Usage de TCP/IP en APL2

Par Bernard Mailhol

1 Présentation

TCP/IP est un outil très répandu, ensemble de protocoles permettant des transmissions entre machines d
donne un exemple d'usage, en APL2.

L'application de TCP/IP la plus connue, aujourd'hui, est le Web. Nous allons ainsi décrire comment écr
répondre à un serveur externe.

Ce texte se réfère à des présentations effectuées à Moscou, en Novembre 1996, et à Paris, en Décembre
"Web", et peuvent être visualisées par votre fureteur Web.

Elles sont bientôt à votre disposition, accompagnées de la programmation en APL, sur le serveur Web d

*Cet article a été publié dans le numéro 22 des Nouvelles, mais des incidents combinés ont fait que les pr
nécessaires. C'est pourquoi nous rééditions cet article, complété à l'occasion par l'usage de TCP/IP en mode cli*

B. Mailhol, 100317.3113@Compuserve.com

2 Ce qu'est TCP/IP

TCP/IP est un mot mythique, lié à Internet, laissant planner le plus grand mystère quant à son domaine,

TCP/IP est en fait un ensemble de protocoles de transmission, utilisés sur le réseau Internet, répondant à
universelles, mais simplifiées.

Ce protocole est certes le protocole utilisé sur Internet, mais c'est aussi un protocole utilisable sur un rés
TCP/IP dans un réseau d'entreprise, ce réseau peut s'appeler un *Intranet*.

2.1 TCP/IP EST UN ENSEMBLE DE PROTOCOLES

TCP/IP est composé de protocoles, élaborés depuis plus de 20 ans, ayant tout d'abord permis - dans le m
par ce réseau universitaire appelé Internet.

Récemment, ce réseau s'est ouvert à des "fournisseurs" (providers), ayant la possibilité d'accueillir des a
chacun d'entre nous de nous connecter, mais déstabilise son usage initial : le partage d'informations non commer

L'ensemble de ce réseau fonctionne en TCP/IP.

On voit ainsi que TCP/IP :

1 - Sait utiliser de nombreux supports de transmission

2 - Est reconnu sur de nombreuses machines (sinon toutes les machines).

... il apparaît comme étant universel.

1 - TCP/IP sait utiliser de nombreux supports de transmission

Les protocoles TCP/IP peuvent utiliser de nombreux supports de transmission. Parmi eux, on peut citer

*Les réseaux locaux (Ethernet, Token-Ring, FDDI ...)

*Les réseaux téléphoniques "permanents" (X25)

*Les réseaux téléphoniques "intermittents" (réseau commuté), selon les protocoles SLIP (Serial Line Int

Ces protocoles physiques sont reconnus et validés par un organisme indépendant, sur des propositions v
protocoles sont ainsi le fait d'un consensus d'utilisateurs et de fournisseurs.

Une même machine peut disposer en même temps de liens de différentes natures : elle devient alors un *r*
Toute installlation IP a la capacité potentielle d'agir en routeur.

2 - TCP/IP est reconnu sur de nombreuses machines (sinon toutes les machines).

En 1997, toutes les machines proposent la connexion d'au moins un support de transmission (des réseau
prise modem pour les machines familiales).

Pratiquement tous les systèmes d'exploitation contiennent des protocoles TCP/IP. (Si ces systèmes n'en
fournissent).

En résumé, toutes les machines peuvent utiliser TCP/IP sur tous types de support. Encore faut-il que les

2.2 PROPRIÉTÉS

TCP/IP est un ensemble de protocoles destinés à permettre une communication entre Universités. On c
d'un seul organisme, mais issue d'un consensus.

L'idée directrice, lors de la création de TCP/IP - et d'Internet - a été la diffusion rapide des thèses et autr
l'antériorité que chaque chercheur doit normalement poursuivre. De même, ce réseau devait faciliter la transmissi

Les propriétés de TCP/IP sont ainsi déduites de ces buts, et de ce public :

1 - Installation facile, mais chacun doit configurer

2 - Indépendance de chacun,

3 - Pas de supervision centralisée du réseau

4 - Routages universels, sans discrimination selon les extrémités

5 - Très peu de confidentialité

6 - égalité de tous devant les transmissions (pas de priorité, pas de notion de qualité de service garantie)

7 - Pas de facturation !

3 Utiliser TCP/IP depuis APL/2

APL2 permet depuis longtemps, via le processeur auxiliaire AP119, l'usage de TCP/IP en mode natif, ou

3.1 USAGE DIRECT ET INDIRECT

APL2 permet la mise en oeuvre de TCP/IP selon deux méthodes complémentaires:

1 - La première méthode consiste à partager des variables entre deux machines différentes, selon les mê
variables entre deux APL situés sur la même machine.

La différence est que les temps de transmission ne sont plus négligeables. Les applications doivent pren
positionner les valeur du $\square SVC$ et assurer un verrouillage mutuel).

On peut ainsi monter une application client/serveur entre plusieurs machines en APL, en ne se servant q
mais souvent suffisant (Je me sers parfois de cette technique).

On peut aussi travailler sur une machine, mais utiliser des processeurs auxiliaires situés sur une autre ma
fichiers d'une autre machine, ou sur le lecteur de disquette d'un PC, tout en étant situé sur une machine ne dispos

Le gestionnaire de session n'étant qu'un processeur auxiliaires, rien n'empêche de piloter une session AP

Les applications sont innombrables.

2 - La seconde méthode consiste à utiliser les primitives de transmission de TCP/IP.

Plusieurs méthodes sont utilisables. APL2, via son processeur AP119 propose l'accès **aux sockets**.

En utilisant alors cette interface *de bas niveau*, on peut dialoguer avec des programmes qui ne sont (peut

On peut alors ouvrir les systèmes APL sur l'univers complet de l'informatique, à condition toutefois de c
transmettre.

Dans la mesure où le monde de TCP/IP est régi par un consensus, des publications numérotées et access
peut toujours connaître et utiliser ces applications - si elles respectent ces règles, évidemment.

3.2 LE PROCESSEUR AUXILIAIRE

La notion de processeur auxiliaire remonte au début des années 70. Un processeur auxiliaire est un prog

1 - Indépendant de l'APL.

Plusieurs processeurs sont fournis avec APL2, mais vous pouvez développer (ou faire développer) vos p
de la documentation standard, et contient des exemples, développés en C.

2 - Ce programme s'exécute indépendamment de la session APL, selon son propre temps, avec les périph
programmation de son choix.

3 - Il peut se synchroniser avec une session APL (ou un autre processeur auxiliaire), et échanger des me

4 - Il peut s'exécuter sur une autre machine que la machine de base, la gestion des transmissions étant ef

De fait ce concept est un concept très contemporain.

Le processeur auxiliaire APL119 permet l'usage de TCP/IP en mode "socket"; il est disponible su

3.3 PROGRAMMER EN SOCKETS

La programmation en **sockets** suit des règles générales, seulement transposées en APL2.

Plusieurs notions sont fondamentales :

1 - Chaque utilisateur est situé sur un **host**. Ce host est aussi bien un site central (disposant de dizaines o
d'un seul utilisateur.

2 - Chaque *host* dispose d'une adresse appelée **adresse ip**. Cette adresse doit être unique sur le réseau. S
liberté de son choix; si vous voulez participer à Internet, vous devez demander une adresse de réseau.

Une *adresse ip* prend la forme de quatre nombres - de 0 à 255 - séparés par un point. Par exemple : *216.*

3 - Une *adresse IP* peut avoir des synonymes (par exemple *www.bmailhol.fr*). Ce synonyme est appelé **n**

4 - Chaque programme, situé sur un *host*, se connecte sur le TCP/IP de sa machine sous un numéro, app

Certains programmes disposent d'un numéro de port *bien connu*, et sont ainsi facilement utilisables.

5 - Chaque programme voulant dialoguer avec le TCP/IP de sa machine via un *numéro de port* doit crée

Vous disposez maintenant du vocabulaire suffisant pour programmer TCP/IP en sockets.

3.4 OUVRIR UNE SESSION D'UN SERVEUR

Un serveur se met à l'écoute des appels entrants. On distingue alors deux catégories de sockets:

1 - Les sockets passifs qui sont à l'écoute des appels entrants

2 - Les sockets actifs, dialoguant avec les clients.

1 - La première étape consiste à ouvrir une session. Pour commencer, créer un socket.

```
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'SOCKET'
('GET SOCKET' APRC TCPIPRC)err APRC
SOCKLIS←CMDRC
```

Le processeur (appelé via la fonction ***tcp***) recoit l'ordre *SOCKET* pour lui demander la création d'un s

La fonction ***tcp*** permet un usage synchrone du processeur AP119, les variables étant partagées préalal

```
Z←tcp CDE
⍝ appeler le processeur TCP/IP
tcpΔ1←CDE
Z←tcpΔ1
```

a - L'objet APL à transmettre au processeur est un vecteur généralisé, dont le premier élément est le mot suivants les paramètres de cette commande.

b - Le retour est un vecteur de trois éléments :

1 - Le premier est le code retour propre du processeur

2 - Le second est le code retour de TCP/IP

3 - Le troisième est le résultat de la commande.

Dans ce cas, le retour est l'identification du socket qui vient qu'être créé.

2 - Associer ce socket à un numéro de port.

```
⍝ lier ce socket au port (spécial) du Web
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'BIND' SOCKLIS 8080 '0.0.0.0'
('BIND SOCKET' APRC TCPIPRC)err APRC
```

Dans cet exemple, le socket est associé au port numéro 8080, sur l'adresse ip locale.

3 - Dans le cas d'un serveur, mettre ce socket à l'écoute des appels entrants.

```
⍝ rendre ce socket passif
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'LISTEN' SOCKLIS 5
('LISTEN' APRC TCPIPRC)err APRC
```

Il dispose d'une file d'attente de 5 appels entrants non encore acceptés.

4 - Il se met en attente du prochain appel entrant

```
⍝ attendre un appel entrant
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'ACCEPT' SOCKLIS
('ACCEPT' APRC TCPIPRC)err APRC
(SOCKBRW PORTBRW IPADDBRW)←CMDRC
'appel entrant de' SOCKBRW IPADDBRW
```

A cette occasion, TCP/IP crée un nouveau socket, dont nous connaissons le numéro. Le retour indique a appelé, ainsi que son numéro de port.

A ce moment, il peut commencer les échanges de messages.

3.5 OUVRIR UNE SESSION D'UN CLIENT

Un client ouvre une session, et a l'initiative de la connexion.

1 - La première étape consiste à ouvrir une session. Pour commencer, créer un socket.

```
⍝ ouvrir un socket
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'SOCKET'
('GET SOCKET' APRC TCPIPRC)err APRC
SOCK←CMDRC ⍝ socket ouvert
```

Le processeur (appelé via la fonction *tcp*) recoit l'ordre *SOCKET* pour lui demander la création d'un s

2 - La seconde étape est d'associer ce socket à un port et une adresse IP.

Dans ce cas présent, le port indiqué (0) fait que le système recherche un port disponible; l'adresse IP ind
première adresse associée à cette machine.

```
⍝ lier ce socket à un port quelconque
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'BIND' SOCK 0 '0.0.0.0'
('BIND' APRC TCPIPRC)err APRC
```

La troisième étape consiste à se connecter sur le serveur dont on connaît l'adresse complète.

```
⍝ se connecter sur le serveur dont on connait l'URL
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'CONNECT' SOCK PORT IP
('CONNECT' APRC TCPIPRC)err APRC
```

3.6 ENVOYER UN MESSAGE

La programmation en sockets vous permet d'envoyer un vecteur alpha.

Le programme de l'interlocuteur n'est peut-être pas écrit en APL. Il n'est alors pas question de lui envo

```
⍝ Envoi d'un message
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'SEND' SOCKBRW 0 'B'(,VCT)
('SEND MESSAGE' APRC TCPIPRC)err APRC
```

Cet appel demande la mise en file d'attente de la chaîne précisée en *VCT*. Un code retour correct signifi
correctement effectuée. Ceci ne signifie nullement que le message sera délivré.

Ce message est seulement considéré comme une suite d'octets, qu'il faut acheminer vers leur destinataire

Plusieurs questions se posent alors :

1 - Puis-je envoyer d'autres messages ?

2 - Où est mon message ?

3 - Quand arrivera-t-il ?

4 - Est-il arrivé ?

Nous devons construire un protocole complémentaire applicatif , pour tenter de répondre à ces questions
personnel.

3.7 RECEVOIR UN MESSAGE

L'ordre de réception permet de recevoir tout ou partie du contenu de la file d'attente.

La réception d'une chaîne vide marque la fin de la session.

```
⍝ recevoir ses données
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'RECV' SOCKBRW 0 'B'
('RECEIVE' APRC TCPIPRC)err APRC
BRW←CMDRC ⍝ données initiales (un seul RECV dans cette démo)
```

Le retour de la commande est un vecteur de caractère.

Les données reçues sont garanties :

1 - Sans erreur de transmission

2 - Les octets reçus sont dans le même ordre que les octets émis.

Mais les messages émis ne sont qu'une suite d'octets, mis en file d'attente, sans notion de séparation entr
sont qu'une suite de caractères, que l'application doit structurer).

Plusieurs questions se posent alors :

1 - Ce retour est-il message est-il complet ?

2 - Est-il un message émis, suivi de tout ou partie d'un autre message ?

Le protocole applicatif que nous devons construire doit pouvoir aussi répondre à ces questions. On cons
initiaux.

Remarquons que cette transmission *de flot* et non *de message* fait qu'un message volumineux peut être e
fichier de 2 MB). Il peut être envoyé par segments de 1000 octets, et reçu par segments de 800 !

4 Utiliser TCP/IP dans des applications

On peut utiliser TCP/IP dans nos applications, soit en utilisant des protocoles personnels, soit en utilisan
TCP/IP. L'usage du protocole du Web est un exemple important de l'usage de protocoles standard.

4.1 PROTOCOLES PERSONNELS

On peut utiliser des protocoles personnels internes à différents composants d'une application.

Par exemple, l'application ADAGIO de la Banque de France (présentée à l'AF/APL par Henri Sinturel),

1 - Un serveur situé sur site central, en VM/CMS, développé en APL2, assurant la gestion des fichiers, d

2 - Les clients situés sur des postes de travail en OS/2, chargés de l'affichage des images et de l'édition g

3 - La liaison entre Client et Serveur se fait en TCP/IP sur un réseau local Token-Ring à 16 Mbits.

On peut aussi imaginer des applications mettant en oeuvre des protocoles TCP/IP personnels sur le réseau
l'interdisant.

4.2 PROTOCOLES STANDARD

APL2 et son processeur AP119 permettent aussi d'utiliser des protocoles standard. Ces protocoles sont d
For Comments - RFC. Chaque RFC est numéroté.

On peut ainsi piloter une imprimante, en connaissant le protocole LPR/LPD, échanger du courrier, trans
simulant la présence de fichiers en APL ...)

En particulier, aujourd'hui, vous pouvez découvrir et mettre en place **le protocole HTTP**(hypertext tran

Si vous disposez d'une application connaissant ou calculant des données importantes, vous pouvez les m
disposent ou non d'un système APL, quelque soit la machine sur laquelle ils travaillent, via un *Serveur Web écrit*

Vos utilisateurs seront impressionnés.

4.3 INSTALLER UN SERVEUR WEB EN APL2

L'installation d'un serveur Web en APL2 demande la connaissance :

1 - Des concepts du *protocole HTTP* (RFC 1945)

2 - Des concepts du *langage html*, dans lequel sont construites les pages du Web

3 - Des concepts du *Header mime* (eux-aussi dans un RFC), introduisant la réponse.

La section suivante donne un exemple concret de serveur écrit en APL2.

Ses fonctions permettent :

1 - De se connecter sur TCP/IP, de se mettre à l'écoute

2 - De recevoir un message complet. (dans ce cas, il est reçu en un seul bloc, vu sa taille)

3 - D'analyser ce message, d'extraire le nombre à multiplier par deux

4 - De construire un texte html pour présenter cette réponse

5 - De préfixer de texte par un header Mime

6 - D'envoyer la réponse

7 - De clore la session.

5 Un serveur Web sommaire

Ce serveur Web est très sommaire : **Il permet de multiplier par deux** (mais le vôtre le sera moins).

5.1 SA PROGRAMMATION

Il est composé de quelques fonctions (développées en origine 0)

1 - START Lancer l'attente générale

2 - REQUEST Attendre et traiter une requête

3 - SCRIPT Traiter (par un script) le message reçu

4 - MIME Préparer le header Mime de la réponse

5.1.1 START Lancement général

Dans cette démonstration, cette fonction est appelée (manuellement) pour lancer le serveur. Le port stan
avons choisi ici le port 8080, ce qui permet d'installer ce serveur spécifique à côté d'un serveur standard, mais im
8080.

```
START
⍝ commencer une session WEB
⍝  1 - créer un socket passif
⍝  2 - attendre sur le port 8080
⍝
CLOSE
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'SOCKET'
```

```
('GET SOCKET' APRC TCPIPRC)err APRC
SOCKLIS←CMDRC
⍝ lier ce socket au port (spécial) du Web
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'BIND' SOCKLIS 8080 '0.0.0.0'
('BIND SOCKET' APRC TCPIPRC)err APRC
⍝ rendre ce socket passif
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'LISTEN' SOCKLIS 5
('LISTEN' APRC TCPIPRC)err APRC
```

5.1.2 REQUEST Attendre et traiter une requête

Cette fonction est appelée pour attendre et traiter un seul message (démonstration oblige).

La variable *BRW* est le texte reçu du fureteur (browser en langue d'origine), la variable *RSP* est le vecte

```
REQUEST
⍝ attendre un appel entrant
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'ACCEPT' SOCKLIS
('ACCEPT' APRC TCPIPRC)err APRC
(SOCKBRW PORTBRW IPADDBRW)←CMDRC
'appel entrant de' SOCKBRW IPADDBRW
⍝
⍝ recevoir ses données
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'RECV' SOCKBRW 0 'B'
('RECEIVE' APRC TCPIPRC)err APRC
BRW←CMDRC ⍝ données initiales (un seul RECV dans cette démo)
⍝
⍝ les données sont la commande HTTP du fureteur (browser)
RSP←IPADDBRW SCRIPT BRW
⍝
⍝ il reste à envoyer la réponse
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'SEND' SOCKBRW 0 'B'(,RSP)
('SEND RESPONSE' APRC TCPIPRC)err APRC
⍝
⍝ et terminer la session.
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'CLOSE' SOCKBRW
('CLOSE SOCKET' APRC TCPIPRC)err APRC
```

5.1.3 SCRIPT Traiter (par un script) le message reçu

Le message reçu du fureteur Web suit les procédures HTTP. La chaîne reçue comprend plusieurs lignes,

1 - Première ligne

a - Le mot GET

b - Les paramètres du get

c - Le nom et la version du protocole

2 - Lignes suivantes

Chaque ligne suivant donne un paramètre du fureteur.

3 - Fin du message

La fin du message est marquée par une ligne vide.

La réponse à faire au fureteur comprend deux parties :

1 - Un *Header Mime* expliquant le contenu de la seconde partie

2 - Une ligne vide pour marquer la fin du header mime.

3 - (dans ce cas) un texte en langage HTML, décrivant l'affichage souhaité.

Le header donne la taille du message en réponse, ce qui permet au fureteur d'en détecter la fin.

La trace de la session montre le message reçu, la réponse fournie.

```
RSP←IPADD SCRIPT BRW;K;W;N
⍝ la requête du fureteur est : GET nnn HTTP/1.0 ...
'Requête reçue' ⎕TS
BRW
W←(K≠' ')⊂K←⎕eb1 30↑BRW ⍝ ignorer les autres éléments
N←'0' ⎕EA 1↓1⊃W
⍝ construire la réponse
K←⊂'<!- APL Demo B. Mailhol -->'
K←K,⊂'<HTML><HEAD><TITLE>WEB en APL2</TITLE></HEAD>'
K←K,⊂'<BODY><H1>Retour : le double de l''entr&eacute.e</H1>'
K←K,⊂'<hr><p>L''entr&eacute.e est...',(⍕N)
K←K,⊂'<p>La r&eacute.ponse est ainsi ... ',(⍕2×N),'<hr></BODY></HTML>
⍝ ajouter le header MIME
RSP←MIME∈K,¨⊂⎕AF 13 10
'Réponse au fureteur'
RSP
```

5.1.4 MIME Préparer le header Mime de la réponse

```
HEAD←MIME RSP
⍝ construire la réponse avec son header
K←'HTTP/1.0 200 Document follows.' 'Server: BMA APL Demo'
K←K,'Content-Type: text/html'('Content-Length: ',⍕⍴RSP)''
⍝ header puis la réponse
HEAD←(∈K,¨⊂⎕AF 13 10),RSP
```

5.2 SON USAGE

Ce serveur est appelé par un message de la forme **http://apl.demo:8080/543**. Cette demande appelle le s
apl.demo, sous le numéro de port *8080*. Le message envoyé comprend 543.

La réponse doit alors être 1086, le double de 543. Cette réponse est présentée, dans un texte html.

Appel entrant de 31 216.94.110.95

Requête reçue 1997 2 18 23 52 35 90

```
GET /543 HTTP/1.0
Accept: */*; q=0.300
Accept: application/octet-stream; q=0.100
Accept: text/plain
Accept: text/html
Accept: application/book
Accept: application/hlp
Accept: application/inf
Accept: text/plain
Accept: audio/x-wav
............e/tiff
Accept: image/jpeg
```

```
Accept: image/gif
Accept: application/editor
User-Agent: IBM-WebExplorer-DLL/v1.1f
```

Réponse au fureteur

```
HTTP/1.0 200 Document follows.
Server: BMA APL Demo
Content-Type: text/html
Content-Length: 224

<!- APL Demo B. Mailhol -->
<HTML><HEAD><TITLE>WEB en APL2</TITLE></HEAD>
<BODY><H1>Retour : le double de l'entrée</H1>
<hr><p>L'entrée est...543
<p>La réponse est ainsi ... 1086<hr></BODY></HTML>
```

Il ne vous reste qu'à essayer ... avec un service un peu plus consistant que cette multiplication. Vous alle

6 Un client Web sommaire.

Ce client Web sommaire est lui aussi très simple : à partir d'un URL, ce client se connecte sur le site, uti extrait de la réponse l'ensemble des références qu'elle contient.

On peut imaginer que cette routine serve à charger localement l'ensemble des pages liées à un URL, pou connexion.

Encore une fois, il suffit de très peu de fonctions APL pour parvenir à cette fonctionnalité intéressante.

Un exemple se sert de l'URL de Gérard Langlet.

6.1 LA PROGRAMMATION

La programmation comprend

1 - CALL_URL Appeler l'URL, globalement

2 - ANZ_URL Eclater l'URL en ses composants simples

3 - GET_URL Lire un URL, après connexion

4 - LINKS Extraire les liens d'un URL lu précédemment.

En voici l'expression (développée en origine 0)

6.1.1 CALL_URL Appeler l'URL, globalement

Cette fonction se connecte sur l'URL donné en argument, en utilisant les primitives TCP/IP décrites ci-d

La variable *rz* est un code retour général, dans lequel le premier élément est un code retour pur (vecteur en cas de succès, ou d'un message d'erreur sinon.

Le retour normal contient le texte de la *page Web* interrogée.

```
rz←CALL_URL URL;PORT;IP;ARG;SOCK;APRC;TCPIIPRC;CMDRC
⍝ appeler un URL http, rendre son texte
→0×⍴(pCC PORT IP ARG)←4↑rz←ANZ_URL URL
⍝ ouvrir un socket
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'SOCKET'
('GET SOCKET' APRC TCPIPRC)err APRC
SOCK←CMDRC ⍝ socket ouvert
```

```
⍝ lier ce socket à un port quelconque
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'BIND' SOCK 0 '0.0.0.0'
('BIND' APRC TCPIPRC)err APRC
⍝ se connecter sur le serveur dont on connait l'URL
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'CONNECT' SOCK PORT IP
('CONNECT' APRC TCPIPRC)err APRC
⍝ envoyer la requête
→0×⍎rz←SOCK GET_URL ARG
⍝ la session est fermée
```

6.1.2 ANZ_URL Eclater l'URL en ses composants simples

Cette fonction utilise la structure de l'URL, en protocole *http*

1 - Le nom du protocole *http:*

2 - Un double slash indiquant que l'URL n'est pas relatif à la page dans laquelle nous sommes

3 - Le nom du domaine

4 - (éventuellement) le numéro du port interrogé, s'il est différent de 80

5 - Les arguments, commençant sur le slash /.

Cette fonction contient un nouvel appel de TCP/IP : l'adresse IP du correspondant peut être donnée soit s
soit par le nom de son domaine (*'www.afapl.fr'*).

La connexion permise par ce processeur AP119 utilise la forme numérique de l'adresse IP. Il est alors né
adresse numérique. Ceci est effectué par la commande *GETHOSTBYNAME* de TCP/IP - AP119.

Cette commande fait alors appel à un site externe, appelé "Domain Name Server", qui connaît les tables
les adresses numériques.

```
rz←ANZ_URL URL;I0;I1;PORT;IP;AREG;N;K
⍝ cette fonction éclate un URL entre PORT, adresse IP, et argument du
⍝ l'URL est de la forme http://<adresse IP>:<port (ou 80)><argument (
→0×'L''URL doit être en protocole http' srvERR~(7↑URL)≡'http://'
N←(K⍳':/'),⍴K←7↓URL
(IP ARG)←((I0←⌊/N)↑K)((I1←1⌷N)↓K)
→(∧/IP∈'.0123456789')/L0
⍝ rechercher l'adresse IP par son numéro
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'GETHOSTBYNAME' IP
('GETHOSTBYNAME' APRC TCPIPRC)err APRC
IP ← CMDRC
L0:PORT←⍎1↓'0 80' ⎕EA '0,',(I0+1)↓I1↑K
rz←(⍳0)PORT IP((0≠⍴ARG)⊃'/' ARG)
```

6.1.3 GET_URL Lire un URL, après connexion

Cette fonction est appelée après ouverture d'une session.

Elle doit transmettre la demande, et recevoir l'ensemble de la réponse.

Le serveur doit clore la session, dès la fin de l'envoi.

```
rz←SOCK GET_URL ARG;K;L
⍝ envoyer un URL, la session est ouverte
'envoi des données'
K←'GET ',ARG,' HTTP/1.0',nl,accept
⍝ envoyer cette chaîne vers le serveur
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'WRITE' SOCK 'B' K
```

```
('WRITE' APRC TCPIPRC)err APRC
⍝ lire la réponse jusqu'à fermeture de la session
K←L←''
⍝ boucle sur des tronçons de réponse
B0:K←K,L
(APRC TCPIPRC CMDRC)←tcp 'TCPIP' 'READ' SOCK 'B'
('READ' APRC TCPIPRC)err APRC
→(0≠⍴L←CMDRC)/B0 ⍝ portion de message rendu
⍝ le message est complet, la session est fermée
rz←(⍳0)K
```

6.1.4 LINKS Extraire les liens d'un URL lu précédemment.

Cette dernière fonction doit rechercher, dans la chaîne passée en argument (sous forme de *rz*), tous les cl

Cette fonction calcule un vecteur dont chaque élément est le vecteur donnant le nom de la référence. La meilleure présentation.

```
HREF←LINKS rep;CH;M;K
⍝ rechercher les liens de cet URL
(1⊃rep)srvERR↑rep
⍝ explorer la réponse href="  ...  "
M←'href='⍷CH←min[⎕AF eb 1⊃rep]
HREF←cOL(K⍳¨'"')↑¨K←5↓¨1↓(~M)⊂CH
```

6.2 EXEMPLE D'USAGE

Cet exemple utilise l'URL de Gérard Langlet.

1 - Le premier appel extrait la réponse html de l'accès dans la variable appelée *Gérard* (APL2 permet d'u variables. Ne nous en privons pas) :

```
   URL←'http://www.ensmp.fr/~scherer/langlet/'
   Gérard ← CALL_URL URL
   ⍴¨Gérard
 0  4893
   500↑1⊃Gérard
HTTP/1.1 200 OK
Date: Mon, 12 May 1997 21:10:46 GMT
Server: Apache/1.2b2
Connection: close
Content-Type: text/html
Last-Modified: Sat, 03 May 1997 20:11:54 GMT
ETag: "2f85e-122c-336b9c0a"
Content-Length: 4652
Accept-Ranges: bytes

<HTML>
<head>
<TITLE>L'oeuvre de G&eacute;rard LANGLET ?</TITLE>
<!-- Thank you for downloading this HTML source page.-->
<!-- You win our best thanks and are pleased to help us to improve it
<!-- Send your comments to Christian Scherer -->
<link rev=mad
```

Le texte précédent est limité aux 500 premiers caractères.

2 - Le second appel analyse le contenu de cette réponse

```
LINKS Gérard

mailto:scherer@ensmp.fr
whoisgl.html
transform.html
erreur.txt
erreur.doc
http://cri.ensmp.fr/~scherer/langlet/nvapl15/
http://cri.ensmp.fr/~scherer/langlet/nvapl16/nvapl16.html
http://cri.ensmp.fr/~scherer/langlet/nvapl17/
http://cri.ensmp.fr/~scherer/langlet/nvapl18/
http://cri.ensmp.fr/~scherer/langlet/nvapl19/
http://cri.ensmp.fr/~scherer/langlet/nvapl19/page69.jpeg
http://cri.ensmp.fr/~scherer/langlet/nvapl20/
http://cri.ensmp.fr/~scherer/langlet/nvapl21/
http://cri.ensmp.fr/~scherer/langlet/nvapl22/
http://cri.ensmp.fr/~scherer/langlet/nvapl16/histapl.htm
http://users.aol.com/lemagnen/Sciences_et_Cosmos/
http://www.vector.org.uk/
http://www.vector.org.uk/baa.html
http://www.vector.org.uk/aplfont.html
http://www.vector.org.uk/apl97.html
http://www.yahoo.com/computers/languages/apl/
http://www.acm.org/sigapl/
http://www.torolab.ibm.com/ap/apl/apl2.html
http://webzone1.co.uk/www/dogon/apl.htm
http://www.adminet.com/mail/hoffmann.22feb97.txt
http://www.ensmp.fr/~scherer/mail/apl.2feb96
http://www.ensmp.fr/~scherer/deuriat/
http://math.uwaterloo.ca/~ljdickey
mailto:scherer@ensmp.fr
http://www.ensmp.fr/~scherer/
```

On peut constater qu'une page Web fait très souvent appel à de nombreuses autres pages Web, afin que

Ces pages sont mises en place par Christian Scherer, que nous remercions vivement.

7 Sources documentaires

Ces programmes liés au Web sont développés selon des définitions trouvées dans diverses sources :

1 - **APL/2 for OS/2 User's guide** Description du processeur AP119

2 - **TCP/IP Reference manual** Warp Online reference + Toolkit, décrit le comportement des primitives

3 - **RFC1945** Hypertext Transfer Protocol HTTP/1.0

4 - **RFC1521** Mime Header

5 - **HTML**, le format des textes transmis, est décrit dans de nombreux ouvrages. Il varie avec chaque fu
Musciano & Bill Kennedy - O'Reilly & Associates) est une bonne synthèse des dialectes reconnus en HTML 2.*

6 - **Les nouvelles d'APL**, pour connaître l'URL de son serveur.

Une trace "IP" des échanges entre un fureteur et un serveur m'ont aussi été d'un grand secours, pour déc
synchronisations... Mais vous en avez ici la synthèse