

Travail sur les librairies

A l'aide de la librairie réalisée dans l'exercice précédent, réaliser une application client-serveur s'échangeant une liste d'éléments. Le client générera la liste contenant un nombre d'éléments passés en argument. Chaque élément contiendra un entier, une chaîne de caractère et un flottant qui seront différents pour chaque élément de la liste (ces données pourront être générées aléatoirement ou suivre une règle que vous inventerez). Le client enverra la liste au serveur et celui-ci affichera la liste à l'écran. L'envoi des données se fera en suivant le point 7.4. *Serialization—How to Pack Data* de la documentation.

1 Définition de la structure de données

Le structure de données envoyée du client au serveur une liste chaînée qui rassemble différents types de données¹ :

```
struct data_node {
    int16_t int_16;           // 16 bytes integer
    int32_t int_32;           // 32 bytes integer
    int64_t int_64;           // 64 bytes integer
    float   f;                // 32 bytes float
    double  d;                // 64 bytes double
    char    *str;             // string
    struct data_node *next;    // next node
};
```

Elles peuvent être affichées sous la forme suivante par la méthode `print_list` qui permet d'afficher l'intégralité d'une liste chaînée² :

```
Node x/TOTAL {
  int_16: 18132
  int_32: 1217769723
  int_64: 9223372036120782570
  float:  54783.093750
  double: -316265447.685803
  string: e04tiD51hvLxeai
}
```

1. Cette structure est définie dans le fichier *data.h*.

2. Comme pour toutes les méthodes concernant la structure de données, le prototype de cette méthode est défini dans le fichier *data.h* et son implémentation est réalisée dans le fichier *data.c*.

Ces données sont générées aléatoirement par la méthode `create_node`.

La liste peut ensuite être envoyée par le client avec `send_list` et reçue par le serveur avec `receive_list`. L'envoi commence par l'envoi d'un packet contenant le nombre de structures de données (nodes) qui vont être envoyées. Puis chaque structure est envoyée dans un packet indépendant précédé de sa taille. Ces headers permettent au serveur d'attendre d'avoir reçu un node entier (au cas où les packets seraient envoyés en plusieurs fois par la couche réseau) avant de le désérialiser pour composer sa propre liste de données.

Enfin, le serveur envoie un acknowledgement contenant la somme de bytes reçue (0 en cas d'erreur lors de la réception) avec `send_ack` et le client le reçoit avec `receive_ack`.

Cette liste de données peut être composée de 1 à 65535 nodes. Il n'est donc pas possible d'envoyer une liste vide.

2 Réalisation de la librairie

Pour que le client puisse envoyer les données et que le serveur puisse les recevoir, les méthodes d'envoi et de réception vont faire appel à une librairie de sérialisation. Les prototypes des fonctions de cette librairie *libserial* sont définis dans *serial-util.h* et leur implémentation est réalisée dans *serial-util.c*.

Fonctions d'écriture d'une donnée dans un buffer :

- `char *write_u16(uint16_t i, char *out_buf)` : writes a given 16 bytes unsigned integer in the buffer
- `char *write_u32(uint32_t i, char *out_buf)` : writes a given 32 bytes unsigned integer in the buffer
- `char *write_u64(uint64_t i, char *out_buf)` : writes a given 64 bytes unsigned integer in the buffer
- `char *write_f32(float f, char *out_buf)` : writes a given 32 bytes float in the buffer
- `char *write_f64(double d, char *out_buf)` : writes a given 64 bytes double in the buffer
- `char *write_str(char *str, char *out_buf)` : writes a given string in the buffer

Fonctions de lecture d'un buffer vers une donnée :

- `char *read_u16(char *buf, uint16_t out_i)` : reads a 16 bytes unsigned integer from the buffer
- `char *read_u32(char *buf, uint32_t out_i)` : reads a 32 bytes unsigned integer from the buffer
- `char *read_u64(char *buf, uint64_t out_i)` : reads a 64 bytes unsigned integer from the buffer
- `char *read_f32(char *buf, float out_f)` : reads a 32 bytes float from the buffer
- `char *read_f64(char *buf, double out_d)` : reads a 64 bytes double from the buffer
- `char *read_str(char *buf, char *out_str)` : reads a string from the buffer

La sérialisation est faite en big-endian. Pour sérialiser les float et double, j'extrais l'exposant et la mantisse (qui conserve son signe) que je sérialise sous forme d'entiers de taille fixe.

3 Structure du projet

Puisque la librairie TCP est maintenant partagée entre le projet précédent et ce projet, les librairies sont maintenant placées dans un répertoire qui se trouve au même niveau que ceux des projets :

```
- ex-lib
  + include
  + src
  Makefile
- ex-serial
  + include
  + src
  Makefile
- include
  serial-util.h
  tcp-util.h
- lib
  serial-util.c
  tcp-util.c
Makefile
```

4 Compilation du projet

Puisque la structure des dossiers a été modifiée, le *Makefile* du projet précédent doit être légèrement modifié. L'arborescence de fichiers présentée au point précédent montre qu'il y a maintenant un *Makefile* principal à la racine des projets qui peut appeler les *Makefile* des sous-projets (ensemble ou séparément).

Le *Makefile* principal compile d'abord les librairies avec les mêmes commandes que pour le projet précédent :

```
[1]      cc -g -Wall -Wpedantic -Wextra -Iinclude -Wl,-soname,libtcp.so.1
        -shared -fPIC -o lib/libtcp.so.1.0 lib/tcp-util.c
[2]      ldconfig -r lib -n .
[3]      ln -sf libtcp.so.1 lib/libtcp.so
[4]      cc -g -Wall -Wpedantic -Wextra -Iinclude -Wl,-soname,libserial.so.1
        -shared -fPIC -o lib/libserial.so.1.0 lib/serial-util.c
[5]      ldconfig -r lib -n .
[6]      ln -sf libserial.so.1 lib/libserial.so
```

- [1] La librairie *libtcp* est compilée dans sa version 1.0 (fichier de sortie : *lib/libtcp.so.1.0*) à partir du fichier *lib/tcp-util.c*. Elle sera référencée via sa version majeure (*libtcp.so.1*) par les projets qui en dépendent. Le compilateur utilisé est *cc*, en mode debug et avec warnings supplémentaires.
- [2] Création d'un lien symbolique de la version 1 de ma librairie (*libtcp.so.1*) vers la version 1.0 (*libtcp.so.1.0*)
- [3] Création d'un lien symbolique de la librairie (*libtcp.so*) vers la version 1 (*libtcp.so.1*)
- [4-6] Les mêmes opérations sont répétées pour la librairie *libserial*

Le *Makefile* compile ensuite le projet *ex-serial* :

```
[1]      gmake -C ex-serial
[2]      gmake[1]: Entering directory '/home/lbin/Documents/2020_RI/ex-serial'
[3]      mkdir out
[4]      cc -g -Wall -Wpedantic -Wextra -Iinclude -I../include
        -c -o out/data.o src/data.c
[5]      cc -g -Wall -Wpedantic -Wextra -Iinclude -I../include
        -Wl,-rpath='$_ORIGIN/../lib' -o client src/client.c
        out/data.o ../lib/libtcp.so ../lib/libserial.so
[6]      cc -g -Wall -Wpedantic -Wextra -Iinclude -I../include
        -Wl,-rpath='$_ORIGIN/../lib' -o server src/server.c
        out/data.o ../lib/libtcp.so ../lib/libserial.so
[7]      gmake[1]: Leaving directory '/home/lbin/Documents/2020_RI/ex-serial'
```

- [1] Exécution du *Makefile* du projet *ex-serial* par le *Makefile* principal
- [2] *gmake* se déplace dans le répertoire du projet
- [3] Création du répertoire *out* si il n'existe pas
- [4] Compilation du fichier *data.c* en fichier objet *data.o*
- [5] Compilation du fichier *client.c* en fichier exécutable *client* avec les dépendances aux librairies *libtcp* et *libserial* (ces librairies se trouvent dans le répertoire *lib* du répertoire parent)
- [6] Même opération que [5] pour le *server*
- [7] *gmake* retourne dans le répertoire parent

5 Installation du client et du serveur

Je commence par compiler le projet *ex-serial* en local en exécutant la commande `gmake ex-serial` dans le répertoire à la racine de mes projets :

```
[lbin@nibbler 2020_RI]$ gmake ex-serial
cc -g -Wall -Wpedantic -Wextra -Iinclude -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 -Iinclude -Wl,-soname,libserial.so.1 -shared -fPIC -o lib/libserial.so.1.0 lib/serial-util.c
ldconfig -r lib -n
ln -sf libserial.so.1 lib/libserial.so
gmake -C ex-serial
gmake[1]: Entering directory '/home/lbin/Documents/2020_RI/ex-serial'
mkdir out
cc -g -Wall -Wpedantic -Wextra -Iinclude -I../include -c -o out/data.o src/data.c
cc -g -Wall -Wpedantic -Wextra -Iinclude -I../include -Wl,-rpath='$_ORIGIN/../lib' -o client src/client.c out/data.o ../lib/libtcp.so ../lib/libserial.so
cc -g -Wall -Wpedantic -Wextra -Iinclude -I../include -Wl,-rpath='$_ORIGIN/../lib' -o server src/server.c out/data.o ../lib/libtcp.so ../lib/libserial.so
gmake[1]: Leaving directory '/home/lbin/Documents/2020_RI/ex-serial'
```

Je copie ensuite les exécutoires sur mes machines virtuelles et puisque la structure de mes répertoires a changé, je remonte de dossier *lib* dans le répertoire racine de mes projet :



Comme pour le projet précédent, les liens symboliques ont été recréés avec la commande `ldconfig -n . :`

```

lbin@ri-client:~/lib$ ldconfig -n .
lbin@ri-client:~/lib$ ls -l
total 44
-rw-rw-r-- 1 lbin lbin 20248 Nov 28 12:55 libserial.so
lrwxrwxrwx 1 lbin lbin 12 Nov 28 12:57 libserial.so.1 -> libserial.so
lrwxrwxrwx 1 lbin lbin 13 Nov 15 18:17 libtcp.so.1 -> libtcp.so.1.0
-rw-rw-r-- 1 lbin lbin 21480 Nov 15 19:19 libtcp.so.1.0

lbin@ri-server:~/lib$ ldconfig -n .
lbin@ri-server:~/lib$ ls -l
total 44
-rw-rw-r-- 1 lbin lbin 20248 Nov 28 12:55 libserial.so
lrwxrwxrwx 1 lbin lbin 12 Nov 28 13:02 libserial.so.1 -> libserial.so
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
  
```

Et les fichiers *client* et *server* ont été rendus exécutoires :

```

lbin@ri-client:~/ex-serial$ chmod +x client
lbin@ri-client:~/ex-serial$ ls -l
total 28
-rwxrwxr-x 1 lbin lbin 27056 Nov 28 12:53 client

lbin@ri-server:~/ex-serial$ chmod +x server
lbin@ri-server:~/ex-serial$ ls -l
total 32
-rwxrwxr-x 1 lbin lbin 28696 Nov 28 12:55 server
  
```

6 Tests

6.1 Envoi d'une liste courte

Après avoir démarré le serveur (sans `sudo` puisque je n'utilise plus le port réservé au protocole echo), je lui envoie une liste de trois éléments via le client :

```
lbin@ri-client:~/ex-serial$ ./client 192.168.122.2 3
Data sent:
Node 1/3 {
    int_16: 30262
    int_32: 893624775
    int_64: -9223372036177124322
    float: 84801.007812
    double: 277809768.747864
    string: DD2KAucizNws3Jhq
}
Node 2/3 {
    int_16: 20573
    int_32: -1944373747
    int_64: 9223372036751106975
    float: 75385.656250
    double: 960775102.360873
    string: uuWLdXM
}
Node 3/3 {
    int_16: 1973
    int_32: 1461355512
    int_64: 9223372034735656790
    float: 19154.291016
    double: 984206314.568902
    string: iE
}
140 bytes have been successfully sent
lbin@ri-client:~/ex-serial$

lbin@ri-server:~/ex-serial$ ./server
[server] waiting for connections...
[server] connection received from 192.168.122.21...
[server:192.168.122.21] 3 nodes received (140 bytes)
Node 1/3 {
    int_16: 30262
    int_32: 893624775
    int_64: -9223372036177124322
    float: 84801.007812
    double: 277809768.747864
    string: DD2KAucizNws3Jhq
}
Node 2/3 {
    int_16: 20573
    int_32: -1944373747
    int_64: 9223372036751106975
    float: 75385.656250
    double: 960775102.360873
    string: uuWLdXM
}
Node 3/3 {
    int_16: 1973
    int_32: 1461355512
    int_64: 9223372034735656790
    float: 19154.291016
    double: 984206314.568902
    string: iE
}
[server:192.168.122.21] closing
```

Le serveur a bien reçu les 140 bytes envoyés par le client. Les données reçues et affichées correspondent bien à celles envoyées par le client. Le serveur ferme la connection avec le client et est prêt à en recevoir une nouvelle.

6.2 Envoi d'une liste vide

L'envoi d'une liste vide n'est simplement pas rendu possible par le programme du client :

```
lbin@ri-client:~/ex-serial$ ./client 192.168.122.2 0
argument list_size must be an integer [1, 65535]
```

Il en va de même pour les listes contenant un nombre d'éléments allant au-delà de 65535, un nombre négatif d'éléments, ou lorsqu'une chaîne de caractères est entrée à la place du nombre :

```
lbin@ri-client:~/ex-serial$ ./client 192.168.122.2 65536
argument list size must be an integer [1, 65535]
lbin@ri-client:~/ex-serial$ ./client 192.168.122.2 -3
argument list size must be an integer [1, 65535]
lbin@ri-client:~/ex-serial$ ./client 192.168.122.2 trois
argument list size must be an integer [1, 65535]
```

6.3 Envoi d'une liste de 65535 éléments

Par contre, l'envoi fonctionne bien avec une liste de 65535 éléments³ :

```
lbin@ri-client:~/ex-serial$ ./client 192.168.122.2 65535
Data sent:
Node 1/65535 {
    int_16: -12417
    int_32: -767207557
    int_64: -9223372036806649768
    float: 20291.546875
    double: 12950550.098724
    string: bnGW
}
Node 2/65535 {
    int_16: -17325
    int_32: 385795093
    int_64: 9223372034803944504
    float: 19367.917969
    double: 452086252.630857
    string: vTdNeJINZJ3yVWoBlmbMsxCVS
}
Node 3/65535 {
    int_16: 4905
    int_32: -1177059796
    int_64: 9223372035602411449
    float: 26998.847656
    double: -928650233.983754
    string: Tl
}
3504356 bytes have been successfully sent
lbin@ri-client:~/ex-serial$
```

```
lbin@ri-server:~/ex-serial$ ./server
[server] waiting for connections...
[server] connection received from 192.168.122.21...
[server:192.168.122.21] 65535 nodes received (3504356 bytes)
Node 1/65535 {
    int_16: -12417
    int_32: -767207557
    int_64: -9223372036806649768
    float: 20291.546875
    double: 12950550.098724
    string: bnGW
}
Node 2/65535 {
    int_16: -17325
    int_32: 385795093
    int_64: 9223372034803944504
    float: 19367.917969
    double: 452086252.630857
    string: vTdNeJINZJ3yVWoBlmbMsxCVS
}
Node 3/65535 {
    int_16: 4905
    int_32: -1177059796
    int_64: 9223372035602411449
    float: 26998.847656
    double: -928650233.983754
    string: Tl
}
[server:192.168.122.21] closing
```

3. Pour plus de facilité de lecture, pour ce test, j'ai modifié la fonction d'impression pour qu'elle n'affiche que les trois premiers éléments de la liste.