

# Programmation Réseau Les Sockets



**Filière Électronique**  
**Semestre 7**

## Table des matières

<b>1</b>	<b>Objectifs du TP</b>	<b>3</b>
<b>2</b>	<b>Utilisation des commandes UNIX</b>	<b>3</b>
2.1	Rappels sur la relation client/serveur . . . . .	3
2.2	Utilisation des commandes d'analyse réseau . . . . .	4
2.2.1	Commande netstat . . . . .	4
2.2.2	Commande telnet . . . . .	4
<b>3</b>	<b>Programmation réseau par sockets</b>	<b>5</b>
3.1	Principe général . . . . .	5
3.2	Appels système principaux . . . . .	5
3.3	Calculatrice TCP client/serveur . . . . .	5
3.3.1	Architecture logicielle . . . . .	5
3.3.2	Serveur TCP . . . . .	5
3.3.3	Client TCP . . . . .	6
3.4	Résultats expérimentaux . . . . .	6
<b>4</b>	<b>Analyse et discussion</b>	<b>7</b>
<b>5</b>	<b>Conclusion</b>	<b>7</b>
<b>A</b>	<b>Annexes</b>	<b>8</b>
A.1	Code complet . . . . .	8
A.2	Terminal pour les essais de calculs . . . . .	8

# 1 Objectifs du TP

L'objectif principal de ce travail pratique est de comprendre et de mettre en œuvre les mécanismes de communication réseau entre processus, en utilisant les **sockets** du langage C sous UNIX.

Le TP s'articule autour de deux parties :

1. L'étude et la manipulation des commandes réseau sous UNIX afin d'observer les communications et d'analyser le comportement des protocoles TCP et UDP.
2. Le développement d'un projet applicatif complet mettant en œuvre le modèle client/serveur : une **calculatrice réseau** fondée sur le protocole TCP.

Les objectifs de ce projet sont de comprendre :

- la notion de port, d'adresse IP et de socket ;
- les différences entre TCP et UDP ;
- la mise en place d'un serveur concurrent capable de gérer plusieurs clients ;
- la validation du bon fonctionnement à l'aide d'outils réseau.

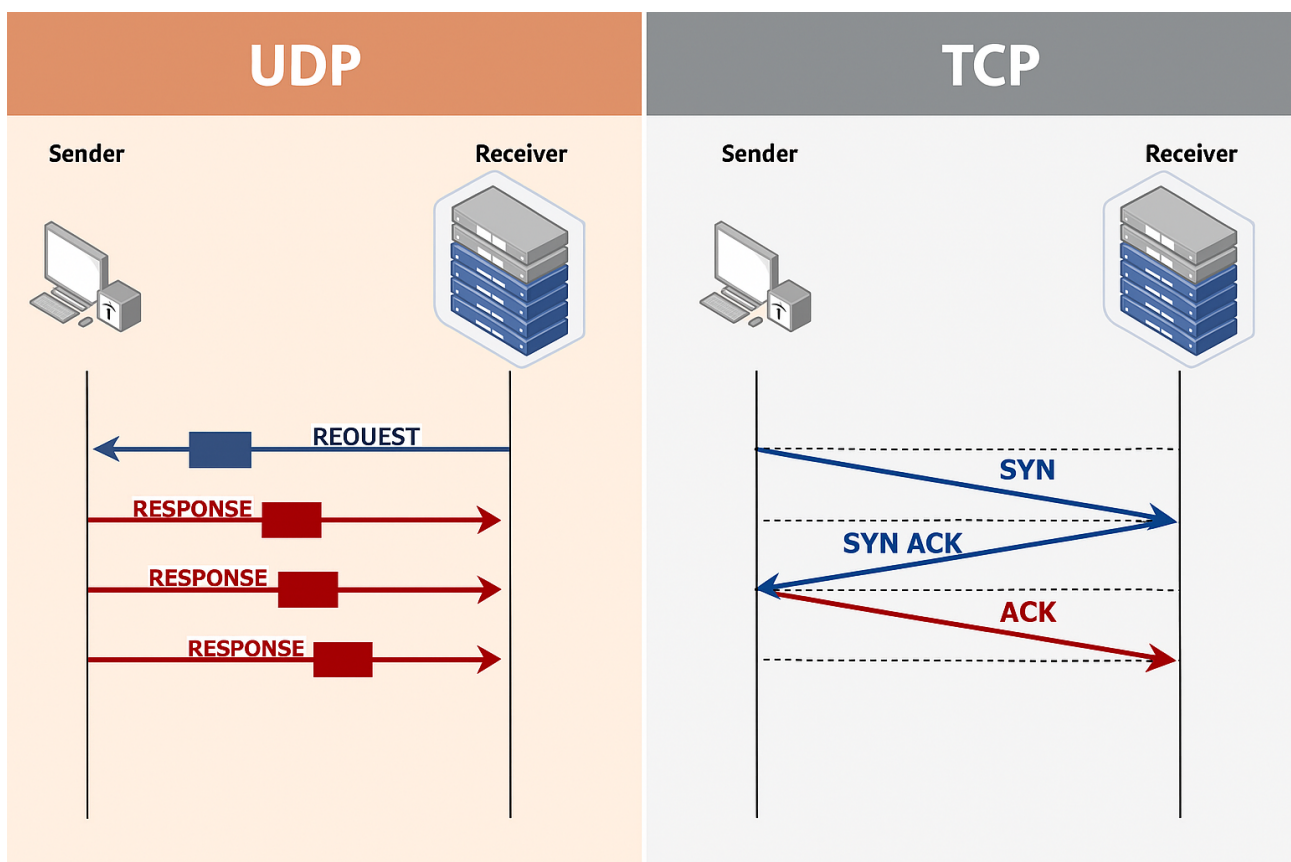
## 2 Utilisation des commandes UNIX

### 2.1 Rappels sur la relation client/serveur

Dans une architecture réseau, un **client** est une application qui initie une communication afin d'obtenir un service ou une ressource. Le **serveur** est une application qui écoute les demandes entrantes et y répond. Les deux programmes communiquent via un ensemble de règles appelées **protocoles de communication**.

Deux protocoles dominent la couche transport :

- **TCP (Transmission Control Protocol)** : orienté connexion, fiable, avec retransmission et contrôle de flux.
- **UDP (User Datagram Protocol)** : sans connexion, moins fiable, plus rapide, utilisé pour les transmissions légères ou temps réel.



## 2.2 Utilisation des commandes d'analyse réseau

### 2.2.1 Commande netstat

Le rôle de la commande netstat est d'afficher des informations sur les connexions réseaux (entrantes et sortantes). La commande : netstat -nr permet d'afficher la table de routage du noyau sous Unix. Elle fournit des informations sur les routes configurées sur le système pour diriger le trafic réseau.

Dans notre cas nous obtenons ce résultat :

```
1 ssh:~ > netstat -nr
2 Kernel IP routing table
3 Destination      Gateway           Genmask          Flags   MSS Window  irtt Iface
4 0.0.0.0           10.21.19.254     0.0.0.0          UG        0 0        0 ens192
5 10.7.0.0          0.0.0.0          255.255.248.0    U        0 0        0 ens224
6 10.21.18.0        0.0.0.0          255.255.254.0    U        0 0        0 ens192
```

Listing 1 – Table de routage

Cette table montre comment le noyau Linux choisit la route pour chaque paquet sortant selon l'adresse IP de destination.

```
1 ssh:~ > netstat -a
2 Active Internet connections (servers and established)
3 Proto Recv-Q Send-Q Local Address           Foreign Address         State
4 tcp      0      0 0.0.0.0:36393           0.0.0.0:*               LISTEN
5 tcp      0      0 0.0.0.0:nrpe           0.0.0.0:*               LISTEN
6
7 ...
8
9 tcp6     0      0 [::]:35715             [::]:*                  LISTEN
10 tcp6     0      0 localhost:ipp           [::]:*                  LISTEN
11
12 ...
13
14 udp6     0      0 [::]:41190             [::]:*
15 udp6     0      0 [::]:mdns               [::]:*
16 Active UNIX domain sockets (servers and established)
17 Proto RefCnt Flags   Type       State       I-Node Path
18 unix  3      [ ]      STREAM    CONNECTED  9507411
19 unix  3      [ ]      STREAM    CONNECTED  630083
20 unix  2      [ ACC ]      STREAM    LISTENING  621763 /var/run/ns1cd/socket
21
22 ...
23
24 unix  2      [ ]      DGRAM     CONNECTED  622766
25 unix  3      [ ]      STREAM    CONNECTED  622777
```

Listing 2 – Ports à l'écoute

Une autre commande utile est la commande netstat -a qui affiche toutes les connexions réseaux actives et les ports TCP et UDP que l'ordinateur écoute.

Les connexions LISTEN indiquent des serveurs en attente de connexions TCP. Les sockets de type STREAM utilisent TCP, tandis que les DGRAM utilisent UDP.

### 2.2.2 Commande telnet

telnet permet de tester une connexion TCP manuellement :

```
1 ssh:~ > telnet brahmane.enseirb.fr 21
2 Trying 10.21.18.113...
3 Connected to brahmane.enseirb.fr.
4 Escape character is '^]'.
5 220 INetSim FTP Service ready.
```

Listing 3 – Commande telnet

Ici, on établit une connexion avec un serveur FTP distant (port 21), ce qui valide la communication TCP.

### 3 Programmation réseau par sockets

#### 3.1 Principe général

Une **socket** est une interface logicielle entre l'application et le réseau. Chaque socket est identifiée par un couple (adresse IP, port).

Le protocole TCP est dit *orienté connexion* : avant tout échange de données, le client et le serveur établissent une session. Une fois la communication terminée, la socket est fermée.

#### 3.2 Appels système principaux

**Côté serveur :**

```
1 socket() -> bind() -> listen() -> accept() -> read()/write() -> close()
```

**Côté client :**

```
1 socket() -> connect() -> write()/read() -> close()
```

#### 3.3 Calculatrice TCP client/serveur

L'objectif est d'implémenter un serveur TCP capable d'effectuer des opérations arithmétiques pour plusieurs clients connectés simultanément. Chaque client envoie une requête du type :

$$a \text{ op } b$$

où op peut être +, -, \*, ou /. Le serveur calcule le résultat et le renvoie au client.

##### 3.3.1 Architecture logicielle

- **serveur.c** : crée une socket TCP, écoute sur un port, accepte les connexions, traite les requêtes.
- **client.c** : crée une socket, se connecte au serveur, envoie des expressions et affiche les résultats.

##### 3.3.2 Serveur TCP

Le serveur gère la concurrence avec `fork()` : chaque client est traité dans un processus fils.

```
1 double calculatrice(char* requete) {
2     double nb1;
3     double nb2;
4     char op;
5     // Lecture de la requete envoyee par le client
6     sscanf(requete, "%lf %c %lf", &nb1, &op, &nb2);
7
8     // Verifications des bornes
9     if ((nb1 > 10000.) || (nb1 < 0.)) {
10         return -1; // erreur : nb1 hors limite
11     }
12     else if ((nb2 > 10000.) || (nb2 < 0.)) {
13         return -1; // erreur : nb2 hors limite
14     }
15     // Selection de l operation
16     else if (op == '+') {
17         return nb1 + nb2;
18     }
19     else if (op == '-') {
20         return nb1 - nb2;
21     }
22     else if (op == '*') {
23         return nb1 * nb2;
24     }
25     else if (op == '/') {
26         return nb1 / nb2;
27     }
28     else {
29         return -2; // operateur inconnu
30     }
31 }
```

Listing 4 – Extrait du serveur TCP

Le serveur retourne des messages d'erreur explicites :

- -1 : valeurs hors limites;
- -2 : format invalide.

### 3.3.3 Client TCP

```

1 while(1){
2     /* Etablissement de la connexion avec le serveur ftp */
3     printf("\n");
4     printf("Calcul : ");
5     fgets(calcul,256,stdin);
6     if(calcul[0]!='quit'){
7         exit(0);
8     }
9
10    /* Creation de la socket TCP */
11    if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
12        printf("Probleme lors de la creation de socket\n");
13        exit(1);
14    }
15
16    /* Etablissement de la connexion avec le serveur ftp */
17    if((connect(sd, (struct sockaddr *)&sa, sizeof(sa))) < 0) {
18        printf("Probleme de connexion avec le serveur\n");
19        exit(1);
20    }
21
22    /* Envoi de la commande ftp vers serveur ftp */
23    write(sd, calcul, strlen(calcul));
24
25    /* Lecture de la reponse du serveur ftp */
26    int n = read(sd, buffer, sizeof(char)*256);
27
28    write(1,buffer,n);
29    close(sd);
30 }
```

Listing 5 – Client TCP - Communication

## 3.4 Résultats expérimentaux

```

1 ssh:~/Documents/S7/RE223 > ./cur tcp_client
2 Compilation de tcp_client...
3 ssh:~/Documents/S7/RE223 > ./tcp_client 10.7.7.185 24000
4
5 Equation : 2+2
6 4.000000
7
8 Equation : 2/0
9 inf
10
11 Equation : 3*9
12 27.000000
13
14 Equation : 0/0
15 -nan
16
17 Equation : 99999+3
18 [Serveur]: Out of range...
19
20 Equation : quit
21 ssh:~/Documents/S7/RE223 >
```

Listing 6 – Tests de divers calculs

Le -nan (Not a Number) est une valeur spéciale renvoyée quand une opération mathématique est indéfinie, comme  $\frac{0}{0}$ .

Le out of range apparait car le resultat de l'opération ne peut dépasser 10000 (cahier des charges).

Le resultat inf vient simplement du fait que la division par 0 tend vers  $\pm\infty$ .

## 4 Analyse et discussion

Ce projet illustre l'importance du protocole TCP pour la fiabilité des communications. L'emploi de sockets permet de contrôler directement la logique réseau au niveau applicatif. L'approche par processus (fork) offre la simplicité, mais au prix d'une consommation mémoire importante. Pour un serveur à haute charge, une architecture à base de threads ou de multiplexage (select(), poll()) serait plus efficace.

Les principaux défis rencontrés ont concerné :

- la gestion des erreurs de connexion ;
- la synchronisation entre processus clients ;
- la validation du format des requêtes.

## 5 Conclusion

Cette pratique de programmation réseau nous a aidés à mieux saisir le fonctionnement des couches de transport et leur rôle crucial dans la communication entre processus. En utilisant des outils UNIX comme netstat et telnet, nous avons été en mesure d'observer les communications réseau et d'étudier le fonctionnement du protocole TCP.

Le développement d'une application client/serveur en C a renforcé notre compréhension des sockets et de la gestion de la communication bidirectionnelle. Ce travail pratique nous a aussi exposés à des enjeux concrets, tels que la gestion des erreurs, la compétition entre processus ou encore le traitement de requêtes non valides.

Pour l'avenir, diverses améliorations sont possibles : l'élaboration d'un protocole applicatif plus exhaustif ou l'intégration d'une interface utilisateur pour rendre l'application plus intuitive. Ainsi, ce projet offre une excellente initiation à la programmation réseau et à la création d'applications distribuées solides et fiables.

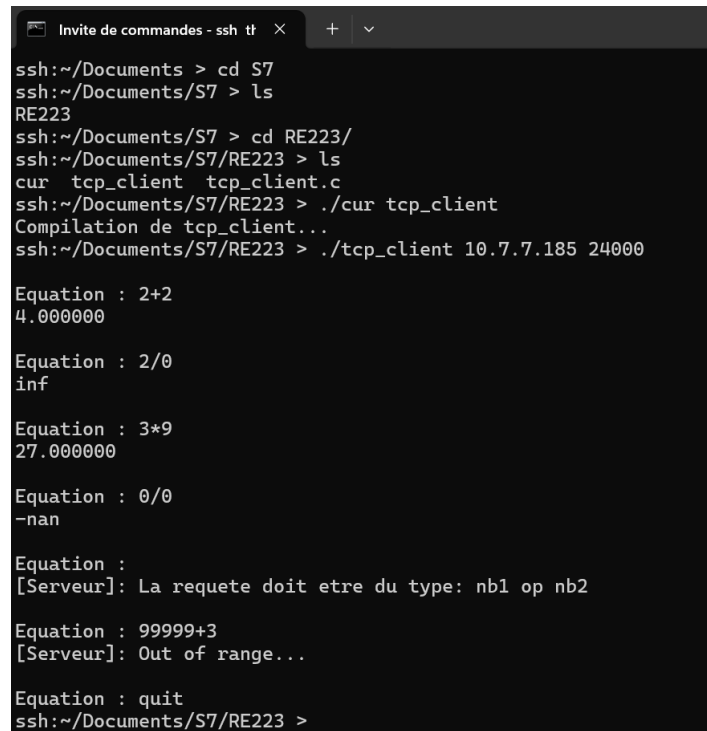
## A Annexes

---

### A.1 Code complet

[Projet Réseau RE223 - Codes](#)

### A.2 Terminal pour les essais de calculs



```
Invite de commandes - ssh tt x + v
ssh:~/Documents > cd S7
ssh:~/Documents/S7 > ls
RE223
ssh:~/Documents/S7 > cd RE223/
ssh:~/Documents/S7/RE223 > ls
cur tcp_client tcp_client.c
ssh:~/Documents/S7/RE223 > ./cur tcp_client
Compilation de tcp_client...
ssh:~/Documents/S7/RE223 > ./tcp_client 10.7.7.185 24000

Equation : 2+2
4.000000

Equation : 2/0
inf

Equation : 3*9
27.000000

Equation : 0/0
-nan

Equation :
[Serveur]: La requete doit etre du type: nb1 op nb2

Equation : 99999+3
[Serveur]: Out of range...

Equation : quit
ssh:~/Documents/S7/RE223 >
```