Usage de TCP/IP en APL2
par Bernard Mailhol

1 Présentation

TCP/IP est un outil très répandu, permettant des transmissions entre machines diverses. Ce texte introduit ces protocoles, puis en 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 écrire un serveur Web spécifique, en APL2.

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

Elles sont à votre disposition, accompagnées de la programmation en APL, sur le serveur Web de l'AF/APL.

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, son universalité.

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

Ce protocole est certes le protocole utilisé sur Internet, mais c'est aussi un protocole utilisable sur un réseau indépendant d'Internet. Dans le cas d'usage de 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 monde Unix - des transmissions entre universités, par ce réseau universitaire appelé Internet.

Récemment, ce réseau s'est ouvert à des "fournisseurs" (providers), ayant la possibilité d'accueillir des abonnés non universitaires. Cette ouverture permet à chacun d'entre nous de nous connecter, mais déstabilise son usage initial : le partage d'informations non commerciales.

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 Internet Protocol) ou PPP (Point to Point Protocol).

Ces protocoles physiques sont acceptés par un organisme indépendant, sur des propositions venant des utilisateurs, et implémenteurs. Ces protocoles sont ainsi le fait de consensus.

Une même machine peut disposer en même temps de liens de différentes natures.

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 réseaux à grands débits pour les sites centraux, jusqu'à la prise modem pour les machines familiales).

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

En résumé, toutes les machines peuvent utiliser TCP/IP sur tous types de support. Encore faut-il que les flux échangés soient aussi normalisés !

2.2 PROPRIETES

TCP/IP est un ensemble de protocoles destinés à permettre une communication entre Universités. On comprend alors que sa définition ne soit pas le fait d'un seul, 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 autres publications universitaires, dans la course à l'antériorité que chaque chercheur doit normalement poursuivre. De même, ce réseau devait faciliter la transmission de messages - écrits en anglais.

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 en mode caché.

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 mêmes méthodes que celles permettant le partage de variables entre deux APL situés sur la même machine.

La différence est que les temps de transmission ne sont plus négligeables. Les applications doivent prendre un minimum de précautions (par exemple, positionner les valeur du □□□□ et assurer un verrouillage mutuel).

On peut ainsi monter une application client/serveur entre plusieurs machines en APL, en ne se servant que des variables partagées. C'est très sommaire 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 machine. D'une machine, on peut travailler sur des fichiers d'une autre machine, ou sur le lecteur de disquette d'un PC, tout en étant situé sur une machine ne disposant pas de lecteur de disquette...

Le gestionnaire de session n'étant qu'un processeur auxiliaires, rien n'empêche de piloter une session APL depuis une autre machine.

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-être) pas écrits en APL2.

On peut alors ouvrir les systèmes APL sur l'univers complet de l'informatique, à condition toutefois de connaître le flot de données à recevoir et à transmettre.

Dans la mesure où le monde de TCP/IP est régi par un consensus, des publications numérotées et accessibles (appelées *Request For Comments RFC*), on 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 programme

1 - Indépendant de l'APL.

Plusieurs processeurs sont fournis avec APL2, mais vous pouvez développer (ou faire développer) vos propres processeurs : leur documentation fait partie 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 périphériques de son choix, ou les interfaces de programmation de son choix.

3 - Il peut se synchroniser avec une session APL (ou un autre processeur auxiliaire), et échanger des messges via un mécanisme de boîte aux lettres.

4 - Il peut s'exécuter sur une autre machine que la machine de base, la gestion des transmissions étant effectuées par le gestionnaire de partage d'APL2.

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 sur tous les environnments d'APL2

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 ou milliers d'utilisateurs) qu'un PC, ne disposant que d'un seul utilisateur.

2 - Chaque *host* dispose d'une adresse appelée **adresse ip**. Cette adresse doit être unique sur le réseau. Si vous créez votre propre réseau, vous avez la 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 : *216.94.110.76*.

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

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

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 créer une **session** via un **socket**.

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

3.4 OUVRIR UNE SESSION

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 socket.

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

```
Z←tcp CDE
⍝ appeler le processeur TCP/IP
tcp∆1←CDE
Z←tcp∆1
```

a - L'objet APL à transmettre au processeur est un vecteur généralisé, dont le premier élément est le mot *'TCPIP* le second la commande passée, les 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 aussi l'adresse IP de l'interlocutaur nous ayant appelé, ainsi que son numéro de port.

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

3.5 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 envoyer un objet APL, qu'il en comprendrait pas.

```
⍝ 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 □□□. Un code retour correct signifie seulement que ... la mise en file d'attente a été 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. Chaque application dispose alors de son protocole personnel.

3.6 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 entre messages (de même que les fichiers sous Unix ne 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 constate plusieurs types de reconstitution des messages initiaux.

Remarquons que cette transmission *de flot* et non *de message* fait qu'un message volumineux peut être envoyé en plusieurs tronçons. (par exemple, un 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 utilisant des protocoles standard dans la communauté 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), est articulée autour de deux composants :

1 - Un serveur situé sur site central, en VM/CMS, développé en APL2, assurant la gestion des fichiers, de l'historique, des reprises, les calculs...

2 - Les clients situés sur des postes de travail en OS/2, chargés de l'affichage des images et de l'édition graphique. Ils sont développés en C.

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 Internet. Il n'y a aucune contrainte *technique* l'interdisant.

4.2 PROTOCOLES STANDARD

APL2 et son processeur AP119 permettent aussi d'utiliser des protocoles standard. Ces protocoles sont définis dans des documents publics, appelés *Request For Comments - RFC*. Chaque RFC est numéroté.

On peut ainsi piloter une imprimante, en connaissant le protocole LPR/LPD, échanger du courrier, transmettre et recevoir des fichiers par FTP. (en simulant la présence de fichiers en APL ...)

En particulier, aujourd'hui, vous pouvez découvrir et mettre en place **le protocole HTTP**(hypertext transfer protocol), suivi par les systèmes "Web".

Si vous disposez d'une application connaissant ou calculant des données importantes, vous pouvez les mettre à disposition de vos interlocuteurs, qu'ils disposent ou non d'un système APL, quelque soit la machine sur laquelle ils travaillent, via un *Serveur Web écrit en APL*.

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

1 - start

Dans cette démonstration, cette fonction est appelée (manuellement) pour lancer le serveur. Le port standard *bien connu* d'un serveur Web est 80. Nous avons choisi ici le port 80, ce qui permet d'installer ce serveur spécifique à côté d'un serveur standard, mais impose que son appel précise ce numéro de port, 8080.

```
START
⍝ commencer une session WEB
⍝  1 - créer un socket passif
⍝  2 - attendre sur le port WEB : 80
⍝
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
```

2 - request

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 □□□ est le vecteur alphanumérique à renvoyer en réponse.

```
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
```

3 - script

Le message reçu du fureteur Web suit les procédures HTTP. La chaîne reçue comprend plusieurs lignes, séparées par un code de saut de ligne.

a - Première ligne

1 - Le mot GET

2 - Les paramètres du get

3 - Le nom et la version du protocole

b - Lignes suivantes

Chaque ligne suivant donne un paramètre du fureteur.

c - Fin du message

La fin du message est marquée par une ligne vide.

La réponse à faire au fureteur comprend deux parties :

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

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

c - (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&amp.eacute.e</H1>'
K←K,⊂'<hr><p>L''entr&amp.eacute.e est...',(⍕N)
K←K,⊂'<p>La r&amp.eacute.ponse est ainsi ... ',(⍕2×N),'<hr></BODY></HT
ML>'
⍝ ajouter le header MIME
RSP←MIME∈K,¨⊂⎕AF 13 10
'Réponse au fureteur'
RSP
```

4 - mime

```
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 serveur Web situé

à l'adresse IP décrite en *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
&dot............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 allez émerveiller vos utilisateurs.