Université de Bretagne Occidentale

TD Réseaux
Introduction au client-serveur en TCP

Licence d'informatique
Université de Bretagne Occidentale

1 Fonctions `gethostname`, `gethostbyname` et `inet_ntoa`

1. A l'aide de la fonction `gethostname`, construire un programme qui permet d'afficher le nom de votre machine.
2. A l'aide de la fonction `gethostbyname`, construire un programme qui permet de récupérer l'adresse IP de votre machine et de l'afficher sous forme d'un entier 32 bits et en notation pointée (`inet_ntoa`).

2 Un échange avec TCP

2.1 Le point de vue du serveur

2.1.1 Rappel du cours

Le rôle du serveur est passif pendant l'établissement de la connexion :
— le serveur doit disposer d'une *socket d'écoute* ou *socket de rendez-vous* (par la primitive `socket`) attachée au port TCP correspondant au service (par la primitive `bind`), ce port est supposé connu des clients.
— le serveur doit aviser le système auquel il appartient qu'il est prêt à accepter les demandes de connexion, sur la socket de rendez-vous (par la primitive `listen`).
— puis le serveur se met en attente de connexion sur la socket de rendez-vous (par la primitive **bloquante** `accept`).
— lorsqu'un client prend l'initiative de la connexion, le processus serveur est **débloqué** et une nouvelle socket, appelée *socket de service*, est créée :
c'est cette socket de service qui est connectée à la socket du client.
— le serveur peut alors déléguer le travail à un nouveau processus créé par `fork` et reprendre son attente sur la socket de rendez-vous,
ou traiter lui-même la demande (mais il risque de faire attendre des nouveaux clients).

2.1.2 Travail à réaliser

Compléter le code ci-dessous en suivant les instructions des commentaires. Le client sera traité directement par le serveur (pas de création d'un nouveau processus).

```
/* Lancement d'un serveur :  serveur_tcp port */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
```

```
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TRUE 1

main (int argc, char **argv)
{
int socket_RV, socket_service;
char buf[256];
struct sockaddr_in adresseRV;
int lgadresseRV;
struct sockaddr_in adresseClient;
int lg_adresseClient;
unsigned short port;

/* creation de la socket de RV */

/* preparation de l'adresse locale */

/* attachement de la socket a l'adresse locale */

/* declaration d'ouverture du service */

/* boucle d'attente de connexion */
while(TRUE)
  {
      printf("Debut de boucle\n");
      fflush(stdout);

      /* attente d'un client */

      /* traitement des erreurs */

      /* un client est arrive */
      printf("connexion acceptee\n");
      fflush(stdout);

      /* lecture dans la socket d'une chaine de caractères */

      printf("j'ai recu %s\n", buf);
      fflush(stdout);

      /* ecriture dans la socket */
      strcpy(buf, "tchao");

      printf("reponse envoyee ...adios\n");
      close(socket_service);
  }
}
```

2.2 Le point de vue du client

2.2.1 Rappel du cours

Le rôle du client est actif pendant l'établissement de la connexion :

— le client doit disposer d'une socket attachée à un port TCP quelconque (par la primitive `socket`).

— le client doit construire l'adresse du serveur, dont il doit connaître le nom ou l'adresse IP et le numéro de port de la socket de rendez-vous, (il obtient éventuellement l'adresse IP du serveur à partir de son nom par la primitive `gethostbyname`).

— puis le client demande la connexion de sa socket locale à la socket de rendez-vous du serveur (par la primitive **bloquante** `connect`).

— lorsque le serveur accepte la connexion, le processus client est **débloqué** et peut dialoguer avec le serveur.

2.2.2 Travail à réaliser

Compléter le code ci-dessous en suivant les instructions des commentaires.

```
/* Lancement d'un client :  client_tcp port machineServeur */

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

main (int argc, char **argv)
{
int sock;
char buf[256];
struct sockaddr_in adresse_serveur;
struct hostent *hote;
unsigned short port;

/* creation de la socket locale */

/* recuperation de l'adresse IP du serveur (a partir de son nom) */

/* preparation de l'adresse du serveur */

/* demande de connexion au serveur */

/* ecriture dans la socket */
strcpy(buf, "salut");

printf("salut envoye\n");
fflush(stdout);

/* lecture dans la socket d'une chaine */
```

```
printf("j'ai recu %s\n", buf);
fflush(stdout);
}
```

3 Copie de fichiers **(exercice préparatoire aux TD/TP à suivre)**

En utilisant `read` et `write`, écrire un programme qui recopie un fichier dans un autre par paquets de `BUFSIZE` octets (`BUFSIZE` étant macro-définie).
Le nom du fichier source est donné dans `argv[1]` et celui du fichier destination dans `argv[2]`.

TP Réseaux
Introduction aux réseaux et programmation TCP

Licence d'informatique
Université de Bretagne Occidentale

1 Fonctions `gethostname`, `gethostbyname` et `inet_ntoa`

1. A l'aide de la fonction `gethostname`, afficher le nom de votre machine.
2. A l'aide de la fonction `gethostbyname`, récupérer l'adresse IP de votre machine et afficher-la sous forme d'un entier 32 bits.
3. A l'aide de la fonction `inet_ntoa`, afficher l'adresse IP de votre machine en notation pointée.

Vous pouvez également tester les deux derniers programme avec par exemple les machines suivante : **vador.univ-brest.fr, www.google.fr** etc.

2 Copie de fichiers **(exercice préparatoire aux exercices suivants)**

En utilisant `read` et `write`, écrire un programme qui recopie un fichier dans un autre par paquets de `BUFSIZE` octets.

Le nom du fichier source est donné dans `argv[1]` et celui du fichier destination dans `argv[2]`.

A partir de maintenant, vous allez réaliser des échanges sous TCP entre un client et un serveur. Il est impératif que client et serveur s'exécutent sur deux machines différentes.

---

3 Un échange avec TCP

On désire mettre en œuvre l'exercice du TD. On cherchera à échanger une chaine de caractère quelconque ainsi qu'une valeur numérique en retour.

3.1 Le point de vue du serveur

Récupérer le fichier `serveur_tcp.c` du répertoire
`/home/commun_depinfo/enseignants/leparc/Reseaux/TCP_exo4`

Compléter les ... dans le code récupéré, en suivant les instructions des commentaires.

3.2 Le point de vue du client

Récupérer le fichier `client_tcp.c` du répertoire
`/home/commun_depinfo/enseignants/leparc/Reseaux/TCP_exo4`

Compléter les ... dans le code récupéré, en suivant les instructions des commentaires.

4 Un client-serveur avec TCP

On souhaite réaliser un système client-serveur TCP, qui permet de transférer des fichiers entre le client et le serveur.

- Lorsqu'un client réussi à se connecter, ce client émet un fichier puis se termine.
- Le serveur est en position d'attente d'un client. Lorsque le serveur accepte un client, le serveur reçoit un fichier puis revient attendre un nouveau client.

4.1 Construction des programmes client et serveur

- A partir du programme `client_tcp.c` réalisé dans l'exercice 3, écrire un programme `emetteur_TCP.c` qui lit (avec `read`) un fichier par paquets de `BUFSIZE` octets et l'écrit (avec `write`) dans une socket TCP. Après chaque envoi, pensez à afficher le nombre d'octets envoyés.
  N'oubliez pas ensuite de fermer la socket, ce qui provoquera la fin de la lecture côté serveur.
  Le numéro de port du serveur est donné dans `argv[1]`.
  Le nom de machine du serveur est donné dans `argv[2]`.
  Le nom du fichier à lire est donné dans `argv[3]`.
- A partir du programme `serveur_tcp.c` réalisé dans l'exercice 3, écrire un programme `receveur_TCP.c` qui :
  - boucle infinie sur l'attente de client,
  - une fois un client connecté, lit (avec `read`) dans la socket de dialogue connectée à la socket du client, des paquets de `BUFSIZE` octets et les écrit (avec `write`) dans un fichier. Après chaque lecture, pensez à afficher le nombre d'octets reçus.
    La lecture s'arrêtera, lorsque la socket sera fermée à l'initiative du client. La fermeture de la socket va entrainer la terminaison de la fonction `read` avec un code de retour égal à 0. Cette façon de faire empêche l'envoi d'un deuxième fichier et une solution permettant l'envoi de plusieurs fichiers sera présentée plus loin dans ce sujet.
    Le numéro de port du serveur est donné dans `argv[1]`.
    Le nom du fichier à écrire est donné dans `argv[2]`.

  Pour simplifier le problème, le serveur n'est lancé qu'avec un seul nom de fichier à écrire. Chaque nouveau transfert écrase donc le précédent.

4.2 Réalisation de test et analyse

Une fois vos programmes au point, testez l'envoi d'un fichier d'une taille d'environ 600 octets en définissant

- un buffer au niveau de l'émetteur de 30 octets et au niveau du récepteur de 120 octets. Expliquez alors le comportement de vos programmes.
- un buffer au niveau de l'émetteur de 150 octets et au niveau du récepteur de 50 octets. Expliquez alors le comportement de vos programmes.

4.3 Amélioration du programme serveur

Le programme `serveur_tcp.c` construit en 4.1 ne permet de recevoir qu'un seul client à la fois. Modifier le pour permettre la réception de plusieurs fichiers simultanément. On utilisera la commande `fork()` et le numéro du `pid` pour distinguer les fichiers recopiés.

5 Un transfert en un seul échange avec TCP

5.1 Test initial

Récupérer le fichier `serveur_tcp.c` du répertoire
`/home/commun_depinfo/enseignants/leparc/Reseaux/TCP_exo6`.

Récupérer le fichier `client_tcp.c` du répertoire
`/home/commun_depinfo/enseignants/leparc/Reseaux/TCP_exo6`.

Ces deux programmes permettent de transférer un fichier entre le client et le serveur au moyen d'un seul échange.

Transférez des fichiers de grande taille (100 000 octets environ). Vous pouvez les fabriquer avec la commande `cat unFichier unFichier >unFichierDeuxFoisPlusGros`). Vous pourrez constater que le `read` du serveur ne permet pas de lire tous les caractères envoyés par le `write` du client. Expliquez et déduisez-en la valeur de la charge utile d'un paquet et du MTU.

5.2 Un échange sûr avec TCP

Afin de pouvoir éliminer le problème de l'exercice précédent, modifiez les programmes de la manière suivante :

- dans `client_tcp.c`, envoyer la taille du message (contenu dans la variable `n`) avant d'envoyer le message. Le client se met en attente d'une valeur envoyée par le serveur pour indiquer la fin du transfert (et éventuellement la fermeture de la socket).
- dans `serveur_tcp.c`, recevoir la taille du message avant de recevoir le message puis comparer la taille du message avec le nombre d'octets effectivement reçus, ce qui permettra de détecter le problème. Le serveur envoie alors une valeur (-1 par exemple) pour signaler la fin du processus de traitement (et la fermeture éventuelle de la socket de dialogue).

Cette méthode permet également de laisser la socket de dialogue ouverte pour des futurs échanges éventuels (ce qui n'est pas possible dans l'exercice 4). Par exemple, vous pouvez modifier votre code pour envoyer deux fichiers (ou plus) l'un après l'autre.

TD Réseaux
Introduction à UDP

Licence d'informatique

1 Un échange avec UDP

On désire transmettre les informations initiales d'une vente aux enchères :
- une chaîne de caractères contenant la description de la vente (chaîne de caractères de longueur au plus 80),
- un entier contenant la mise à prix initiale.

1.1 Le point de vue de l'émetteur

Compléter le code ci-dessous en suivant les instructions des commentaires.

```
/* emetteur portReceveur machineReceveur */
main(int argc, char **argv)
{
int sock, envoye, recu;
char confirmation[256];
struct sockaddr_in adresseReceveur;
int lgadresseReceveur;
struct hostent *hote;

char descriptionVente[80];
int offreInitiale = 149;

strcpy(descriptionVente, "Téléphone portable");

/* cr'eation de la socket */
if ((sock = socket( ... , ... , 0 )) == -1) {
  perror("socket"); exit(1);}

/* recherche de l'@ IP de la machine distante */
if ((hote = gethostbyname(argv[2])) == NULL){
  perror("gethostbyname"); exit(2);}

/* pr'eparation de l'adresse distante : port + la premier @ IP */
adresseReceveur.sin_family = AF_INET;
adresseReceveur.sin_port = htons(atoi(argv[1]));
bcopy(hote->h_addr, &adresseReceveur.sin_addr, hote->h_length);
printf("L'adresse en notation pointee %s\n", inet_ntoa(adresseReceveur.sin_addr));

/* envoi de la description */
lgadresseReceveur = ...
```

```
if ((envoye = sendto(sock, ..., ... , 0, ... , ... )) != ... ) {
  perror("sendto descriptionVente"); exit(1);}
printf("descriptionVente envoyee\n");

/* envoi de l'offre initiale */
if ((envoye = sendto(sock, ..., ... , 0, ... , ... )) != ... ) {
  perror("sendto offreInitiale"); exit(1);}
printf("offreInitiale envoyee\n");

/* confirmation de reception */
if ((recu = recvfrom(sock, ... , ... , 0, ... , ... )) == ... ) {
  perror("recvfrom"); exit(1);}
printf("j'ai recu la confirmation ... bye-bye\n");
}
```

1.2 Le point de vue du receveur

Compléter le code ci-dessous en suivant les instructions des commentaires.

```
/* receveur portReceveur */
main (int argc, char **argv)
{
int sock,recu,envoye;
char confirmation[256], nomh[50];
struct sockaddr_in adresseLocale;
int lgadresseLocale;
struct sockaddr_in adresseEmetteur;
int lgadresseEmetteur;
struct hostent *hote;

char descriptionVente[80];
int offreInitiale;

/* cr'eation de la socket */
if ((sock = socket( ... , ... , 0 )) == -1) {
  perror("socket"); exit(1);}

/* r'ecup'eration du nom de la machine pr'esente */
if (gethostname(nomh, 50) == -1) {
  perror("gethostbyname"); exit(1);}
printf("Je m'execute sur %s\n", nomh);

/* pr'eparation de l'adresse locale : port + toutes les @ IP */
adresseLocale.sin_family = AF_INET;
adresseLocale.sin_port = htons(atoi(argv[1]));
adresseLocale.sin_addr.s_addr = htonl(INADDR_ANY);

/* attachement de la socket a' l'adresse locale */
lgadresseLocale = sizeof(adresseLocale);
if ((bind(sock, &adresseLocale, lgadresseLocale)) == -1) {
  perror("bind"); exit(1);}
```

```
/* reception de la description */
lgqdresseEmetteur = ...
if ((recu = recvfrom(sock, ... , ... , 0, ... , ... )) == ... ) {
perror("recvfrom descriptionVente"); exit(1);}
printf("j'ai recu la description : %s\n", descriptionVente);

/* reception de l'offre initiale */
if ((recu = recvfrom(sock, ... , ... , 0, ... , ... )) == ... ) {
perror("recvfrom offreInitiale"); exit(1);}
printf("j'ai recu l'offre initiale : %d\n", offreInitiale);

/* envoi de la confirmation de reception */
strcpy(confirmation, "conditions initiales recues");
if ((envoye = sendto(sock, ..., ... , 0, ... , ... )) != ... ) {
  perror("sendto"); exit(1);}
printf("confirmation envoyee ...adios\n");
}
```

2 Vente aux enchères - première version simplifiée

Une vente aux enchères sur le réseau peut être mis en œuvre par un système client-serveur.

Le commissaire-priseur (= serveur) est responsable de la vente, il accueille les acheteurs et leur donne les conditions initiales de vente. Puis, il lance le début de la vente en envoyant l'offre de prix en cours à tous les acheteurs. Il reçoit les offres de prix et conclut la vente.

Les acheteurs (= clients) reçoivent les conditions initiales de la vente aux enchères. Jusqu'à ce que la vente soit conclue, les acheteurs reçoivent des offres de prix successives et renvoient des surenchères ou non.

2.1 Enchères : le point de vue du commissaire-priseur (serveur)

Le serveur utilise également deux sockets, l'une pour accueillir les clients `socketAccueil`, l'autre pour la réalisation de la vente `socketVente`. Il dispose de `adressesClients` tableau global d'adresses des clients, de `nbclients` variable globale contenant le nombre de clients et de `offreLaPlusElevee` variable globale contenant l'offre la plus élevée reçue.

1. Le serveur réalise l'accueil des clients sur la socket `socketAccueil`. Après chaque nouvel accueil, le serveur demande à l'utilisateur (interaction clavier) si il doit continuer l'accueil. Si oui, le serveur retourne au point 1, si non, la vente peut commencer (point 2).
2. Le serveur lance la vente. Il envoie les conditions de mise en vente à chaque client :
   — une chaîne de caractères contenant la description de la vente,
   — un entier long contenant la mise à prix initiale.
3. Le serveur se met en attente de réception d'une offre sur la socket de vente `socketVente`. Lorsqu'un client se manifeste (cf. 2.2), le serveur lit l'offre de prix du client. Le serveur calcule la nouvelle offre de prix, qui est le maximum entre l'offre de prix la plus élevée courante et l'offre de prix reçue et l'affiche.
4. Le serveur demande à l'utilisateur (interaction clavier) s'il faut continuer les enchères ou non. S'il faut continuer, le serveur envoie la nouvelle offre de prix à tous les clients puis retourne au point 3.

S'il faut arrêter, le serveur envoie un message vide à chaque client (= un message de longueur 0), puis le serveur envoie ensuite la dernière offre de prix reçue à tous les clients. La vente sera alors adjugée au client qui a fait la dernière offre.

2.2 Encheres : le point de vue de l'acheteur (client)

Le client dispose de `sockDialogue` variable globale contenant sa socket locale, de `adresseVente` variable globale contenant l'adresse de la socket de service du serveur qui lui est dédiée (construite grâce à la connaissance du nom de la machine sur laquelle le serveur est executé, et du numéro de port utilisé par le serveur), de `offreCourante` variable globale contenant la dernière offre envoyée au serveur, et de `offreLaPlusEleveeRecue` variable globale contenant la dernière offre reçue du serveur.

1. Le client envoie un message au serveur pour demander si il peut participer à la vente.
2. Le client se met en attente du début de la vente. Celle-ci commence par la réception des conditions de mise en vente envoyées par le serveur : la description de la vente et la mise à prix initiale.
3. le client demande à l'utilisateur (interaction clavier) si il veut ou non faire une offre.
   — Si le client désire faire une offre, il envoie au serveur un entier contenant son offre, puis il se met en attente de réception de la nouvelle offre courante en provenance du serveur (point 4).
   — Si le client ne désire pas faire d'offre (saisie de la valeur 0 par exemple), il se met tout de suite en attente de réception de la nouvelle offre courante en provenance du serveur (point 4).
4. Lorsque le serveur envoie une nouvelle valeur, le client doit déterminer si la vente est terminée ou non :
   — Si le client a reçu un paquet de taille nulle, la vente est terminée. Le client se met en attente de réception de la dernière offre reçue par le serveur. Le client compare alors sa dernière offre envoyée à la dernière offre reçue du serveur, si elles sont identiques, c'est à lui que la vente est adjugée.
   — Si le client a reçu une valeur, c'est donc la nouvelle offre faite par un client au serveur. Le client retourne au point 3.

2.3 travail à faire

Décrire avec un schéma temporel, le déroulement d'une vente aux enchères avec 3 acheteurs et un commissaire priseur. Cette description doit vous permettre de mieux comprendre le fonctionnement attendu.

Construire les algorithmes associés au commissaire priseur et aux acheteurs.

TP Réseaux
Introduction à UDP

Licence d'informatique

Vous allez réaliser des échanges sous UDP entre un émetteur et un receveur. Il est impératif qu'émetteur et receveur s'exécutent sur deux machines différentes.

---

1 Caractéristiques d'UDP

1.1 Consignes

Récupérez les fichiers `emetteur_udp.c` et `receveur_udp.c` dans le répertoire `/home/commun_depinfo/enseignants/leparc/Reseaux/UDP_exo1` et complêtez-les, pour qu'`emetteur_udp.c` envoie un entier tandis que `receveur_udp.c` reçoit un entier.

1.2 UDP est en mode non connecté

Faites fonctionner ces deux programmes entre deux machines Linux :
— en lançant d'abord le receveur, puis l'émetteur ;
— en lançant d'abord l'émetteur, puis le receveur.

1.3 UDP perd des paquets

Modifier les programmes `emetteur_udp.c` et `receveur_udp.c` pour que :
— à l'aide d'une boucle, l'émetteur envoie les entiers de 1 à N, N étant passé en paramètre (dans `argv[3]`).
— le receveur boucle indéfiniment sur la réception d'un entier, et à chaque réception incrémente une variable globale `somme` initialement à 0.
La boucle infinie sera arrêtée au moyen d'un `Control-C` (signal `SIGINT`), en conséquence il faut rajouter dans le handler `arretParControlC` l'affichage du contenu de la variable `somme`.

Faire fonctionner ces deux programmes, en augmentant progressivement N jusqu'à ce que des paquets soient perdus. La valeur de N peut-être relativement grande en fonction des machines utilisées (charge de la machine) et du réseau les reliant.

1.4 UDP préserve les limites de messages

Modifier votre programme précédent pour que l'émetteur puisse envoyer, successivement, deux chaines d'une vingtaine de caractères, au receveur. Vous ferez en sorte que le receveur soit lancé, mais pas en attente de lecture lors de l'émission des deux messages (receveur bloqué sur un `scanf` par exemple). Observer le comportement des programmes en vous souvenant du comportement en TCP.

Modifier votre receveur pour que le buffer de réception ait une taille de 10 octets. Observer le comportement des programmes en vous souvenant du comportement en TCP.

2 Vente aux enchères - première version simplifiée

Une vente aux enchères sur le réseau peut être mis en œuvre par un système client-serveur.

Le commissaire-priseur (= serveur) est responsable de la vente, il accueille les acheteurs et leur donne les conditions initiales de vente. Puis, il lance le début de la vente en envoyant l'offre de prix en cours à tous les acheteurs. Il reçoit les offres de prix et conclut la vente.

Les acheteurs (= clients) reçoivent les conditions initiales de la vente aux enchères. Jusqu'à ce que la vente soit conclue, les acheteurs reçoivent des offres de prix successives et renvoient des surenchères ou non.

2.1 Enchères : le point de vue du commissaire-priseur (serveur)

Le serveur utilise également deux sockets, l'une pour accueillir les clients `socketAccueil`, l'autre pour la réalisation de la vente `socketVente`. Il dispose de `adressesClients` tableau global d'adresses des clients, de `nbclients` variable globale contenant le nombre de clients et de `offreLaPlusElevee` variable globale contenant l'offre la plus élevée reçue.

1. Le serveur réalise l'accueil des clients sur la socket `socketAccueil`. Après chaque nouvel accueil, le serveur demande à l'utilisateur (interaction clavier) si il doit continuer l'accueil. Si oui, le serveur retourne au point 1, si non, la vente peut commencer (point 2).
2. Le serveur lance la vente. Il envoie les conditions de mise en vente à chaque client :
   — une chaîne de caractères contenant la description de la vente,
   — un entier long contenant la mise à prix initiale.
3. Le serveur se met en attente de réception d'une offre sur la socket de vente `socketVente`. Lorsqu'un client se manifeste (cf. 2.2), le serveur lit l'offre de prix du client. Le serveur calcule la nouvelle offre de prix, qui est le maximum entre l'offre de prix la plus élevée courante et l'offre de prix reçue et l'affiche.
4. Le serveur demande à l'utilisateur (interaction clavier) s'il faut continuer les enchères ou non. S'il faut continuer, le serveur envoie la nouvelle offre de prix à tous les clients puis retourne au point 3.
   S'il faut arrêter, le serveur envoie un message vide à chaque client (= un message de longueur 0), puis le serveur envoie ensuite la dernière offre de prix reçue à tous les clients. La vente sera alors adjugée au client qui a fait la dernière offre.

Ecrire, en C, un programme mettant en œuvre le point de vue du commissaire-priseur (serveur) décrit ci-dessus.

2.2 Encheres : le point de vue de l'acheteur (client)

Le client dispose de `sockDialogue` variable globale contenant sa socket locale, de `adresseVente` variable globale contenant l'adresse de la socket de service du serveur qui lui est dédiée (construite grâce à la connaissance du nom de la machine sur laquelle le serveur est executé, et du numéro de port utilisé par le serveur), de `offreCourante` variable globale contenant la dernière offre envoyée au serveur, et de `offreLaPlusEleveeRecue` variable globale contenant la dernière offre reçue du serveur.

1. Le client envoie un message au serveur pour demander si il peut participer à la vente.
2. Le client se met en attente du début de la vente. Celle-ci commence par la réception des conditions de mise en vente envoyées par le serveur : la description de la vente et la mise à prix initiale.
3. le client demande à l'utilisateur (interaction clavier) si il veut ou non faire une offre.
   — Si le client désire faire une offre, il envoie au serveur un entier contenant son offre, puis il se met en attente de réception de la nouvelle offre courante en provenance du serveur (point 4).

— Si le client ne désire pas faire d'offre (saisie de la valeur 0 par exemple), il se met tout de suite en attente de réception de la nouvelle offre courante en provenance du serveur (point 4).

4. Lorsque le serveur envoie une nouvelle valeur, le client doit déterminer si la vente est terminée ou non :
   — Si le client a reçu un paquet de taille nulle, la vente est terminée. Le client se met en attente de réception de la dernière offre reçue par le serveur. Le client compare alors sa dernière offre envoyée à la dernière offre reçue du serveur, si elles sont identiques, c'est à lui que la vente est adjugée.
   — Si le client a reçu une valeur, c'est donc la nouvelle offre faite par un client au serveur. Le client retourne au point 3.

Ecrire, en C, un programme mettant en œuvre le point de vue de l'acheteur (client) décrit ci-dessus.

3 Vente aux enchères - deuxième version

Par rapport à la première version, on rajoute les contraintes suivantes :
— Les clients peuvent entrer et sortir de la vente aux enchères quand ils le souhaitent.
— La réception des offres proposées par les autres acheteurs se fait parallèlement à l'attente d'une proposition d'offre.
— La vente est conclue quand le commissaire ne reçoit plus d'offres pendant un temps déterminé.

D'un point de vue technique, on utilisera de manière importante la fonction `select` pour l'attente multiple sur la ou les sockets et sur l'interaction clavier.

Il est conseillé de modifier d'abord le client et de le tester avec le serveur de la question précédente.

3.1 Le point de vue du commissaire-priseur (serveur)

3.1.1 Description logique

Pendant toute la vente, le commissaire-priseur gère les arrivées et les départs des clients (un client qui détient l'enchère la plus élevée ne peut pas partir).

Le commissaire-priseur reçoit les offres, et vérifie leur validité (une enchère ne peut qu'augmenter, deux clients ne peuvent posséder simultanément l'offre la plus élevée).

Si aucune offre n'est reçue pendant un temps limité (paramétrable), le commissaire-priseur décide de la fin de la vente (*1, 2, 3, Adjugé-Vendu)* et adjuge la vente au client ayant fait l'offre la plus élevée courante. Le commissaire-priseur signale alors la fin de la vente aux clients.

3.1.2 Description technique

— Le serveur dispose de deux sockets :
   — d'une *socket d'acceuil* attachée sur un port UDP connu des clients, ce port sera utilisé pour demander à entrer dans la vente,
   — d'une *socket de vente* attachée sur un port UDP quelconque, ce port sera utilisé par les clients pour participer à la vente.
— Le serveur se met en attente en lecture sur **les deux sockets** car on peut maintenant arriver ou partir pendant la vente. Il se met également en attente sur le clavier (descripteur 0), pour que l'utilisateur puisse indiquer la fin de la vente, si nécessaire.
— Lorsqu'un client se manifeste pour la première fois (sur la socket d'accueil), le serveur stocke l'adresse de ce nouveau client puis répond au nouveau client, **via la socket de vente**, qu'il est reconnu dans le système (c'est cette socket de vente qui est utilisée pour communiquer avec les clients), puis se remet en attente sur la socket d'accueil.

On suppose que les adresses des sockets des clients sont stockées dans un tableau global d'adresses appelé `adressesClients` et que le nombre de clients est stocké dans la variable globale `nbclients` initialement à 0.

```
#define NB_CLIENTS_MAX 100
struct sockaddr_in adressesClients[NB_CLIENTS_MAX];
int nbclients = 0;
```

Ecrire, en C, un programme mettant en œuvre le point de vue du commissaire-priseur (serveur) décrit ci-dessus.

3.2 Le point de vue de l'acheteur (client)

3.2.1 Description logique

A tout moment, l'acheteur peut arriver dans la vente aux enchères, en se manifestant au commissaire-priseur. Il reçoit alors la description de la vente et l'offre la plus élevée courante.

A tout moment, l'acheteur peut partir de la vente aux enchères, sauf si il détient l'offre la plus élevée courante.

Pendant la vente, le programme de l'acheteur est en permanence en réception de l'offre la plus élevée courante et peut également faire une offre.

L'acheteur doit pouvoir reconnaître le signal de la fin de la vente. Le commissaire-priseur indique alors à l'acheteur s'il a remporté ou non l'enchère.

3.2.2 Description technique

- Le client doit disposer d'une socket.
- Le client doit construire l'adresse du serveur, il connaît le numéro de port de la socket d'accueil et le nom de la machine du serveur.
- Puis le client envoie un message de sa socket locale à la socket d'accueil du serveur et se met en attente de réponse.
- Lorsque le serveur répond, le client est débloqué, il connait alors l'adresse de la socket du serveur sur laquelle a lieu la vente, et peut participer à la vente.
- le client se met en écoute sur **sa socket et sur le clavier**. Il peut ainsi recevoir les nouvelles offres et prendre en compte les surenchères de l'utilisateur.

On suppose que la socket locale du client est stockée dans une variable globale `sockDialogue`, que l'adresse de la socket d'accueil du serveur est stockée dans la variable globale `adresseServeurAccueil` et que l'adresse de la socket du serveur, sur laquelle a lieu la vente, est stockée dans la variable globale `adresseVente`

```
int sockDialogue;
struct sockaddr_in adresseServeurAccueil;
struct sockaddr_in adresseVente;
```

Ecrire, en C, un programme mettant en œuvre le point de vue de l'acheteur (client) décrit ci-dessus.