Exercice sur les librairies

1. Afin de simplifier la programmation réseau, il vous est demandé de programmer une librairie de fonctions destinée aux communications TCP v4 (v6).
2. Compiler votre libraire pour en faire une librairie dynamique (.so sous Linux)
3. A l'aide de la libraire réalisée plus haut, refaire les clients et serveurs pour le protocole echo.

1 Réalisation de la librairie

La librairie réalisée implémente les fonctions suivantes [1] :

— `int client_connect(char *url, char *service)` : initiates an active TCP connection

— `int server_listen(char *service, int backlog)` : initiates a passive TCP connection

— `int server_accept(int sockfd, char *out_client_ip)` : accepts the connection from a client

— `int send_data(int sockfd, char *buffer, ssize_t length)` : sends data to the remote host

— `ssize_t receive_data(int sockfd, char *out_buffer, ssize_t max_length)` : receives data from the remote host

— `int expect_data(int sockfd, char *out_buffer, ssize_t length)` : expects a given amount of data from the remote host

— `void disconnect(int sockfd)` : closes the socket

2 Compilation en librairie dynamique et liens symboliques

La librairie est compilée en version 1.0 au moyen de la commande suivante [2] :

```
cc -g -Wall -Wpedantic -Wextra -Iinc -Wl,-soname,libtcp.so.1
    -shared -fPIC -o lib/libtcp.so.1.0 lib/tcp-util.c
```

`cc` compilateur utilisé

`-g` debug

`-Wall -Wpedantic -Wextra` warnings supplémentaires à la compilation

`-Iinc` ajout du répertoire *inc* au path des include (les fichiers présents dans le répertoire *inc* peuvent être inclus par `"file.h"` au lieu de `"inc/file.h"`)

`-Wl,-soname,libtcp.so.1` les projets utilisant cette version (1.0) devront se référer à sa version majeure uniquement

`-shared` compilation dynamique de la librairie

`-fPIC` offsets relatifs (compilation en Position Independent Code)

`-o lib/libtcp.so.1.0` output de la compilation

`lib/tcp-util.c` fichier compilé

1. Prototypes dans *inc/tcp-util.h* et implémentation dans *lib/tcp-util.c*
2. Makefile, ligne 30

La commande ldconfig me permet de créer le lien symbolique de la version 1 de ma librairie (.so.1) vers la version 1.0 (.so.1.0) [3] :

```
ldconfig -r lib -n .
```

Je crée un lien symbolique de la librairie (.so) vers la version 1 (.so.1) avec la commande suivante [4] :

```
ln -sf libtcp.so.1 lib/libtcp.so
```

3 Compilation des clients et serveurs echo avec la librairie dynamique

Je compile les fichiers source en utilisant la librairie dynamique *libtcp.so* avec les commandes suivantes, respectivement pour *echo-client.c* et *echo-server.c* [5] :

```
cc -g -Wall -Wpedantic -Wextra -Iinc -Llib -Wl,-rpath='\$ORIGIN/lib'
    -o echo-client src/echo-client.c lib/libtcp.so
cc -g -Wall -Wpedantic -Wextra -Iinc -Llib -Wl,-rpath='\$ORIGIN/lib'
    -o echo-server src/echo-server.c lib/libtcp.so
```

`-Llib` spécifie où trouver ma librairie au moment de la compilation (dans le répertoire *lib*)

`-Wl,-rpath='$ORIGIN/lib'` commande pour le linker (lors de l'exécution) : ma librairie ne se trouve pas dans les librairies du système mais dans le répertoire relatif *lib*)

`-o echo-client` **et** `-o echo-server` output de la compilation (fichiers exécutables)

`src/echo-client.c` **et** `src/echo-server.c` fichiers source à compiler

`lib/libtcp.so` interprété par `gmake` à partir de `-ltcp`, fait référence à ma librairie

3. Makefile, ligne 26
4. Makefile, ligne 22
5. Makefile, ligne 18

4 Installation sur les machines virtuelles

Je commence par compiler le projet sur ma machine locale avec le *Makefile* (dont les commandes sont détaillées plus haut) en lançant la commande `gmake` [6] :

```
[lbin@nibbler ex-lib]$ gmake
cc -g  -Wall -Wpedantic -Wextra -Iinc -Wl,-soname,libtcp.so.1 -shared -fPIC -o lib/libtcp.so.1.0 lib/tcp-util.c
ldconfig -r lib -n .
ln -sf libtcp.so.1 lib/libtcp.so
cc -g  -Wall -Wpedantic -Wextra -Iinc -Llib -Wl,-rpath='$ORIGIN/lib' -o echo-client src/echo-client.c lib/libtcp.so
cc -g  -Wall -Wpedantic -Wextra -Iinc -Llib -Wl,-rpath='$ORIGIN/lib' -o echo-server src/echo-server.c lib/libtcp.so
```

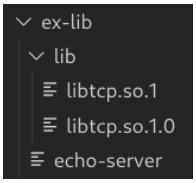
Les commandes prévues ont été exécutées et la racine du projet contient maintenant les exécutables *echo-client* et *echo-server* :

```
[lbin@nibbler ex-lib]$ ls -l
total 72
-rwxrwxr-x. 1 lbin lbin 22688 Nov 15 18:49 echo-client
-rwxrwxr-x. 1 lbin lbin 24632 Nov 15 18:49 echo-server
drwxr-xr-x. 2 lbin lbin  4096 Nov 14 21:17 inc
drwxr-xr-x. 2 lbin lbin  4096 Nov 15 18:49 lib
-rw-r--r--. 1 lbin lbin   779 Nov 15 18:29 Makefile
drwxr-xr-x. 2 lbin lbin  4096 Nov 14 15:58 src
```

Le répertoire *lib* contient bien le fichier de ma librairie et les liens symboliques :

```
[lbin@nibbler ex-lib]$ ls -l lib/
total 32
lrwxrwxrwx. 1 lbin lbin    11 Nov 15 18:49 libtcp.so -> libtcp.so.1
lrwxrwxrwx. 1 lbin lbin    13 Nov 15 18:49 libtcp.so.1 -> libtcp.so.1.0
-rwxrwxr-x. 1 lbin lbin 21640 Nov 15 18:49 libtcp.so.1.0
-rw-r--r--. 1 lbin lbin  6347 Nov 15 15:17 tcp-util.c
```

Je copie l'exécutable du serveur et la librairie sur ma machine virtuelle :



Je recrée le lien symbolique :

```
lbin@ri-server:~/ex-lib/lib$ ldconfig -n .
lbin@ri-server:~/ex-lib/lib$ ls -l
total 24
lrwxrwxrwx 1 lbin lbin    13 Nov 15 19:42 libtcp.so.1 -> libtcp.so.1.0
-rw-rw-r-- 1 lbin lbin 21480 Nov 15 19:18 libtcp.so.1.0
```

6. `gmake` est utilisé à la place de `make` car le paramètre `-ltcp` est une fonctionnalité GNU

Et je rend mon fichier *echo-server* exécutable :

```
lbin@ri-server:~/ex-lib$ chmod +x echo-server
lbin@ri-server:~/ex-lib$ ls -l
total 32
-rwxrwxr-x 1 lbin lbin 24632 Nov 15 18:05 echo-server
drwxrwxr-x 2 lbin lbin  4096 Nov 15 18:11 lib
```

Je répète les mêmes opérations pour le client.

5 Tests

5.1 Test de l'envoi d'un message simple

Je lance le serveur (avec `sudo`) :

```
lbin@ri-server:~/ex-lib$ sudo ./echo-server
[sudo] password for lbin:
[server] waiting for connections...
```

Je lance mon client en envoyant un message simple au serveur :

```
lbin@ri-client:~/ex-lib$ ./echo-client 192.168.122.2 "hello !"
[client] message received: "hello !"
lbin@ri-client:~/ex-lib$
```

Le message a bien été renvoyé par le serveur :

```
[server] waiting for connections...
[server] connection received from 192.168.122.21...
[server:192.168.122.21] 7 bytes received
[server:192.168.122.21] closing
```

Après avoir renvoyé le message, le serveur ferme le socket de la connexion ouverte par le client.

5.2 Test de l'envoi d'un message de 100000 caractères

Je teste l'envoi d'un message plus conséquent en augmentant la taille du buffer accepté par mon programme [7] et en activant les messages de debug des méthodes `send_data` et `expect_data` [8] :

```
#define BUF_SIZE 1024*1024
```

J'envoie un message contenant 100000 "1" :

```
lbin@ri-client:~/ex-lib$ ./echo-client 192.168.122.2 "$(head -c 100000 /dev/zero | tr "\0" 1)"
DEBUG [send_data] sending 100000 bytes
DEBUG [send_data] sent 100000 of 100000 bytes
DEBUG [expect_data] expecting 100000 bytes
DEBUG(expect_data): received 14480 of 100000
DEBUG [expect_data] expecting 85520 bytes
DEBUG(expect_data): received 28960 of 85520
DEBUG [expect_data] expecting 56560 bytes
DEBUG(expect_data): received 49024 of 56560
DEBUG [expect_data] expecting 7536 bytes
DEBUG(expect_data): received 7240 of 7536
DEBUG [expect_data] expecting 296 bytes
DEBUG(expect_data): received 296 of 296
[client] message received: "11111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111
```

Mon serveur reçoit ce message en plusieurs paquets et renvoie ces paquets :

```
[server] connection received from 192.168.122.21...
[server:192.168.122.21] 14480 bytes received
DEBUG [send_data] sending 14480 bytes
DEBUG [send_data] sent 14480 of 14480 bytes
[server:192.168.122.21] 28960 bytes received
DEBUG [send_data] sending 28960 bytes
DEBUG [send_data] sent 28960 of 28960 bytes
[server:192.168.122.21] 49024 bytes received
DEBUG [send_data] sending 49024 bytes
DEBUG [send_data] sent 49024 of 49024 bytes
[server:192.168.122.21] 7536 bytes received
DEBUG [send_data] sending 7536 bytes
DEBUG [send_data] sent 7536 of 7536 bytes
[server:192.168.122.21] closing
```

Le client, via la méthode `expect_data`, a bien reçu les paquets renvoyés successivement par le serveur jusqu'à atteindre un message de 100000 caractères et l'afficher. En comparant les deux dernières captures d'écran, on voit bien que les tailles des paquets envoyés et reçus correspondent.

7. La variable globale `BUF_SIZE` est définie dans le fichier *constants.h*
8. Dans le fichier *tcp-util.c*

Enfin, ce genre de tests permet de mettre en avant l'intérêt des liens symboliques puisque pour modifier les messages de debug des méthodes de ma librairie, je n'ai du réuploader que le fichier *libtcp.so.1.0* dans le répertoire *lib* de mon client et de mon serveur après avoir recompilé la librairie. J'ai également dû recompiler mes exécutables pour modifier la taille du buffer. Pour simplifier cette étape, le projet pourrait être modifié en ajoutant un fichier de configuration au projet. Cela nécésitterait toutefois de modifier mes codes source avec des allocations dynamiques.