# TP1

9 octobre 2022

## Table des matières

# 1 Client-serveur UDP

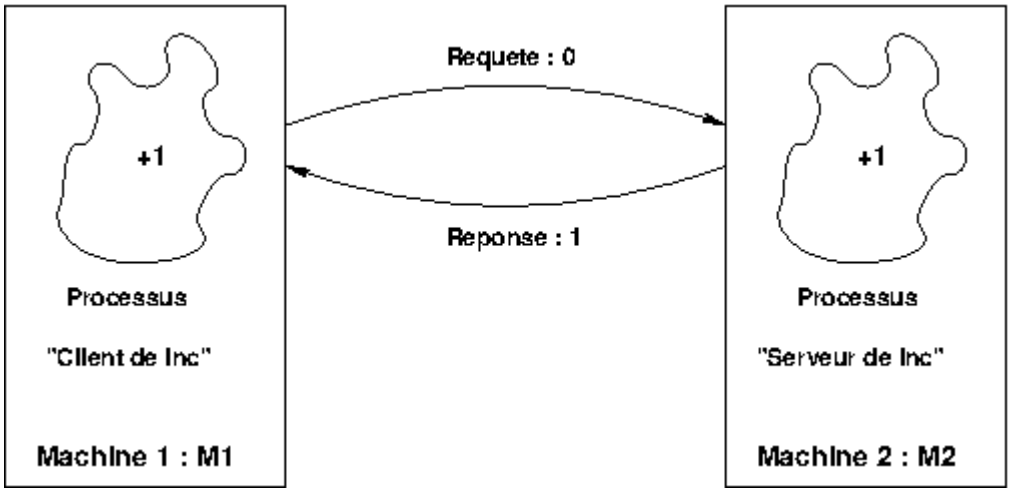
Le travail qui vous est proposé dans cette section aborde l'étude des mécanismes relatifs à la communication de processus par l'intermédiaire de sockets de type **SOCK_DGRAM** .

Vous êtes invité à vous remémorer les caractéristiques induites par ce choix => cf le cours : communication par datagrammes !

## 1.1 CS Incrémentation

On commence par un premier exemple de programme client-serveur en UDP.

Soit deux processus : un client et un serveur.




- ➢ La requête (du client) est un entier $\in [0, 99]$.
- ➢ La réponse (du serveur) est cet entier incrémenté de 1 modulo 100 (pour rester $\in [0, 99]$).

Un première version de ce programme est fournie dans le fichier `inc.c` .

Il s'agit d'une version :

- ✔ **itérative** car le client et le serveur bouclent ... à l'infini.
  L'un pour soumettre un succession de requêtes et l'autre de réponses.
- ✔ et **séquentielle** car les requêtes sont traitées par le serveur les unes après les autres.
  Le traitement n'engendre pas de nouveau processus !

C'est un choix architecturale simpliste qui pourrait poser de gros problèmes de performances si le service devenait plus complexe. Pour l'instant on s'en contente, et pour simplifier encore plus votre travail on utilise le même programme en tant que client et en tant que serveur.

### 1.1.1 TODO : Votre travail

(a) Analyser, comprendre

(b) Expérimenter sur deux machines **différentes** (une pour le serveur et une pour le client).

Pour voir les machines de l'UCA disponibles et utilisables sur Nice (bat "Petit Valrose") vous pouvez consulter la page du CRIPS (cliquez ensuite sur le lien indiqué !) :

`http://crips.unice.fr/plan_etage.html`

et utiliser les machines du niveau 3 (Linux).

Vous notez qu'au lancement du programme il faut fournir en argument le nom de la machine distante.

Si vous ne prenez pas deux machines différentes, cela ne fonctionnera PAS puisque client et serveur voudraient utiliser le même port !

Les questions qui suivent doivent "alimenter" votre réflexion :

(a) On peut remarquer que les deux processus (client et serveur) issus du même programme commencent leur itération par un `sendto()` :
Pourquoi cela fonctionne t-il, notamment au regard de ce qui est expliqué dans le cours dans la section "organisation d'un client/serveur UDP" ?

(b) Si on considère que le serveur est le processus qui réalisera le premier recvfrom(), quel est-il ?

(c) Pourquoi travailler avec la représentation ASCII d'un entier (atoi() et sprinf()) plutôt qu'avec la représentation binaire de l'entier sur 4 octets ?

(d) Modifier le code et plus précisément l'appel de la primitive recvfrom() pour récupérer l'adresse de la machine dont on reçoit le datagramme.

(e) Au final, quelles sont les informations que l'on peut extraire de l'appel de la fonction recvfrom() et donc du datagramme que l'on vient de recevoir ?

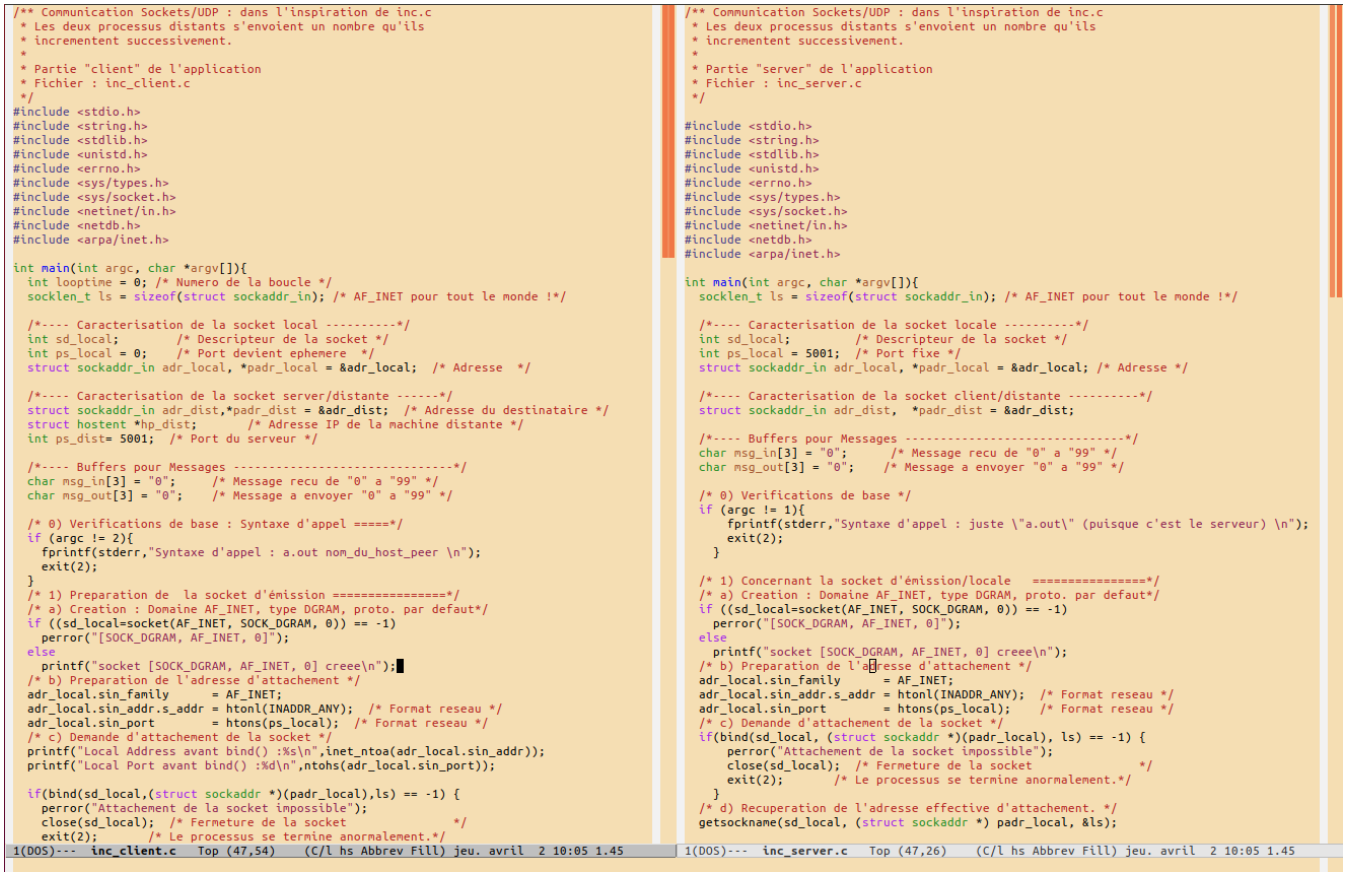
(f) Faire afficher à l'écran le nom de la machine qui a expédiée le datagramme à l'aide de la fonction `inet_ntoa()`

(g) Est-ce que le dialogue entre deux processus peut se bloquer ? Pourquoi ?

# 2 Evolutions

## 2.1 Séparation des codes Client et Serveur

(a) Vous "découpez" `inc.c` en deux fichiers qui feront apparaître le client (fichier `inc_client.c` ) et le serveur (fichier `inc_server.c` .



```c
/** Communication Sockets/UDP : dans l'inspiration de inc.c
 * Les deux processus distants s'envoient un nombre qu'ils
 * incrementent successivement.
 *
 * Partie "client" de l'application
 * Fichier : inc_client.c
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]){
  int looptime = 0; /* Numero de la boucle */
  socklen_t ls = sizeof(struct sockaddr_in); /* AF_INET pour tout le monde !*/

  /*---- Caracterisation de la socket local ----------*/
  int sd_local;        /* Descripteur de la socket */
  int ps_local = 0;    /* Port devient ephemere  */
  struct sockaddr_in adr_local, *padr_local = &adr_local;  /* Adresse  */

  /*---- Caracterisation de la socket server/distante ------*/
  struct sockaddr_in adr_dist,*padr_dist = &adr_dist;  /* Adresse du destinataire */
  struct hostent *hp_dist;       /* Adresse IP de la machine distante */
  int ps_dist= 5001;  /* Port du serveur */

  /*---- Buffers pour Messages ------------------------------*/
  char msg_in[3] = "0";     /* Message recu de "0" a "99" */
  char msg_out[3] = "0";    /* Message a envoyer "0" a "99" */

  /* 0) Verifications de base : Syntaxe d'appel =====*/
  if (argc != 2){
    fprintf(stderr,"Syntaxe d'appel : a.out nom_du_host_peer \n");
    exit(2);
  }
  /* 1) Preparation de  la socket d'émission ================*/
  /* a) Creation : Domaine AF_INET, type DGRAM, proto. par defaut*/
  if ((sd_local=socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    perror("[SOCK_DGRAM, AF_INET, 0]");
  else
    printf("socket [SOCK_DGRAM, AF_INET, 0] creee\n");
  /* b) Preparation de l'adresse d'attachement */
  adr_local.sin_family      = AF_INET;
  adr_local.sin_addr.s_addr = htonl(INADDR_ANY);  /* Format reseau */
  adr_local.sin_port        = htons(ps_local);  /* Format reseau */
  /* c) Demande d'attachement de la socket */
  printf("Local Address avant bind() :%s\n",inet_ntoa(adr_local.sin_addr));
  printf("Local Port avant bind() :%d\n",ntohs(adr_local.sin_port));

  if(bind(sd_local,(struct sockaddr *)(padr_local),ls) == -1) {
    perror("Attachement de la socket impossible");
    close(sd_local);  /* Fermeture de la socket              */
    exit(2);        /* Le processus se termine anormalement.*/
  }
  /* d) Recuperation de l'adresse effective d'attachement. */
  getsockname(sd_local, (struct sockaddr *) padr_local, &ls);
```

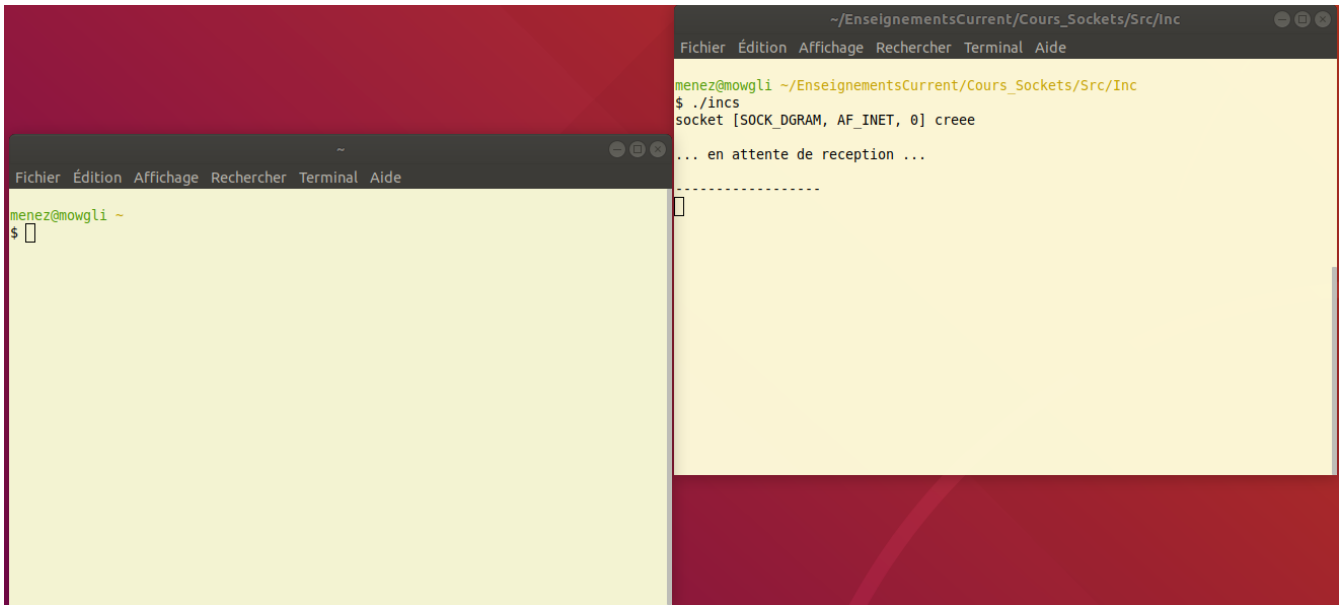
1(DOS)--- inc_client.c Top (47,54) (C/l hs Abbrev Fill) jeu. avril 2 10:05 1.45

```c
/** Communication Sockets/UDP : dans l'inspiration de inc.c
 * Les deux processus distants s'envoient un nombre qu'ils
 * incrementent successivement.
 *
 * Partie "server" de l'application
 * Fichier : inc_server.c
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]){
  socklen_t ls = sizeof(struct sockaddr_in); /* AF_INET pour tout le monde !*/

  /*---- Caracterisation de la socket locale ----------*/
  int sd_local;         /* Descripteur de la socket */
  int ps_local = 5001;  /* Port fixe */
  struct sockaddr_in adr_local, *padr_local = &adr_local; /* Adresse */

  /*---- Caracterisation de la socket client/distante ----------*/
  struct sockaddr_in adr_dist,  *padr_dist = &adr_dist;

  /*---- Buffers pour Messages ------------------------------*/
  char msg_in[3] = "0";      /* Message recu de "0" a "99" */
  char msg_out[3] = "0";    /* Message a envoyer "0" a "99" */

  /* 0) Verifications de base */
  if (argc != 1){
      fprintf(stderr,"Syntaxe d'appel : juste \"a.out\" (puisque c'est le serveur) \n");
      exit(2);
   }

  /* 1) Concernant la socket d'émission/locale   ================*/
  /* a) Creation : Domaine AF_INET, type DGRAM, proto. par defaut*/
  if ((sd_local=socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    perror("[SOCK_DGRAM, AF_INET, 0]");
  else
    printf("socket [SOCK_DGRAM, AF_INET, 0] creee\n");
  /* b) Preparation de l'adresse d'attachement */
  adr_local.sin_family      = AF_INET;
  adr_local.sin_addr.s_addr = htonl(INADDR_ANY);  /* Format reseau */
  adr_local.sin_port        = htons(ps_local);    /* Format reseau */
  /* c) Demande d'attachement de la socket */
  if(bind(sd_local, (struct sockaddr *)(padr_local), ls) == -1) {
      perror("Attachement de la socket impossible");
      close(sd_local);  /* Fermeture de la socket              */
      exit(2);        /* Le processus se termine anormalement.*/
   }
  /* d) Recuperation de l'adresse effective d'attachement. */
  getsockname(sd_local, (struct sockaddr *) padr_local, &ls);
```

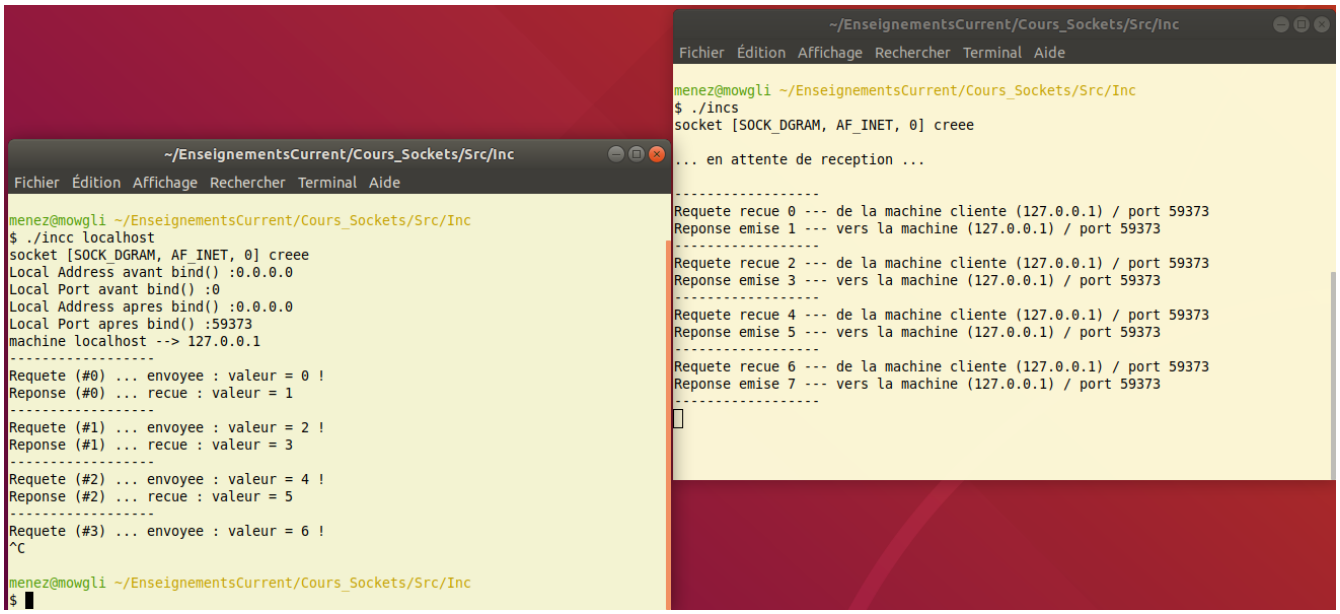
1(DOS)--- inc_server.c Top (47,26) (C/l hs Abbrev Fill) jeu. avril 2 10:05 1.45

(b) On peut désormais les essayer en local (localhost) puisque client et serveur **n'utilisent plus le même port !**

Vous commencez par lancer le serveur qui se bloque :




Puis le client (en précisant l'adresse du serveur) :




Vous remarquez les "log" qui s'affichent dans la console pour savoir qui émet, d'où . . . tout ce qui permet de piloter le déroulement du dialogue !

➢ Ajouter cela à votre code !

## 2.2 Un truc à essayer et à savoir

Moi j'aimerais bien essayer mon client-serveur entre chez moi (machine "mowgli") et l'UCA (machine "polymnie"). Mais je sais que la connexion sur polymnie est protégée.

Je vous propose d'utiliser `ssh` pour résoudre ce problème :

➢ Si on utilisait un flux TCP, on pourrait plus simplement créer un **tunnel** entre le port 5001 de ma machine locale (port sur lequel je pense trouver le serveur) et le port 5001 de la machine polymnie avec la commande

```
ssh -L 5001:polymnie.unice.fr:5001 menez@polymnie.unice.fr
```

`http://www.linux-france.org/prj/edu/archinet/systeme/ch13s04.html`

Mais on utilise un flux UDP . . . qui ne passe pas sur ce style de tunnel.

On va donc utiliser une manipulation issue de

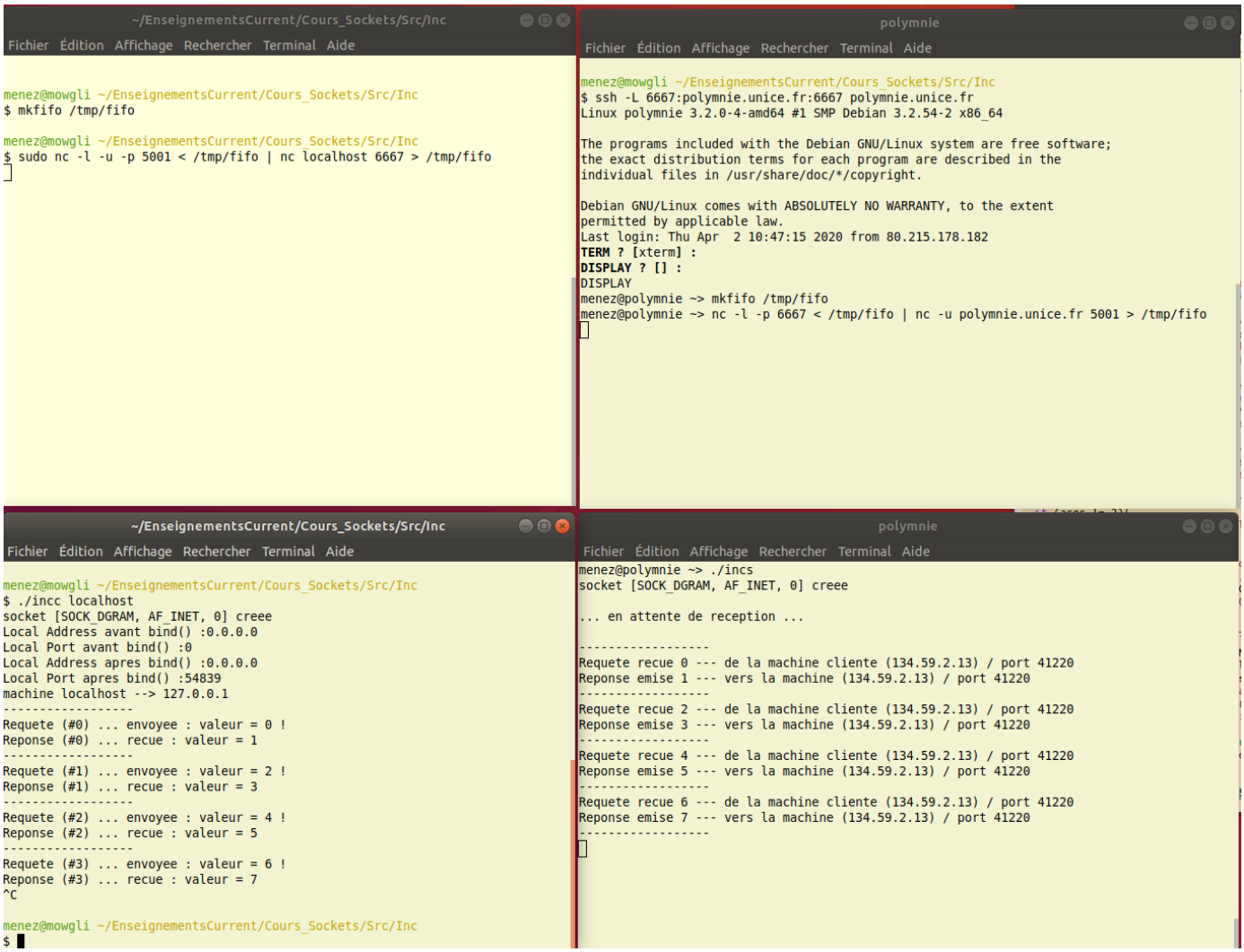
`http://zarb.org/~gc/html/udp-in-ssh-tunneling.html`

Avant de commencer, on fait l'hypothèse que vous avez compilé le serveur "inc_server.c" sur polymnie !

Vous n'oubliez pas non plus que si on est plusieurs à utiliser polymnie, **il faudra adapter les noms et les numéros de port pour éviter qu'il y ait "collision" comme pour les sémaphores** !

Il va vous falloir 4 terminaux mais heureusement c'est gratuit !

➢ Sur la figure qui suit les deux terminaux de droite sont sur polymnie.




① Les deux terminaux du haut préparent la liaison entre ma machine locale et polymnie.

➢ Ca commence par celui de **droite** (haut) qui est initialement en local.

✔ On crée un tunnel qui sera capable d'acheminer un flux TCP entre le local et polymnie.

J'utilise le port 6667 . . . vous choisirez le vôtre . . . sinon bim sur polymnie !

✔ On est désormais sur polymnie et on crée une FIFO .. attention au risque de collision sur le nom.

✔ La commande en "nc" (netcat) permet de forwarder le trafic TCP du port 6667 sur le port 5001 en UDP.

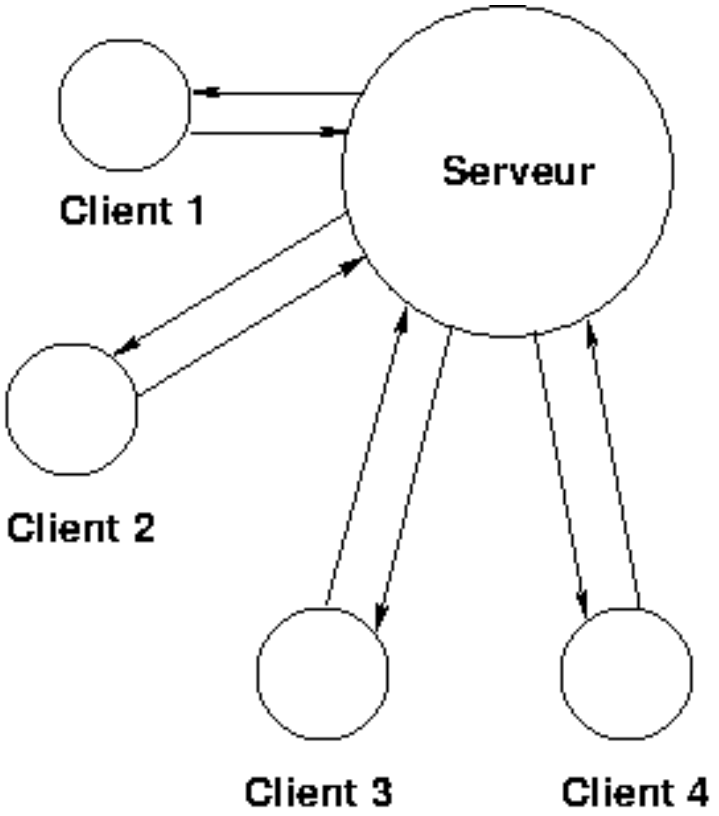
Fort comme commande . . . non ?

➢ Ensuite sur le terminal haut gauche (donc en local), on fait en sorte que le trafic UDP arrivant sur le port 5001 s'écoule en TCP sur le port local 6667 . . . qui est tunnelé avec le port 6667 de polymnie.

② Lorsque tout ceci est fait, les deux terminaux du bas montrent que l'on peut mettre en oeuvre le TP !

## 2.3 Un serveur et plusieurs clients

Toujours dans le contexte du client-serveur d'incrémentation, il s'agit de mettre en place une expérimentation de type : —— **un serveur Vs plusieurs clients** ———




Il y a peu de chose à faire pour que cela fonctionne :

(a) Modifier le code pour que le serveur réponde aux émetteurs des datagrammes qu'il reçoit.

Il faut que le serveur sache exploiter l'information contenu dans chaque datagramme reçu : cf recvfrom() !

(b) Montrer au niveau des consoles/terminaux associés à chaque client que le serveur gère correctement les flux . . . de chaque client.

Sur les consoles des clients et des serveurs, on veut voir aussi s'afficher les adresses et les ports des interlocuteurs : qui envoie , qui reçoit, . . .

# 3 Un client-serveur de calcul en UDP

Pour cet exercice faire un "petit" serveur de calcul :

(a) Au lieu d'envoyer "seulement" un entier au serveur, on lui envoie une expression mathématique élémentaire de type :

"A op B"

avec A et B entiers et op $= \{+, -, *, /\}$

**Attention !**

➢ Vous êtes décideur de la forme qu'aura le message transmis au serveur.

Vous pouvez imposer des espaces entre les tokens si cela vous facilite l'analyse de la chaîne de caractères (c'est à dire le message). On peut s'en passer tellement les messages sont simples, mais sinon remember strtok() au premier semestre !

Deux solutions parmi (sans doute) d'autres :

① Considérer que le message est la chaîne de caractères formant l'expression élémentaire.
Par exemple : "2 + 3"

② Considérer que le message a une structure fixe ; un opérateur suivi des deux opérandes.
Par exemple : "+ 2 3"

➢ Le serveur doit répondre en fournissant la valeur calculée.

➢ **Pas de parenthèse** dans l'expression pour l'instant !

# 4 Un serveur concurrent en UDP

On souhaite introduire dans le serveur de calcul la possibilité, pour un client, de définir et d'utiliser des variables.

Par exemple :

➢ "= x 3" renverrait 3 (pour signifier que la variable a bien été créée).

➢ "x * 2" renverrait 6 (ou "error" si x n'existe pas ?")

On cherchera à ne pas trop complexifier la syntaxe des expressions . . . car cela n'apporterait rien d'un point de vue "réseau".

Par contre, **on prendra soin de bien différencier les variables de chaque client** (notamment si elles ont le même identificateur) :

Pour gérer cette difficulté, on pourrait, au niveau du serveur :

➢ Mettre en place par une table des symboles, qui tiendrait compte pour chaque symbole de l'identification du client qui lui est associé.

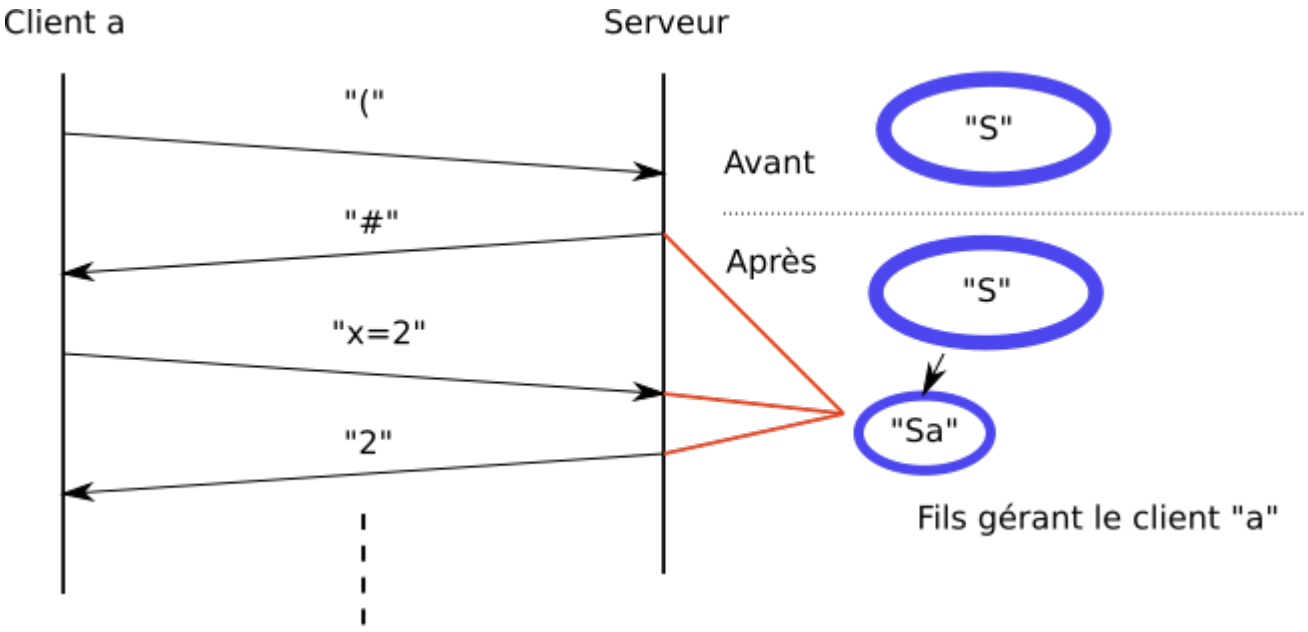
**Cette solution est écartée !** parce que l'on souhaite mettre en place un serveur concurrent.

➢ En effet, plutôt que d'associer à chaque symbole un client, on va utiliser un serveur concurrent où chaque client pourra "ouvrir" une session de calcul dans laquelle il pourra gérer "ses" variables.

D'un point de vue du serveur, cela nécessite de "forker" mais du coup, chaque fils est responsable d'un client et donc on n'a pas à gérer au niveau du père/serveur la complexité de l'association variable/client.

- ✔ Une session est ouverte par une "(" et se termine par une ")".
- ✔ On ne peut pas utiliser de variables sans avoir au préalable ouvert une session.
- ✔ Pour faire simple (et ne pas avoir à gérer une véritable des symboles ), on ne peut avoir qu'une seule variable par client et elle s'appelle "x" !

Cela devrait donner un début de dialogue comme suit :




## 5 Perte de datagrammes ?

UDP ne garantit pas la livraison de chaque datagramme.

Pour constater/provoquer des pertes on peut essayer :

① Soit de "saturer" le serveur en l'inondant de requètes. Vous pouvez vous mettre à plusieurs clients !

Vous pouvez aussi "ralentir" le serveur . . . un `sleep()` ?

② Soit de travailler sur un réseau plus incertain. Il faudrait essayer avec des clients situés sur des ISP (free, orange, ...) : vos machines à la maison ?

Comment va se manifester une perte ?