- 4-+

Projet C et Réseau
API socket (UDP / TCP)
Système de boîtes aux lettres en langage C

A. Présentation générale

1. Objectif

L'objectif du BE est de réaliser une application distribuée de boîtes aux lettres sur Internet, utilisant l'API socket. Le langage utilisé est le langage C. Le travail à réaliser en TP est décomposé en 2 parties : la 1ère partie est principalement dédiée à l'API socket (à mettre en œuvre dans le cadre d'une application simple) ; la 2ème partie traite de concepts avancés du langage C.

2. Organisation et évaluation

Le BE comporte 2 séances de TD et 5 séances de TP. Les séances de TD ont pour premier objectif de permettre aux étudiants de comprendre le sujet et de se familiariser avec les fonctions C nécessaires au BE, potentiellement au-delà de celles étudiées en cours. Le second objectif est de modéliser le comportement des entités mises en jeu dans la 2ème partie du BE (E, R et BAL).

Les TP se déroulent sur PC Linux (en présentiel à l'INSA).

Pour tester le programme, il s'agit idéalement d'exécuter les programmes applicatifs sur 2 machines distinctes. Un test en local est cependant possible (localhost).

Évaluation. Le travail demandé sera noté sur la base de l'implication en TP et des versions successives du programme qui sont demandées. Le travail réalisé en séance sera à déposer dans le répertoire partagé avec l'enseignant chargé du TP à l'issue de chaque séance de TP.

B. Présentation de la 1ère partie (notée sur 10)

1. Spécification de l'application

L'application à développer est nommée **tsock** (trivial sock) et doit permettre de réaliser des échanges d'informations entre deux machines connectées à l'Internet. **tsock** est configurable soit comme une source d'informations (un émetteur), soit comme un puits d'informations (récepteur).

Usage général de l'application tsock

L'usage général de l'application **tsock** doit être le suivant :

- **tsock -p [-options] port** pour la mise en œuvre d'un puits en attente sur le port **port**
- **tsock -s [-options] host port** pour la mise en œuvre d'une source vers un puits s'exécutant sur la station **host** en attente sur le port **port.**

Les options possibles sont au nombre de trois :

- **-u** : utilise le service du protocole UDP ; par défaut, le protocole TCP est utilisé
- **-l ##** : longueur en octets du message à émettre ou longueur maximale du message à lire ; par défaut, cette longueur est de 30 octets. ## signifie une valeur (100 par exemple)
- **-n ##** : définit le nombre de messages à émettre pour le source (par défaut : 10), ou à lire pour le puits (par défaut : infini)

3. Format des messages émis

Les messages émis par la source sont composés de la manière suivante :

- la longueur d'un message est de 30 octets par défaut et peut être modifiée par l'option –l ##
- le contenu des messages est composé de deux champs :
  - le 1er champ représente le numéro du message ; il est codé en ASCII sur 5 caractères et prend pour valeur : -
  - ----1, ----2, …, 99999 (« - » désignant un caractère blanc) ;
  - le 2ème champ est une chaîne de caractères égale à la répétition d'un même caractère : le 1er message émis contient la répétition du caractère 'a', le second 'b', ..., le 26ème 'z', le 27ème 'a', etc.
- **Ex** : format du 29ème message émis de longueur 25 :

1---56----------25

---29ccccccccccccccccccccccc

4. Affichage des messages émis et reçus

Exécuté en tant que Source, le programme **tsock** doit afficher :

- les informations suivantes :
  - SOURCE : longueur du message émis, n° de port local, valeur des options, protocole de transport utilisé, nom de la machine destinataire
- puis pour chaque message émis :
  - SOURCE : Envoi n° xxxxx (yyyyy) [*…*] où :
    - xxxxx est le numéro de l'envoi (codé en ASCII sur 5 positions),
    - yyyyy est la taille du message envoyé,
    - *…* désigne le contenu du message (numéro + chaîne de caractères).

Exécuté en tant que Puits, le programme **tsock** doit afficher :

- les informations suivantes :
  - PUITS : longueur du message lu, n° de port local, valeur des options, protocole de transport utilisé
- puis pour chaque message reçu :
  - PUITS: Réception n°xxxxx (yyyyy) [*…*] où :
    - xxxxx est le numéro de la réception,
    - yyyyy est la taille du message reçu,
    - *…* désigne le contenu du message reçu (numéro + chaîne de caractères).

Exemple de session :

- Émission (depuis la machine dumas) via UDP de 5 messages de longueur égale à la longueur par défaut, à destination de la machine gauthier sur le n° port 9000.

Machine dumas (source

```
dumas> tsock -s -u -n5 gauthier 9000
 SOURCE : lg_mesg_emis=30, port=9000, nb_envois=5, TP=udp, dest=gauthier
 SOURCE : Envoi n°1  (30) [----1aaaaaaaaaaaaaaaaaaaaaaa]
 SOURCE : Envoi n°2  (30) [----2bbbbbbbbbbbbbbbbbbbbbbb]
 SOURCE : Envoi n°3  (30) [----3ccccccccccccccccccccccc]
 SOURCE : Envoi n°4  (30) [----4ddddddddddddddddddddddd]
 SOURCE : Envoi n°5  (30) [----5eeeeeeeeeeeeeeeeeeeeeee]
 SOURCE : fin
 dumas>
```

- Réception via UDP sur le n° port = 9000 d'un nombre infini de messages de longueur = celle par défaut.

Machine gauthier (puits)

```
gauthier> tsock -p -u 9000
 PUITS : lg_mesg-lu=30, port=9000, nb_receptions=infini, TP=udp
 PUITS : Reception n°1  (30) [----1aaaaaaaaaaaaaaaaaaaaaaa]
 PUITS : Reception n°2  (30) [----2bbbbbbbbbbbbbbbbbbbbbbb]
 PUITS : Reception n°3  (30) [----3ccccccccccccccccccccccc]
 PUITS : Reception n°4  (30) [----4ddddddddddddddddddddddd]
 PUITS : Reception n°5  (30) [----5eeeeeeeeeeeeeeeeeeeeeee]
```

5. Travail à faire

a) Télécharger la version v0 du code tsock : lien

tsock_v0.c contient le programme fourni en annexe pour expliquer la fonction getopt().

b) Ecrire alors une **version v1** du programme tsock_v0.c:

- en ajoutant au programme la prise en compte de l'option -u
- basée sur l'utilisation du service fourni par UDP
- permettant l'échange de données de format simple : chaînes de caractères de taille fixe construites et affichées à l'aide des fonctions suivantes :

```
void construire_message (char *message, char motif, int lg) {
  int i;
  for (i=0 ; i<lg; i++) message[i] = motif; }

void afficher_message (char *message, int lg) {
  int i;
  printf("message construit : ");
  for (i=0 ; i<lg ; i++) printf("%c", message[i]) ; printf("\n"); }
```

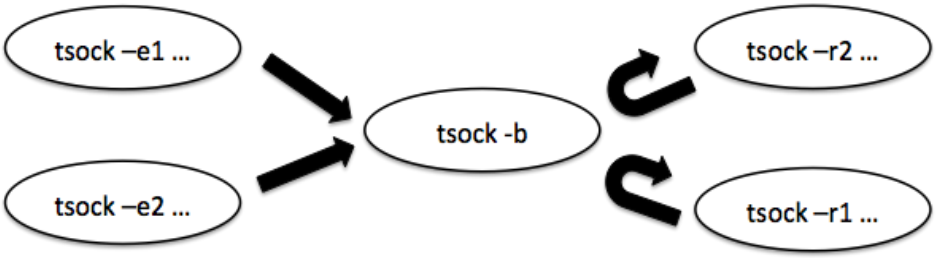
c) Inclure à la version v1 l'usage de TCP (**version v2**).

d) Inclure à la version v2 les fonctions de formatage et d'affichage des messages émis et reçus ainsi que la gestion des options restantes : -n et -l (**version v3**)

e) Bonus optionnel (si temps restant en fin de TP2) : Écrire une **version v4** basée sur la version v3, qui inverse les rôles (émetteur / récepteur) des programmes source et puit (sous TCP), et qui fasse en sorte :

- que le serveur puisse répondre à plusieurs demandes de clients émis simultanément ;
- que le serveur (dans son rôle de programme père) ne s'arrête jamais (seuls ses fils ferment la connexion une fois les messages envoyés au client) ;
- que le choix du rôle soit paramétrable via des options appropriées (par ex à la place des options -s et -p : -c -e pour client de la connexion / émetteur des données, -c -r pour client de la connexion / récepteur des données, etc.).

C. Présentation de la 2ème partie (notée sur 10)



Exemple de session :

1. Lancement sur la machine gauthier d'un serveur de BAL accessible via TCP sur le n° de port = 9000

Machine gauthier (serveur BAL)
gauthier> tsock -b 9000 PUITS : port=9000, TP=tcp

2. Émission de 2 lettres (par défaut de 30 octets chacun) à destination du récepteur 1 sur le serveur de BAL de la machine gauthier utilisant le n° de port 9000

Machine dumas (émetteur)
dumas> tsock –e1 –n2 gauthier 9000 SOURCE : lg_mesg_emis=30, port=9000, nb_envois=2, TP=tcp, dest=gauthier SOURCE : Envoi lettre n°1 à destination du récepteur 1 (30) [----1aaaaaaaaaaaaaaaaaaaaaaaaaa] SOURCE : Envoi lettre n°2 à destination du récepteur 1 (30) [----1bbbbbbbbbbbbbbbbbbbbbbbb] SOURCE : fin

```
dumas>
```

3. Émission d'1 lettre (par défaut de 30 octets) à destination du récepteur 2 sur le serveur de BAL de la machine gauthier utilisant le n° de port 9000

Machine tarzan (émetteur)

```
tarzan> tsock –e2 –n1 gauthier 9000
SOURCE : lg_mesg_emis=30, port=9000, nb_envois=1, TP=tcp, dest=gauthier
SOURCE : Envoi lettre n°1 à destination du récepteur 2 (30) [----2aaaaaaaaaaaaaaaaaaaaaaaaa]
SOURCE : fin
tarzan>
```

4. Au niveau du serveur de BAL : réception et stockage des lettres
   - réception et stockage (dans bal_1) d'une 1ère lettre à destination du récepteur 1
   - réception et stockage (dans bal_1) d'une 2ème lettre à destination du récepteur 1
   - réception et stockage (dans bal_2) d'une 1ère lettre à destination du récepteur 2

Machine gauthier (serveur BAL)

```
gauthier> tsock -b 9000
 PUITS : port=9000, TP=tcp
 PUITS : Réception et stockage lettre n°1 pour le récepteur n°1[----1aaaaaaaaaaaaaaaaaaaaaaaaa]
 PUITS : Réception et stockage lettre n°2 pour le récepteur n°1[----1bbbbbbbbbbbbbbbbbbbbbbbb]
 PUITS : Réception et stockage lettre n°1 pour le récepteur n°2[----2aaaaaaaaaaaaaaaaaaaaaaaaa]
```

5. Récupération de ses lettres par le récepteur 2 via le serveur de BAL de la machine gauthier utilisant le n° de port 9000

Machine beta (récepteur)

```
beta> tsock –r2 gauthier 9000
RECEPTION : Demande de récupération de ses lettres par le récepteur 2 port=9000, TP=tcp, dest=gauthier
RECEPTION : Récupération lettre par le récepteur 2 (30) [----2aaaaaaaaaaaaaaaaaaaaaaaaa]
RECEPTION : fin
beta>
```

6. Récupération de ses lettres par le récepteur 1 via le serveur de BAL de la machine gauthier utilisant le n° de port 9000

Machine alpha (récepteur)

```
alpha> tsock –r1 gauthier 9000
RECEPTION : Demande de récupération de ses lettres par le récepteur 1 port=9000, TP=tcp, dest=gauthier
RECEPTION : Récupération lettre par le récepteur 1 (30) [----1aaaaaaaaaaaaaaaaaaaaaaaaa]
RECEPTION : Récupération lettre par le récepteur 1 (30) [----1bbbbbbbbbbbbbbbbbbbbbbbb]
RECEPTION : fin
alpha>
```

2. Travail à faire

a) Sur la base des hypothèses suivantes, modéliser le comportement de chacune des entités E, BAL et R, incluant leur interactions **=> travail initié en TD**.

- Tous les échanges se font via des connexions TCP
- Un émetteur E est client de la connexion TCP à établir avec le serveur de BAL. Il ferme la connexion une fois la / les lettres envoyées
- Un récepteur R est client de la connexion TCP à établir avec le serveur de BAL en vue de la récupération de ses lettres.

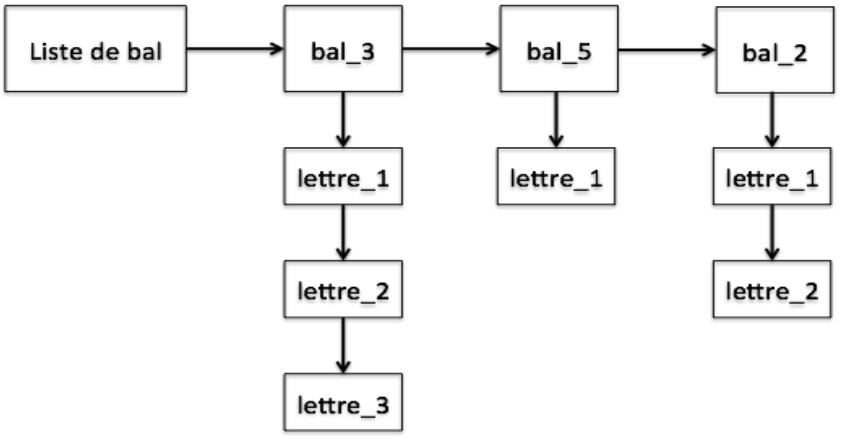
- Le serveur de BAL est serveur des connexions TCP en provenance des émetteurs / récepteurs, via un unique socket dont le un numéro de port supposé connu de tous. Il ferme la connexion avec un R une fois la / les lettres envoyées à R.
- **Simplification cette année** : le serveur de BAL traite lui-même les échanges avec les E et R (pas de fork), chaque connexion TCP étant fermée à l'issue de la transaction avec le serveur de BAL :
  - soit par le client si celui-ci est un E
  - soit par le serveur de BAL si son client est un R

IMPORTANT : Côté serveur de BAL, vous aurez à définir comment différencier si la connexion TCP acceptée provient d'un émetteur ou d'un récepteur de lettre.

b) Modifier le programme tsock pour prendre en compte les options –b, -e et –r, et le nouveau format des messages demandé. On considérera des lettres de taille fixe.

c) Ecrire en langage C le code du système de boîte aux lettres (BAL) suivant les consignes données ci-après :

- BAL est à réaliser avec une liste chaînée de boîtes aux lettres bal_i dédiées chacune à un récepteur i
- bal_i est à créer quand BAL reçoit une lettre pour le récepteur i et que bal_i n'est pas encore créée
- bal_i est à réaliser avec une liste chaînée de lettres reçues accessibles dans leur ordre d'arrivée (cf figure ci-dessous)
- une action de récupération de ses lettres par le récepteur i engendre le retrait de toutes les lettres stockées dans bal_i, dans leur ordre d'arrivée. La bal_i (une fois vidée) n'est pas détruite.



D. Déroulement du BE

TD 1 et 2

- TD1 : Lecture du sujet et éclaircissement des points ambigus + Modélisation détaillée du comportement des entités tsock source et tsock puits de la partie 1
- TD 2 : Modélisation détaillée (pseudo code) du comportement des entités E, R et BAL de la partie 2

TP ⇒ estimation prévisionnelle de la répartition des 5 séances de TP :

- 2 séances dédiées à la 1ère partie
- 3 séances dédiées à la 2ème partie dont :
  - 1,5 séance dédiée à la mise en place des boîtes aux lettres
  - 1,5 séance dédiée à la gestion des communications du système de BAL

E. Annexe nécessaire aux TD / TP

1. Quelques rappels

Pour compiler le programme tsock.c et générer un .exe du nom de tsock : gcc tsock.c -o tsock

L'exécution de la commande : tsock -p -u 9000 donnera la valeur 4 à argc et les valeurs suivantes à argv : argv[0] = "tsock", argv[1] = "-p", argv[2] = "-u", argv[argc-1] = "9000"

=> pour récupérer l'entier correspondant à la chaîne de caractère "9000" :

```
int port = atoi(argv[argc-1])
```

Pour plus de précaution, utiliser aussi la fonction htons() :

```
port = atoi(argv[argc-1])
port = htons(port)
```

2. La fonction getopt(). NB : voir aussi : *man -D 3 getopt*

L'exécution d'un programme (tel que tsock) nécessite souvent la spécification d'un certain nombre de paramètres (éventuellement optionnels) lors du lancement de la commande correspondante.

Exemple :

- tsock -s -u -n5 dumas 9000
  - 5 paramètres : -s, -u, -n5, dumas et 9000, signifiant que le programme tsock doit être exécuté en tant que source de 5 messages via UDP à destination d'un puits s'exécutant sur la machine dumas et en attente sur le n° de port 9000 ;
- tsock -p -u 9000
  - 3 paramètres : -p, -u et 9000 signifiant que le programme tsock doit être exécuté en tant que puits d'information via UDP en attente sur le n° de port 9000.

Pour que le programme puisse interpréter de tels paramètres au moment de son exécution, une solution consiste à utiliser la fonction getopt() de la librairie <stdlib.h> lors du codage du programme.

getopt() permet au programmeur d'indiquer la liste des paramètres qui seront potentiellement spécifiés lors de l'exécution de la commande, et d'y affecter un certain comportement.

Plus exactement, cette fonction permet de spécifier la liste des paramètres **composés d'une lettre précédée d'un "-"** (et éventuellement suivie d'un argument) susceptibles d'être injectés lors de l'exécution de la commande (par exemple : -s, -u, -n 5 ou -p dans les deux exemples précédents).

Voici un extrait du man décrivant l'usage de la fonction getopt(). **Lisez le avec attention** et inspirez vous de l'exemple d'utilisation de la fonction getopt() fourni à la suite, pour écrire le programme tsock.

NAME

getopt - get option letter from argument vector

SYNOPSIS

#include <stdlib.h>

int getopt(int argc, char** argv, const char* optstring);
extern char *optarg;
extern int optind, opterr, optopt;

DESCRIPTION

getopt() returns the next option letter in argv that matches a letter in optstring.

optstring must contain the option letters the command using getopt() will recognize; if a letter is followed by a colon (:), the option is expected to have an argument, or group of arguments, which may be separated from it by white space. optarg is set to point to the start of the option argument on return from getopt().[…]

getopt() places in optind the argv index of the next argument to be processed. optind is external and is initialized to 1 before the first call to getopt(). When all options have been processed (that is, up to the first non-option argument), getopt() returns EOF. […]

RETURN VALUES

getopt() prints an error message on the standard error and returns a "?" when it encounters an option letter not included in optstring or no argument after an option that expects one.

EXAMPLE

Le programme suivant permet de prendre en compte 3 paramètres : -s et -p d'une part (identifiant si le programme doit être exécuté en tant que source ou puits) et -n ## d'autre part (identifiant le nombre de messages à envoyer ou à lire). Les paramètres -p et -s sont exclusifs et l'un des deux doit obligatoirement être spécifié ; le paramètre -n ## est optionnel.

```
/* librairie standard ... */
	#include <stdlib.h>
/* pour getopt */
	#include <unistd.h>
/* déclaration des types de base */
	#include <sys/types.h>
/* constantes relatives aux domaines, types et protocoles */
	#include <sys/socket.h>
/* constantes et structures propres au domaine UNIX */
	#include <sys/un.h>
/* constantes et structures propres au domaine INTERNET */
	#include <netinet/in.h>
/* structures retournées par les fonctions de gestion de la base de données du réseau */
	#include <netdb.h>
/* pour les entrées/sorties */
	#include <stdio.h>
/* pour la gestion des erreurs */
	#include <errno.h>

void main (int argc, char **argv)
{
	int c;
	extern char *optarg;
	extern int optind;
	int nb_message = -1; /* Nb de messages à envoyer ou à recevoir, par défaut : 10 en émission, infini en
	réception */
	int source = -1 ; /* 0=puits, 1=source */

	while ((c = getopt( "psn:") != -1) {
			switch (c) {
				case 'p':
					if (source == 1) {
						printf("usage: cmd [-p|-s][-n ##]\n");
						exit(1);
					}
					source = 0;
					break;

				case 's':
					if (source == 0) {
						printf("usage: cmd [-p|-s][-n ##]\n");
						exit(1) ;
					}
					source = 1;
					break;

				case 'n':
					nb_message = atoi(optarg);
					break;

				default:
					printf("usage: cmd [-p|-s][-n ##]\n");
					break;
			}
	}

	if (source == -1) {
			printf("usage: cmd [-p|-s][-n ##]\n");
			exit(1) ;
	}
```

```
	if (source == 1)
		printf("on est dans le source\n");
	else
		printf("on est dans le puits\n");

	if (nb_message != -1) {
		if (source == 1)
			printf("nb de tampons à envoyer : %d\n", nb_message);
		else
			printf("nb de tampons à recevoir : %d\n", nb_message);
	} else {
		if (source == 1) {
			nb_message = 10 ;
			printf("nb de tampons à envoyer = 10 par défaut\n");
		} else
			printf("nb de tampons à recevoir = infini\n");
	}
}
```

NB (hors BE) : pour information si vous aviez à travailler sous Windows, il s'agirait alors :

- d'ajouter :

  #include <winsock2.h>

  #include <windows.h>
- de supprimer tout ce qui est en lien avec Linux (sys/***)
- Supprimer aussi net/inet.h
- Si vous utilisez codeblocks, consulter aussi :
  - https://openclassrooms.com/forum/sujet/undefined-reference-to-socket12039-24247