Exercice 1. Pile de protocoles

La pile de protocoles internet est constituée de 5 couches :

(a) donnez les noms de ces 5 couches
(b) pour chacune de ces couches rappelez le nom des unités de données (PDU) qui y circulent
(c) pour chaque couche, citez des exemples de protocoles opérant sur celle-ci

Exercice 2. POP3

Le protocole POP3 est défini par RFC 1939. Vous pouvez trouver ce document en ligne à l'URL suivant : `http://www.ietf.org/rfc/rfc1939.txt`, et traduit en français à l'URL :

`http://abcdrfc.free.fr/rfc-vf/rfc1939.html`

Connection au serveur. Par convention, un serveur POP est à l'écoute sur le port 110 (voir `/etc/services`) :

```
% telnet pop.free.fr 110
Trying 212.27.48.3...
Connected to pop.free.fr.
Escape character is '^]'.
```

Autorisation. dès que nous nous sommes connectés, le serveur entre dans une phase dite *d'autorisation*. Dans cette phase, le client doit s'authentifier. Pour rendre les échanges entre client et serveur plus lisibles, nous préfixerons avec "`C:` " les messages émis par le client et avec "`S:` " les réponses du serveur. Ces préfixes ne font évidemment pas partie des messages : il servent simplement (dans ce document) à indiquer qui dit quoi. A priori, on aurait aussi bien pu surligner les lignes émises par le serveur en rouge, et celles émises par le client en vert. Voici la première réponse du serveur ; elle signale qu'il est entré en phase d'autorisation.

```
S: +OK <29752.1157807659@pop1-g25.free.fr>
```

Le client dispose alors des commandes `USER` et `PASS` pour (respectivement) s'identifier et s'authentifier, ou bien de la commande `QUIT` pour terminer la session et couper la connection.

```
C: USER martin
S: +OK martin is a known user
C: PASS skippyx
S: +OK martin has 2 messages
```

Transaction. Lorsque le client s'est correctement authentifié, le serveur passe alors dans la phase dite de *transaction*. La commande `LIST` permet de lister les messages présents dans la boite : pour chaque message, le serveur donne le numéro du message suivi du nombre d'octets de ce message. La fin de la liste est indiquée par une ligne contenant juste un point :

```
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
```

La command `RETR` (c'est à dire *retrieve*) permet d'obtenir un message grace à son numéro. La fin du message est indiquée par une ligne contenant juste un point :

```
C: RETR 1
S: +OK 120 octets
S: <le contenu du message>
S: .
```

La commande `DELE` (c'est à dire *delete*) permet de supprimer un message grace à son numéro. Le message est juste marqué comme devant être supprimé. La suppression proprement dite ne prend place qu'à la fin de la session, lorsque le client émet la commande `QUIT`.

```
C: DELE 1
S: +OK message 1 deleted
```

Update. La commande `QUIT` permet de terminer la session. Le serveur entre en phase *update* (c'est à dire de mise à jour) ; les messages marqué pour la suppression sont effectivement éliminé ; la connection est ensuite coupée.

Exercice : Ecrivez un script bash pour obtenir et éliminer le premier message dans votre boite de messagerie (celle dont vous disposez sur `info2`). Vous utiliserez `netcat` au lieu de `telnet`. Ces deux programmes prennent les même arguments et remplissent essentiellement les même fonctions : `telnet` est plus spécialement conçu pour l'interaction avec un utilisateur sur un terminal et mal-adapté au scripting, alors que `netcat` est plus spécialement dédié au scripting et est mal-adapté à l'interaction avec un utilisateur sur un terminal.

POP3 en bref

`USER` *nom*	fournit le nom de l'utilisateur
`PASS` *secret*	fournit le mot de passe
`LIST`	listing bref des messages présents
`RETR` *n*	obtient le message numéro *n*
`DELE` *n*	supprime le message numéro *n*
`QUIT`	termine la transaction et ferme la connection

Exercice 3. SMTP

Le protocole SMTP est défini par RFC 821. Vous pouvez trouver ce document en ligne à l'URL suivant : `http://www.ietf.org/rfc/rfc0821.txt`, et traduit en français à l'URL :

`http ://abcdrfc.free.fr/rfc-vf/rfc821.html`

Par convention, un serveur SMTP est à l'écoute sur le port 25. Les réponses du serveur commence toujours par un entier : celui-ci indique le *statut* de la réponse. Un statut de la forme 2XY indique que la commande (ou ici la connection) a été correctement exécutée.

```
% telnet bbn-unix.arpa 25
S: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
```

La commande `HELO` permet au client de se présenter : elle prend comme argument le nom de domaine du client. La réponse contient le nom de domaine du serveur :

```
C: HELO USC-ISIF.ARPA
S: 250 BBN-UNIX.ARPA
```

Le client indique ensuite de qui provient le message grace à la commande `MAIL`. Cette information correspond à l'en-tête `From:` dans vos couriels :

```
C: MAIL FROM:<Smith@USC-ISIF.ARPA>
S: 250 OK
```

Le client indique ensuite le destinataire du message grace à la commande `RCPT` (c'est à dire *recipient* : destinaire en anglais). Cette information correspond à l'en-tête `To:` dans vos couriels :

```
C: RCPT TO:<Jones@BBN-UNIX.ARPA>
S: 250 OK
```

Voici un example ou le destinataire n'existe pas. Notez le statut 550 dans la réponse : ceci indique une erreur.

```
C: RCPT TO:<Green@BBN-UNIX.ARPA>
S: 550 No such user here
```

On peut ajouter des destinaires supplémentaires en répétant la commande `RCPT` :

```
C: RCPT TO:<Brown@BBN-UNIX.ARPA>
S: 250 OK
```

La commande `DATA` indique au serveur que le client va maintenant envoyer le contenu du message : toutes les lignes suivantes envoyées par le client sont accumulées par le serveur pour former le corps du message. La fin du message est indiquée par une ligne contenant un seul point :

```
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
```

Le client peut alors mettre fin à la session et fermer la connection grace à la commande `QUIT` :

```
C: QUIT
S: 221 BBN-UNIX.ARPA Service closing transmission channel
```

Exercice : Ecrivez un script bash pour envoyer un couriel (vous pouvez mettre le texte du message lui-même dans le script).

SMTP en bref

`HELO` *domain*	permet au client de s'identifier
`MAIL FROM :` *addresse*	indique l'expéditeur
`RCPT TO :` *addresse*	indique le destinataire
`DATA`	indique que les lignes suivantes forme le contenu du message. La fin du message est signalée par une ligne contenant juste un point.
`QUIT`	termine la session et ferme la connection
`RSET`	annule la transaction en cours
`VRFY nom`	vérifie qu'un utilisateur est géré par le serveur SMTP
`EXPN liste`	affiche les membres d'une liste de diffusion
`HELP` [*command*]	affiches des infos sur les commandes disponibles, ou sur une commande particulière

Exercice 4. Serveur clé/valeur (début du TD2)[1]

Une *table associative* est une structure de données qui permet d'associer des valeurs à des clés : chaque clé présente dans la table est associée à une seule valeur. Elle offre les opérations suivantes :

`PUT` étant donné une clé K et une valeur V, la table est modifiée pour associer la valeur V à la clé K. Si la clé K était déjà présente dans la table et était associée à une valeur W, alors W est maintenant remplacée par V.

`GET` étant donné une clé K, consulter la table pour obtenir la valeur V qui lui est associée. Si K n'est pas présente dans la table, une erreur est signalée.

`DEL` étant donnée une clé K, éliminer cette clé de la table. Après cette opération, tenter un `GET` sur la clé K produira une erreur.

La plupart des langages de programmation offre des tables associatives. Dans cet exercice, nous voulons offrir les services d'une table associative sous la forme d'un serveur. Nous admettons les stipulations suivantes :

- une clé est une séquence de caractères graphiques, c'est à dire ne contenant ni espace, ni tabulation, ni fin de ligne (une clé tient donc sur une seule ligne)
- une valeur est une séquence arbitraire de lignes de texte

En vous inspirant des protocoles SMTP et POP, proposez un protocole pour communiquer avec votre serveur. Décrivez les opérations supportées par celui-ci. Présentez les cas d'utilisations sous la forme de scenarii donnant chacun une séquence de requètes et de réponses échangées entre client et serveur.

[1] en fin de TD1, on commencera à vous expliquer ce que vous devrez faire dans le TD2