;
    else printf(" protocol inconnue \n") ;

/* on ferme le socket */
close(sock);
```

```
}
```

Setsock.c pour le seveur

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

#define DEFPORTBASE 0


/*****************************************************************/
/*                                                               */
/* Ce programme alloue un socket correspondant aux arguments passes */
/* par le programme appelant.                                    */
/*                                                               */
/* Les arguments a passer sont:                                  */
/*                                                               */
/*      1) le port avec lequel on peut communiquer avec le serveur. */
/*                                                               */
/*      2) le protocole de communication ( TCP ou UDP ).         */
/*                                                               */
/*        3) le nombre max de clients en attente sur le socket
          */
/*                                                               */
/*                                                               */
/*      AUTEURS:                                                 */
/*            JAZOULI Abdelillah                                 */
/*            RADI    Nour-eddine                                */
/*            ZGHAL   Tarek                                      */
/*                                                               */
/*      DATE:   JUIN 1994                                        */
/*                                                               */
/*                                                               */
/*****************************************************************/



int setsock( service,protocol,queue_len)

/* numero de port ou nom d'un service connu */
char *service ;

/* nom du protocole utilise TCP ou UDP */
char *protocol ;

/* nombre maximum de clients en attente sur le socket */
int queue_len ;

{
struct servent *pse;            /* pointeur sur la structure d info du service   */
struct protoent *ppe;           /* pointeur sur la structure d info du protocole */
struct sockaddr_in server_in; /* structure d info sur l adresse du serveur */
int sock;                       /* descripteur de socket                 */
```

```
int socket_type;                /* type associe au socket: STREAM ou DATAGRAM*/
u_short portbase = DEFPORTBASE ;


/*****************************************************************/
/*                                                               */
/* mise en place de la structure d information sur le serveur: server_in */
/*                                                               */
/*****************************************************************/

/* initialise la structure server_in a zero */
bzero( (char * ) &server_in,sizeof(server_in) ) ;

/* mise en place de la famille d'adresse */
server_in.sin_family = AF_INET ;
server_in.sin_addr.s_addr = INADDR_ANY ;

/* mise en place du numero de port du service */
if ( pse = getservbyname( service,protocol) )
        {
        /* cas ou le service est designe par son nom */
        server_in.sin_port = htons (ntohs( (u_short)pse->s_port)+portbase) ;
        }
        /* cas ou le service est designe par un numero de port */
else if  ( (server_in.sin_port = htons((u_short)atoi(service))) == 0 )
        {
          /* cas d'erreur sur le service */
          printf("Ce service n'existe pas : %s\n",service) ;
          exit(1) ;
        }


/*****************************************************************/
/*                                                               */
/* mise en place de la structure d information sur le protocole  */
/*                                                               */
/*****************************************************************/


if ( ( ppe = getprotobyname(protocol)) == 0 ){
        printf(" Protocole %s inconnu\n",protocol) ;
        exit(1) ;
}

/*****************************************************************/
/*                                                               */
/*                  allocation de socket                         */
/*                                                               */
/*****************************************************************/


/* mise en place du type de socket */
if ( strcmp( protocol,"udp") == 0 )
        socket_type = SOCK_DGRAM ;
else
```

```
        socket_type = SOCK_STREAM ;

/* creation du socket avec le protocole choisi */
sock = socket( PF_INET,socket_type,ppe->p_proto) ;

if( sock < 0 )
        {
        printf("IMPOSSIBLE DE CREER UN SOCKET :\n") ;
        exit(1) ;
}

/* lien avec le port */
if ( bind(sock,(struct sockaddr *)&server_in,sizeof(server_in)) <0 )
        {
        printf("IMPOSIBLE DE LIER LE PORT %s :\n",service) ;
        exit(1) ;
}

/* on met le socket en mode passif dans le cas */
/* d'un serveur fonctionnant en mode connecte  */
if ( socket_type == SOCK_STREAM ) {
        if ( listen(sock,queue_len) < 0 )
        {
        printf("IMPOSIBLE DE METTRE LE PORT %s EN ATTENTE DE
REQUETES :\n",service);
        exit(1) ;
        }
}

/* on renvoi le descripteur de socket alloue */
return (sock) ;

}       /* fin de setsock */
```

Serveur TCP itératif

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

extern int errno ;
extern char *sys_errlist[] ;
extern void dialogueTCP() ;


void serverTCP(dialsock)
int dialsock ;
{
int slavesock ;
int client_addr_len ;
struct sockaddr_in client_in ;
client_addr_len = sizeof( client_in) ;


 while(1){
   slavesock = accept(dialsock,(struct sockaddr *)&client_in,&client_addr_len);

   if ( slavesock < 0){
      printf(" OUVERTURE D'UN SOCKET ESCLAVE IMPOSSIBLE\n");
      exit(1) ;
    }
  dialogueTCP(slavesock);
close(slavesock) ;
 }

}
```

Serveur TCP parallèle

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

extern void dialogueTCP() ;

/*****************************************************************/
/*                                                               */
/* Ce programme met le serveur en attente de clients.            */
/* Des que le serveur recoit une requete d'un client, il se debloque
        */
/* et cre un fils qui comuniquera avec le client.                */
/* Le pere se remet ensuite en attente d'un autre client.        */
/*                                                               */
/* L'argument a passer est:                                      */
/*                                                               */
/*       1) Le descripteur de socket avec lequel on va communiquer */
/*                                                               */
/*                                                               */
/*     AUTEURS:                                                  */
/*           JAZOULI Abdelillah                                  */
/*           RADI    Nour-eddine                                 */
/*           ZGHAL   Tarek                                       */
/*                                                               */
/*     DATE:   JUIN 1994                                         */
/*                                                               */
/*                                                               */
/*****************************************************************/


void serverTCP(mastersock)

int mastersock ;
{
/* descripteur de socket du fils */
int slavesock ;

struct sockaddr_in client_in; /* structure d info sur l adresse du client */

int client_addr_len ;

client_addr_len = sizeof( client_in) ;

/* le pere n'attends pas la fin de ses fils */
signal(SIGCHLD,SIG_IGN);

/* le serveur boucle en attendant des clients */
while (1) {

        /* le serveur bloque sur accept en attendant un client */
```

```
        slavesock = accept(mastersock,(struct sockaddr *)&client_in,&client_addr_len);

        /* slavesock contient un descripteur de socket qui servira au fils */

        if ( slavesock < 0 )
                {       /* cas d'erreur */
                printf(" OUVERTURE D'UN SOCKET ESCLAVE IMPOSSIBLE\n");
                exit(1) ;
        }

        /* le serveur va creer un fils qui communiquera avec le client */
        switch( fork() )
                {
                        case 0: /* fils */

                                /* le fils ferme le descripteur du pere */
                                close(mastersock) ;

                                /* le fils va dialoguer */
                                dialogueTCP(slavesock) ;

                                /* le fils ferme son socket */
                                close(slavesock) ;

                                /* suicide du fils */
                                exit() ;

                        case -1:
                                /* le pere a rencontre un probleme */
                                printf(" impossible de creer un fils \n") ;
                                break ;

                        default: /* pere */
                                 /* le pere ferme le descripteur du fils */
                                 /* et se remet en attente de client      */
                                close(slavesock) ;
                                break ;
        }       /* fin du switch */

} /* fin while: remet le serveur en attente de client */

}
```

Serveur UDP parallèle

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

#define DEFBUFREQ 128

extern void dialogueUDP() ;

/*****************************************************************/
/*                                                               */
/* Ce programme met le serveur en attente de clients.
        */
/* Des que le serveur recoit une requete d'un client, il se debloque   */
/* et cre un fils qui comuniquera avec le client.                */
/* Le pere se remet ensuite en attente d'un autre client.        */
/*                                                               */
/* L'argument a passer est:                                      */
/*                                                               */
/*      1) Le descripteur de socket avec lequel on va communiquer */
/*                                                               */
/*                                                               */
/* Ce programme concerne les serveurs UDP                        */
/*                                                               */
/*      AUTEURS:                                                 */
/*            JAZOULI Abdelillah                                 */
/*            RADI    Nour-eddine                                */
/*            ZGHAL   Tarek                                      */
/*                                                               */
/*      DATE:   JUIN 1994                                        */
/*                                                               */
/*                                                               */
/*****************************************************************/


void serverUDP(dialsock,protocol)

int dialsock ;
char *protocol ;
{
/* descripteur de socket du fils */
int slavesock  ;

char bufreq[DEFBUFREQ+1] ;
struct sockaddr_in client ,name ;
struct protoent *ppe ;
int client_addr_len ;
client_addr_len = sizeof(client) ;


while(1) {
```

```
        /* On doit bloquer le pere s'il n'ya pas de client */

if (recvfrom(dialsock,bufreq,sizeof(bufreq),0,(struct sockaddr
*)&client,&client_addr_len)<0) {
        printf("erreur sur recvfrom\n");
        exit(1);
        }

  switch(fork() ) {
        case 0 :/* On lance un fils pour s'occuper du dialogue avec un client*/
                {
                close(dialsock) ; /* socket du pere  a fermer */


                if ( ( ppe = getprotobyname(protocol) ) == 0 ){
                  printf("CE PROTOCOLE N'EXISTE PAS : %s\n",protocol) ;
                  exit(1) ;
                }
                slavesock = socket(AF_INET,SOCK_DGRAM,ppe->p_proto) ;
                name.sin_family = AF_INET ;
                name.sin_addr.s_addr = INADDR_ANY ;
                name.sin_port = 0 ;
                bind(slavesock,( struct sockaddr * )&name ,sizeof(name) ) ;
                dialogueUDP(slavesock,client) ;
              close(slavesock) ;/*socket du fils a fermer en fin de dialogue */
                exit(1) ; /* la mort du fils est ineluctable ! */
                }
        case -1 : /* erreure sur le fork() */
                printf("IMPOSSIBLE DE CRER UN FILS \n") ;
                break ;
        default : /* Le pere */
                close(slavesock) ; /* socket du fils a fermer */
  } /* fin switch */
 } /* fin while */
}
```

Dialogue TCP serveur

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

#define DEFBUFREQ 128
#define DEFBUFANS 128



/*****************************************************************/
/*                                                               */
/* Ce programme represente l'application du cote serveur.
        */
/*                                                               */
/* L'application developpee consiste en un jeu ou le client doit */
/* deviner un mot que le serveur lui a cache.                    */
/* Le client propose successivement des lettres et le serveur lui */
/* repond en lui signalant l'existance ou pas de la lettre dans
        */
/* le mot cache. Le client a un nombre d'essais determine par le serveur */
/* pour deviner le mot cache.                                    */
/*                                                               */
/* Le serveur choisi de facon aleatoire un mot dans le fichier DICO */
/* et demande au client de le deviner.                           */
/*                                                               */
/* Le serveur envoi la valeur 1 pour signaler qu'il va envoyer   */
/* son dernier message. C'est le cas ou on a trouve le mot cache */
/* ou on a depasse le nombre d'essais autorise.                  */
/*                                                               */
/* Cette application est developpe pour un protocole TCP         */
/*                                                               */
/* L'argument a passer est:                                      */
/*                                                               */
/*     1) Le descripteur de socket avec lequel on va communiquer */
/*                                                               */
/*                                                               */
/*     AUTEURS:                                                  */
/*           JAZOULI Abdelillah                                  */
/*           RADI    Nour-eddine                                 */
/*           ZGHAL   Tarek                                       */
/*                                                               */
/*     DATE:   JUIN 1994                                         */
/*                                                               */
/*                                                               */
/*****************************************************************/


void dialogueTCP(sock)

int sock ;
```

```
{
/* buffer qui contiendra la requete du client */
char bufreq[DEFBUFREQ+1];

/* buffer qui contiendra la reponse du serveur */
char bufans[DEFBUFANS+1];

/* mot a devine */
char word1[DEFBUFANS+1];

/* mot intermadiaire devine par le client */
char word2[DEFBUFANS+1];

/* message de bienvenue */
char *MSG_BIENVENUE;

/* message a envoyer au client pour lui indiquer qu'on va arreter  */
/* de jouer: soit il a trouve le mot ou nombre d'essais est epuise */
char *TROUVE;

/* contient le nombre de mot dans le fichier dico */
int file_lg=0;

/* le mot cache sera l'entree aleatoire de rang n dans dico */
int n ;

int i;

/* nombre d'essais du client */
int NB_ESSAIS ;

/* descripteur de fichier pour dico */
FILE *fd ;

/* message de bienvenue */
MSG_BIENVENUE="\nBIENVENUE SUR LE SERVEUR GAMES\n\nvous avez 15
coups pour deviner le mot cache. bonne chance\n\n";

/* envoi du message de bienvenue */
write(sock,MSG_BIENVENUE,strlen(MSG_BIENVENUE)) ;

/* code de fin de communication */
TROUVE="1";

/*****************************************************************/
/*                                                               */
/* Choix aleatoire du mot a deviner par le client                */
/*                                                               */
/*****************************************************************/

        /* ouverture du fichier dico */
        if ( (fd=fopen("dico","r")) == NULL)
                printf ("impossible d'ouvrir le fichier dico\n");

        bzero(word1,sizeof(word1));
```

```
        /* calcul du nombre total de mots dans dico */
        while (fgets(word1,15,fd)!=NULL)
                file_lg++ ;


        /* repositionne le pointeur au debut de dico */
        rewind(fd);

        /* choix d une valeur aleatoire n et initialise */
        /* word1 a la Nieme entree de dico */
        srand(getpid());
        /* on s'assure que n n'est pas nul */
        while( (n=rand()%file_lg)==0 );
        /* on lit l'entree N */
        for (i=1; i<=n; i++)
                fgets(word1,15,fd);

        /* positonne la fin de chaine */
        word1[strlen(word1)-1]='\0';

        /* initialise word2 avec des . */
        for ( i=0; i<strlen(word1); i++ )
                word2[i]='.';

        /* positonne la fin de chaine */
        word2[strlen(word1)]='\0';


        printf("word1:%s\n",word1);


/*****************************************************************/
/*                                                               */
/* boucle principale de reception et de renvoi de reponse        */
/*                                                               */
/*****************************************************************/

for ( NB_ESSAIS=1; NB_ESSAIS<16; NB_ESSAIS++) {

        /* on initialse les buffers de requete et de reponse a 0 */
        bzero(bufreq,sizeof(bufreq) ) ;
        bzero(bufans,sizeof(bufans) ) ;

        /* on lit la lettre proposee par le client */
        if ( (n=read(sock,bufreq,DEFBUFREQ)) < 0)
                printf ("erreur sur le read \n");
        else {
                bufreq[n] = '\0';
        }

        /* compare la lettre recue avec les lettres composant word1 */
        for ( i=0; i<strlen(word1); i++) {
                /* on met la lettre dans word2 si elle existe dans word1 */
```

```
                if ( bufreq[0]==*(word1+i*sizeof(char)) )
                *(word2+i*sizeof(char))=*(word1+i*sizeof(char));
        }

        if (!strcmp(word2,word1))
                {       /* cas ou le client a devine le mot cache */
                        /* on lui envoi le code d'arret             */
                        write(sock,TROUVE,strlen(TROUVE)) ;
                        TROUVE= "BRAVO vous venez de trouver le mot ";
                        strcat(TROUVE,word1);
                        /* on lui envoi le message de felicitations */
                        write(sock,TROUVE,strlen(TROUVE)) ;
                        /* on ferme le socket */
                        fclose(fd);
                        return;
                }
        else
                {       /* word2 different de word1 */
                        if (NB_ESSAIS==15)
                                {       /* le client a epuise ses essais */
                                TROUVE= "1";
                                /* on lui envoi le code d'arret             */
                                write(sock,TROUVE,strlen(TROUVE)) ;
                                TROUVE= "DESOLE le mot a deviner etait ";
                                strcat(TROUVE,word1);
                                /* on lui envoi la solution */
                                write(sock,TROUVE,strlen(TROUVE)) ;
                                }
                        else
                                /* on lui envoi word2 */
                                write(sock,word2,strlen(word2)) ;
        }

} /* fin du for */

/* on ferme le fichier */
fclose(fd);

return ;

}
```

Dialogue serveur UDP

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

#define DEFBUFREQ 128
#define DEFBUFANS 128
#define DEFFLAG 0

/*****************************************************************/
/*                                                               */
/* Ce programme represente l'application du cote serveur.        */
/*                                                               */
/* L'application developpee consiste en un jeu ou le client doit */
/* deviner un mot que le serveur lui a cache.                    */
/* Le client propose successivement des lettres et le serveur lui */
/* repond en lui signalant l'existance ou pas de la lettre dans
        */
/* le mot cache. Le client a un nombre d'essais determine par le serveur */
/* pour deviner le mot cache.                                    */
/*                                                               */
/* Le serveur choisi de facon aleatoire un mot dans le fichier DICO */
/* et demande au client de le deviner.                           */
/*                                                               */
/* Le serveur envoi la valeur 1 pour signaler qu'il va envoyer   */
/* son dernier message. C'est le cas ou on a trouve le mot cache */
/* ou on a depasse le nombre d'essais autorise.                  */
/*                                                               */
/* Cette application est developpe pour un protocole UDP         */
/*                                                               */
/* L'argument a passer est:                                      */
/*                                                               */
/*     1) Le descripteur de socket avec lequel on va communiquer */
/*                                                               */
/*     2) La structure d'adresse du client                       */
/*                                                               */
/*                                                               */
/*     AUTEURS:                                                  */
/*           JAZOULI Abdelillah                                  */
/*           RADI    Nour-eddine                                 */
/*           ZGHAL   Tarek                                       */
/*                                                               */
/*     DATE:   JUIN 1994                                         */
/*                                                               */
/*                                                               */
/*****************************************************************/


void dialogueUDP(sock,client)

int sock ;
```

```
struct sockaddr client ;
{
/* buffer qui contiendra la requete du client */
char bufreq[DEFBUFREQ+1];

/* buffer qui contiendra la reponse du serveur */
char bufans[DEFBUFANS+1];

/* mot a devine */
char word1[DEFBUFANS+1];

/* mot intermadiaire devine par le client */
char word2[DEFBUFANS+1];

/* message de bienvenue */
char *MSG_BIENVENUE;

/* message a envoyer au client pour lui indiquer qu'on va arreter  */
/* de jouer: soit il a trouve le mot ou nombre d'essais est epuise */
char *TROUVE;

/* contient le nombre de mot dans le fichier dico */
int file_lg=0;

/* le mot cache sera l'entree aleatoire de rang n dans dico */
int n ;

int i;

/* nombre d'essais du client */
int NB_ESSAIS ;

/* descripteur de fichier pour dico */
FILE *fd ;

/* taille de la structure client */
int client_addr_len = sizeof(client) ;

/* message de bienvenue */
MSG_BIENVENUE="\nBIENVENUE SUR LE SERVEUR GAMES\n\nvous avez 15
coups pour deviner le mot cache. bonne chance\n\n";

/* envoi du message de bienvenue */
sendto(sock,MSG_BIENVENUE,strlen(MSG_BIENVENUE),DEFFLAG,(struct
sockaddr*)&client,sizeof(client)) ;

/* code de fin de communication */
TROUVE="1";


/*****************************************************************/
/*                                                               */
/* Choix aleatoire du mot a deviner par le client                */
/*                                                               */
/*****************************************************************/
```

```
        /* ouverture du fichier dico */
        if ( (fd=fopen("dico","r")) == NULL)
                printf ("impossible d'ouvrir le fichier dico\n");

        bzero(word1,sizeof(word1));

      /* calcul du nombre total de mots dans dico */
        while (fgets(word1,10,fd)!=NULL)
                file_lg++ ;

        /* repositionne le pointeur au debut de dico */
        rewind(fd);

      /* choix d une valeur aleatoire n et initialise */
        /* word1 a la Nieme entree de dico */
        srand(getpid());
        /* on s'assure que n n'est pas nul */
        while( (n=rand()%file_lg)==0 );
        /* on lit l'entree N */
        for (i=1; i<=n; i++)
                fgets(word1,15,fd);

        /* positonne la fin de chaine */
        word1[strlen(word1)-1]='\0';

        /* initialise word2 avec des . */
        for ( i=0; i<strlen(word1); i++ )
                word2[i]='.';

        /* positonne la fin de chaine */
        word2[strlen(word1)]='\0';

        printf("word1:%s\n",word1);


/*****************************************************************/
/*                                                               */
/* boucle principale de reception et de renvoi de reponse
        */
/*                                                               */
/*****************************************************************/

for ( NB_ESSAIS=1; NB_ESSAIS<16; NB_ESSAIS++) {

      /* on initialse les buffers de requete et de reponse a 0 */
        bzero(bufreq,sizeof(bufreq) ) ;
        bzero(bufans,sizeof(bufans) ) ;

        /* on lit la lettre proposee par le client */
        if ( ( n=recvfrom(sock,bufreq,sizeof(bufreq), 0, (struct sockaddr *)&client,
&client_addr_len) ) < 0)
                printf ("erreur sur le read \n");
        else {
                bufreq[n] = '\0';
```

```
        }

      /* compare la lettre recue avec les lettres composant word1 */
        for ( i=0; i<strlen(word1); i++) {
                /* on met la lettre dans word2 si elle existe dans word1 */
                if ( bufreq[0]==*(word1+i*sizeof(char)) )
                        *(word2+i*sizeof(char))=*(word1+i*sizeof(char));
        }

      if (!strcmp(word2,word1))
                {       /* cas ou le client a devine le mot cache */
                        /* on lui envoi le code d'arret          */
                        sendto(sock,TROUVE,strlen(TROUVE),DEFFLAG,(struct
sockaddr *)&client,sizeof(client)) ;
                        TROUVE= "BRAVO vous venez de trouver le mot ";
                        strcat(TROUVE,word1);
                        /* on envoi au client un message de felictations */
                        sendto(sock,TROUVE,strlen(TROUVE),DEFFLAG,(struct
sockaddr *)&client,sizeof(client)) ;
                        /* on ferme le socket */
                        fclose(fd);
                        return;
                }
        else
                {       /* word2 different de word1 */
                        if (NB_ESSAIS==15)
                                {       /* le client a epuise ses essais */
                                TROUVE= "1";
                                /* on lui envoi le code d'arret */

        sendto(sock,TROUVE,strlen(TROUVE),DEFFLAG,(struct sockaddr
*)&client,sizeof(client)) ;
                                TROUVE= "DESOLE le mot a deviner etait ";
                                strcat(TROUVE,word1);
                                /* on lui envoi la solution */

        sendto(sock,TROUVE,strlen(TROUVE),DEFFLAG,(struct sockaddr
*)&client,sizeof(client)) ;
                                }
                        else
                                /* on lui envoi word 2*/
                                sendto(sock,word2,strlen(word2),DEFFLAG,(struct
sockaddr *)&client,sizeof(client)) ;
        }

} /* fin du for */

/* on ferme le fichier */
fclose(fd);

return ;

}
```

Dans nos programmes on a utilisé les fonctions bzero() et bcopy() qui n'éxistent pas sur toutes les versions UNIX. Nous avons donc joint un listing de ces deux fonctions.

La fonction *bzero()*

```
/*

NAME

        bzero -- zero the contents of a specified memory region

SYNOPSIS

        void bzero (char *to, int count)

DESCRIPTION

        Zero COUNT bytes of memory pointed to by TO.

*/

void bzero (to, count)
  char *to;
  int count;
{
  while (count-- > 0)
    {
      *to++ = 0;
    }
}
```

La fonction *bcopy()*

```
/* bcopy -- copy memory regions of arbitary length
   Copyright (C) 1991 Free Software Foundation, Inc.

NAME

        bcopy -- copy memory regions of arbitrary length

SYNOPSIS

        void bcopy (char *in, char *out, int length)

DESCRIPTION

        Copy LENGTH bytes from memory region pointed to by IN to memory
        region pointed to by OUT.

*/
```

```
void bcopy (src, dest, len)
  register char *src, *dest;
  int len;
{
  if (dest < src)
    while (len--)
      *dest++ = *src++;
  else
    {
      char *lasts = src + (len-1);
      char *lastd = dest + (len-1);
      while (len--)
        *(char *)lastd-- = *(char *)lasts--;
    }
}
```

BIBLIOGRAPHIE

Douglas **COMER**
TCP/IP : Architecture, protocoles, application.

W.RICHARD **STEVENS**.
UNIX network programming

Documentation Solaris sur les Sockets

Rapport **ENSIMAG** de Morel et Tomassino.1993
TCP_IP : programmation réseau; les sockets.

Rapport **ENSIMAG** de Borderies, Chatel, Dens, Reis, 1993
Administration réseau