Application Internet et Programmation Socket (AIPS) : partie Programmation Socket en langage C (API socket UDP / TCP)

Lien d'accès au sujet : https://cutt.ly/QkiK9vh

A. Présentation du sujet

1. Objectif

L'objectif des TP (2 séances) est de réaliser une application distribuée sur Internet, utilisant l'API socket. Les spécifications de l'application sont décrites en section 3. Le langage de programmation utilisé est le langage C.

2. Organisation

Les TP se déroulent sur PC Linux. Vous pourrez tester vos programmes en vous connectant à distance sur une autre station. Le travail demandé sera noté sur la base de votre implication en TP et des versions successives du programme qui vous sont demandées.

Le travail réalisé en séance est à déposer sous Moodle à l'issue de chaque séance de TP. Un questionnaire Moodle sera également à remplir pour indiquer votre état d'avancement.

3. Spécifications de l'application

L'application à mettre en œuvre est nommée **tsock** 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).

3.1. 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 (en émission) ou longueur maximale du message à lire (en réception) ; par défaut, en émission comme en réception, cette longueur est de 30 octets. ## signifie une valeur (100 par exemple)
- **`-n ##`** : définit le nombre de messages soit à émettre pour la source (par défaut : 10), soit à lire pour le puits (par défaut : infini)

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 premier 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 second champ est une chaîne de caractères égale à la répétition d'un même caractère ASCII : le premier 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
```

3.2. 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).

3.3. Exemple de session

Emission (depuis la machine dumas) via UDP de 4 messages de longueur égale à la longueur par défaut, à destination de la machine gauthier sur le n° de port 9000.

Machine dumas (source)

```
dumas> tsock -s -u -n 4 gauthier 9000
 SOURCE : lg_mesg_emis=30, port=9000, nb_envois=4, TP=udp, dest=gauthier
 SOURCE : Envoi n°1  (30) [----1aaaaaaaaaaaaaaaaaaaaaaaa]
 SOURCE : Envoi n°2  (30) [----2bbbbbbbbbbbbbbbbbbbbbbbb]
 SOURCE : Envoi n°3  (30) [----3ccccccccccccccccccccccc]
 SOURCE : Envoi n°4  (30) [----4ddddddddddddddddddddddddd]
 SOURCE : fin
 dumas>
```

Réception via UDP sur le n° de port 9000 d'un nombre infini de messages de longueur égale à la longueur 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) [----1aaaaaaaaaaaaaaaaaaaaaaaa]
 PUITS : Reception n°2  (30) [----2bbbbbbbbbbbbbbbbbbbbbbbb]
 PUITS : Reception n°3  (30) [----3ccccccccccccccccccccccc]
 PUITS : Reception n°4  (30) [----4ddddddddddddddddddddddddd]
```

4. Travail à réaliser

Écrire en langage C le programme correspondant à la spécification de l'application tsock décrite en section 3. La gestion des paramètres sera réalisée à l'aide de la primitive getopt()dont la documentation est fournie en Annexe.

Pour cela, vous adopterez la démarche suivante.

a) Récupérer le fichier tsock_v0.c accessible au lien suivant : https://urlz.fr/8RkY
tsock_v0.c contient le programme fourni en annexe pour expliquer la fonction getopt().

b) Ecrire 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 le plus simple possible : chaînes de caractères de taille fixe construite et affichée 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) **Optionnel (bonus)** : Écrire une **version v4** (finale) basée sur la version v3, qui inverse les rôles (émetteur / récepteur) des programmes source et puit, et qui fasse en sorte que :

- le serveur puisse répondre à plusieurs demandes de clients émis simultanément.
- 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) .
- 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.).

B. Annexe

1. Quelques rappels utiles

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. Fonction getopt(). 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 -n4 dumas 9000

- 5 paramètres : -s, -u, -n4, 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 ces paramètres au moment de son exécution, une solution consiste à utiliser la fonction `getopt()` de la librairie <stdlib.h> lors du codage du programme. Cette fonction permet 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 "-"** (suivie ou non d'un argument) susceptibles d'être injectés lors de l'exécution de la commande (par exemple : -s, -u, -n 5 ou -p dans les 2 exemples précédents).

Voici un extrait du man décrivant l'usage de `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 en tant que 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(argc, argv, "pn:s")) != -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 envoyer = infini\n");
      }
}
```

Veuillez nommer vos fichiers .zip de la façon suivante : NOM1-NOM2-TP1 et NOM1-NOM2-TP2

Chaque fichier .zip doit contenir :

1. le code source produit
2. un fichier Readme indiquant :
   - vos noms et prénoms
   - la commande utilisée pour générer votre exécutable
   - votre état d'avancement à l'issue de la séance de TP : ce qui fonctionne et ce que vous n'avez pas eu le temps de finaliser