UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR | COLLÈGE STEE SCIENCES ET TECHNOLOGIES POUR L'ÉNERGIE ET L'ENVIRONNEMENT

Systèmes d'exploitation - TP n° 1 - La mémoire partagée

Samson Pierre <samson.pierre@univ-pau.fr>

04/02/2019

La mémoire partagée permet la communication inter-processus. Pour ce TP, nous avons un processus serveur et un processus client qui communiquent entre eux à travers un segment de mémoire partagée. Ce segment de mémoire partagée contient un message écrit par le client et lu par le serveur.

Le travail à rendre

L'archive à rendre doit contenir exactement ces fichiers :

```
$ tar tf tp1-mauro-gaio-samson-pierre.tar.xz | sort
tp1-mauro-gaio-samson-pierre/
tp1-mauro-gaio-samson-pierre/file.ftok
tp1-mauro-gaio-samson-pierre/Makefile
tp1-mauro-gaio-samson-pierre/shm-client.c
tp1-mauro-gaio-samson-pierre/shm-common.c
tp1-mauro-gaio-samson-pierre/shm-common.h
tp1-mauro-gaio-samson-pierre/shm-message.c
tp1-mauro-gaio-samson-pierre/shm-message.h
tp1-mauro-gaio-samson-pierre/shm-server.c
$
```

Les prénoms `mauro` et `samson` ainsi que les noms `gaio` et `pierre` sont à remplacer par les vôtres. Le fichier `file.ftok` est un fichier vide. Il constitue l'un des deux éléments permettant la création d'une clé (le deuxième élément est un identifiant de projet). Cette clé est nécessaire pour obtenir un segment de mémoire partagée. Le fichier `Makefile` contient les règles dont se sert le programme `make`. Les fichiers `shm-message.c` et `shm-message.h` contiennent respectivement les définitions et les déclarations pour le message. Les fichiers `shm-client.c` et `shm-server.c` contiennent respectivement les définitions pour le client et le serveur. Les fichiers `shm-common.c` et `shm-common.h` contiennent respectivement les définitions et les déclarations pour ce qui n'est pas spécifique au client, au serveur et au message.

La compilation du projet doit se passer ainsi :

```
$ make
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -c -o shm-common.o shm-common.c
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -c -o shm-message.o shm-message.c
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -o shm-client.out shm-client.c shm-common.o
shm-message.o
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -o shm-server.out shm-server.c shm-common.o
shm-message.o
$
```

Vous êtes autorisés à utiliser les fonctions suivantes :

```
void exit(int status);
int fprintf(FILE *stream, const char *format, ...);
key_t ftok(const char *pathname, int proj_id);
int getchar(void);
int getopt_long(int argc, char * const argv[], const char *optstring, const struct option
*longopts, int *longindex);
struct tm *localtime(const time_t *timep);
int printf(const char *format, ...);
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
int shmget(key_t key, size_t size, int shmflg);
unsigned int sleep(unsigned int seconds);
int strcmp(const char *s1, const char *s2);
char *strcpy(char *dest, const char *src);
size_t strlen(const char *s);
long int strtol(const char *nptr, char **endptr, int base);
time_t time(time_t *t);
int vfprintf(FILE *stream, const char *format, va_list ap);
```

Pour obtenir une bonne note, vous devez respecter ces règles :

- les fichiers doivent être encodés en UTF-8
- les fichiers ne doivent pas contenir de fautes d'orthographe
- les noms de fichiers doivent être ceux indiqués dans ce sujet
- le fichier `shm-message.h` doit être identique à celui du sujet
- les options de compilation doivent être celles précisées dans ce sujet
- les affichages à l'écran doivent correspondre à ceux du sujet
- la solution doit se rapprocher au maximum de ce qui est demandé dans ce sujet
- le code doit être correctement indenté
- le code doit être homogène concernant le nom des variables, les espaces, ...
- la mémoire allouée doit être correctement libérée
- les fonctions que vous utilisez dans votre code doivent figurer parmi celles autorisées dans ce sujet
- les valeurs de retour des fonctions pouvant échouer doivent être vérifiées afin de traiter les erreurs
- le traitement des erreurs consiste à afficher un message d'erreur personnalisé
- le format des messages d'erreur doit être identique à celui présenté dans ce sujet
- les messages d'erreur doivent être envoyés dans le flux d'erreur standard
- les autres messages seront envoyés dans le flux de sortie standard
- le code retourné par un processus rencontrant une erreur doit être 1
- le code retourné par un processus se terminant normalement doit être 0
- l'archive ne doit pas contenir d'autres fichiers que ceux demandés dans ce sujet
- l'archive doit être envoyée au plus tard le 08/03/2019 à 23:59
- l'archive doit être envoyée par e-mail à l'adresse `samson.pierre@univ-pau.fr`
- le sujet de l'e-mail doit être `SysEx - TP 1 - Mauro Gaio - Samson Pierre`
- les prénoms `Mauro` et `Samson` sont à remplacer par les vôtres
- les noms `Gaio` et `Pierre` sont à remplacer par les vôtres
- le travail est à réaliser impérativement en binôme

Le serveur

Le serveur doit proposer plusieurs options. L'option `-h` permet d'obtenir cet affichage :

```
$ ./shm-server.out -h
Usage: ./shm-server.out [OPTION]...
Receive messages from clients through shared memory.

Options:
        -h, --help
                display this help and exit
        -i, --key-proj-id=PROJ_ID
                set the key project identifier to PROJ_ID (the default value is "1")
        -p, --key-pathname=PATHNAME
                set the key pathname to PATHNAME (the default value is "file.ftok")
        -s, --seconds=SECONDS
                set the seconds between each try (the default value is "1", a value less than or equal to 0 enables the interactive mode where the input stream is read)
        -t, --times=TIMES
                set the number of times this program tries to receive a message (the default value is "-1", a negative value means repeat for ever)
        -v, --version
                output version information and exit

Report bugs to Mauro Gaio <mauro.gaio@univ-pau.fr> and Samson Pierre <samson.pierre@univ-pau.fr>.
$
```

Les prénoms `Mauro` et `Samson`, les noms `Gaio` et `Pierre` ainsi que les adresses e-mail `mauro.gaio@univ-pau.fr` et `samson.pierre@univ-pau.fr` sont à remplacer par les vôtres. L'option `-v` permet d'obtenir cet affichage :

```
$ ./shm-server.out -v
shm-server 20190204

Copyright (C) 2019 Mauro Gaio and Samson Pierre.

Written by Mauro Gaio <mauro.gaio@univ-pau.fr> and Samson Pierre <samson.pierre@univ-pau.fr>.
$
```

Les prénoms `Mauro` et `Samson`, les noms `Gaio` et `Pierre` ainsi que les adresses e-mail `mauro.gaio@univ-pau.fr` et `samson.pierre@univ-pau.fr` sont à remplacer par les vôtres. La version `20190204` est à remplacer par la version de votre programme. Lorsque le serveur est lancé sans option, il doit simplement attendre l'arrivée d'un message en lisant à intervalles réguliers dans le segment de mémoire partagée :

```
$ ./shm-server.out
proj_id = "1"
pathname = "file.ftok"
```

Lorsqu'un message est écrit par le client, le serveur doit alors l'afficher :

```
$ ./shm-server.out
proj_id = "1"
pathname = "file.ftok"
2019-02-04 08:04:51: Default name: This is the default message text
```

Pour arrêter le serveur, nous devons lui envoyer le signal `SIGINT`. Pour cela, nous pouvons presser les touches `Ctrl` et `C` :

```
$ ./shm-server.out
proj_id = "1"
pathname = "file.ftok"
2019-02-04 08:04:51: Default name: This is the default message text
^C
$
```

Réessayons de lancer le serveur :

```
$ ./shm-server.out
proj_id = "1"
pathname = "file.ftok"
./shm-server.out:shm-server.c:122: Unable to get the identifier of the System V shared memory
segment from the "0x0104421b" key.
$
```

La raison de l'erreur ci-dessus est que le segment de mémoire partagée obtenu n'a pas été détruit. En effet, avec la commande `ipcs`, nous pouvons voir que le segment est toujours présent. Pour le détruire, la commande `ipcrm` peut servir. Une fois le segment détruit, nous pouvons relancer le serveur. Cette fois, essayons avec l'option `-i` :

```
$ ./shm-server.out -i 2
proj_id = "2"
pathname = "file.ftok"
```

Le serveur est en attente de messages. Le client essaye d'envoyer un message mais n'y parvient pas parce qu'il doit indiquer le même identifiant de projet que le serveur. Une fois le serveur arrêté et le segment de mémoire partagée détruit, nous essayons l'option `-p` :

```
$ ./shm-server.out -p toto.tok
proj_id = "1"
pathname = "toto.tok"
./shm-server.out:shm-server.c:116: Unable to create the System V IPC key from the "toto.tok"
pathname and the "1" project identifier.
$
```

La raison de l'erreur ci-dessus est que le fichier `toto.tok` n'existe pas. Une fois le fichier `toto.tok` créé, nous pouvons relancer le serveur. Cette fois, essayons avec l'option `-s` :

```
$ ./shm-server.out -s 10
proj_id = "1"
pathname = "file.ftok"
2019-02-04 09:26:52: Default name: This is the default message text
2019-02-04 09:27:02: Default name: This is the default message text
```

Le client a essayé d'envoyer un message à chaque seconde, cependant le serveur est configuré cette fois pour lire un message toutes les `10` secondes. C'est pour cette raison que nous voyons seulement deux messages lus et affichés par le serveur sur une période de `10` secondes. Comme indiqué dans l'aide du programme, si nous donnons une valeur inférieure ou égale à `0` à cette option, nous passons en mode interactif :

```
$ ./shm-server.out -s 0
proj_id = "1"
pathname = "file.ftok"
Press the Enter key to continue...
Press the Enter key to continue...
2019-02-04 10:34:36: Default name: This is the default message text
Press the Enter key to continue...
```

Dans ce mode, le serveur attend que l'utilisateur presse la touche `Entrée`. Une fois la touche pressée, le serveur essaye de lire un message. Avant la première pression de cette touche, aucun client n'a envoyé de message ce qui explique pourquoi rien n'est affiché. Entre la première et la deuxième pression de cette touche, un client écrit un message ce qui

explique pourquoi un message est affiché. Une fois le serveur arrêté et le segment de mémoire partagée détruit, nous essayons l'option `-t` :

```
$ ./shm-server.out -t 3
proj_id = "1"
pathname = "file.ftok"
$
```

Le serveur essaye trois fois de lire un message puis s'arrête. Aucun client n'a envoyé de message durant ces trois tentatives donc aucun affichage de message n'est visible. Puisque cette fois, le serveur s'est arrêté par lui même, il s'est occupé de détruire le segment de mémoire partagé. Nous pouvons vérifier cette affirmation en lançant le serveur une nouvelle fois. Toutes les options que nous venons de voir peuvent être combinées et ceci dans n'importe quel ordre :

```
$ ./shm-server.out -t 3 -s 10 -p toto.tok -i 2
proj_id = "2"
pathname = "toto.tok"
$
```

D'autre part, chaque option courte possède sa version longue. Ainsi, la commande ci-dessus est équivalente à celle-ci :

```
$ ./shm-server.out --key-proj-id=2 --key-pathname=toto.tok --seconds=10 --times=3
proj_id = "2"
pathname = "toto.tok"
$
```

Le client

Comme le serveur, le client doit proposer plusieurs options. L'option `-h` permet d'obtenir cet affichage :

```
$ ./shm-client.out -h
Usage: ./shm-client.out [OPTION]...
Send a message to a server through shared memory.

Options:
        -h, --help
                display this help and exit
        -i, --key-proj-id=PROJ_ID
                set the key project identifier to PROJ_ID (the default value is "1")
        -n, --message-name=NAME
                set the message name to NAME (the default value is "Default name")
        -p, --key-pathname=PATHNAME
                set the key pathname to PATHNAME (the default value is "file.ftok")
        -s, --seconds=SECONDS
                set the seconds between each try (the default value is "1", a value less than or equal to 0 enables the interactive mode where the input stream is read)
        -t, --times=TIMES
                set the number of times this program tries to send a message (the default value is "1", a negative value means repeat for ever)
        -v, --version
                output version information and exit
        -x, --message-text=TEXT
                set the message text to TEXT (the default value is "This is the default message text")

Report bugs to Mauro Gaio <mauro.gaio@univ-pau.fr> and Samson Pierre <samson.pierre@univ-pau.fr>.
$
```

Les prénoms `Mauro` et `Samson`, les noms `Gaio` et `Pierre` ainsi que les adresses e-mail `mauro.gaio@univ-pau.fr` et `samson.pierre@univ-pau.fr` sont à remplacer par les vôtres. L'option `-v` permet d'obtenir cet affichage :

```
$ ./shm-client.out -v
shm-client 20190204

Copyright (C) 2019 Mauro Gaio and Samson Pierre.

Written by Mauro Gaio <mauro.gaio@univ-pau.fr> and Samson Pierre <samson.pierre@univ-pau.fr>.
$
```

Les prénoms `Mauro` et `Samson`, les noms `Gaio` et `Pierre` ainsi que les adresses e-mail `mauro.gaio@univ-pau.fr` et `samson.pierre@univ-pau.fr` sont à remplacer par les vôtres. La version `20190204` est à remplacer par la version de votre programme. Lorsque le client est lancé sans option, il doit simplement essayer d'envoyer un message et l'afficher :

```
$ ./shm-client.out
proj_id = "1"
pathname = "file.ftok"
```

```
4 ./shm-client.out:shm-client.c:155: Unable to get the identifier of the System V shared memory
  segment from the "0x0104421b" key.
5 $
```

La raison de l'erreur ci-dessus est que nous avons oublié de lancer le serveur. Une fois le serveur lancé, nous pouvons relancer le client et obtenir cet affichage :

```
1 $ ./shm-client.out
2 proj_id = "1"
3 pathname = "file.ftok"
4 2019-02-04 11:59:11: Default name: This is the default message text
5 $
```

Relançons le client avec l'option `-i` :

```
1 $ ./shm-client.out -i 2
2 proj_id = "2"
3 pathname = "file.ftok"
4 ./shm-client.out:shm-client.c:155: Unable to get the identifier of the System V shared memory
  segment from the "0x0204421b" key.
5 $
```

La raison de l'erreur ci-dessus est que nous avons indiqué un identifiant de projet différent de celui utilisé par le serveur. Maintenant, essayons l'option `-p` :

```
1 $ ./shm-client.out -p toto.tok
2 proj_id = "1"
3 pathname = "toto.tok"
4 ./shm-client.out:shm-client.c:149: Unable to create the System V IPC key from the "toto.tok"
  pathname and the "1" project identifier.
5 $
```

La raison de l'erreur ci-dessus est que le fichier `toto.tok` n'existe pas. Une fois le fichier `toto.tok` créé, nous pouvons relancer le client :

```
1 $ ./shm-client.out -p toto.tok
2 proj_id = "1"
3 pathname = "toto.tok"
4 ./shm-client.out:shm-client.c:155: Unable to get the identifier of the System V shared memory
  segment from the "0x010406d3" key.
5 $
```

La raison de l'erreur ci-dessus est que nous avons indiqué un fichier différent de celui utilisé par le serveur. Maintenant, essayons l'option `-s` :

```
1 $ ./shm-client.out -s 10
2 proj_id = "1"
3 pathname = "file.ftok"
4 2019-02-04 12:11:55: Default name: This is the default message text
5 $
```

Le client se comporte de la même façon que si l'option n'avait pas été utilisée. La raison est que le temps de `10` secondes est celui entre chaque message, or nous n'avons qu'un seul message à envoyer. Afin de voir l'intérêt de l'option `-s`, nous devons la combiner avec l'option `-t` :

```
1 $ ./shm-client.out -s 10 -t 2
2 proj_id = "1"
3 pathname = "file.ftok"
4 2019-02-04 13:17:35: Default name: This is the default message text
5 2019-02-04 13:17:45: Default name: This is the default message text
6 $
```

Ainsi, nous voyons qu'il y a un intervalle de `10` secondes entre les `2` messages envoyés. Revenons à l'option `-s` afin de passer en mode interactif grâce à une valeur inférieure ou égale à `0` :

```
1 $ ./shm-client.out -s 0
2 proj_id = "1"
3 pathname = "file.ftok"
4 2019-02-04 14:20:58: Default name: This is the default message text
5 Press the Enter key to continue...
```

Dans ce mode, le client attend que l'utilisateur presse la touche `Entrée`. Une fois la touche pressée, le client va s'arrêter puisque nous n'avons qu'un seul message à envoyer. Cependant, si nous combinons cette option avec l'option `-t`, le programme pourra éventuellement envoyer un nouveau message une fois la touche pressée. Le client ne doit pas écrire un nouveau message dans le segment de mémoire partagée si ce segment contient un message non lu par le serveur. Voyons donc ce qui se passe si nous écrivons plus rapidement nos messages que le serveur ne parvient à les lire :

```
$ ./shm-client.out -s 0 -t -1
proj_id = "1"
pathname = "file.ftok"
2019-02-04 15:23:31: Default name: This is the default message text
Press the Enter key to continue...
./shm-client.out:message.c:11: Unable to copy the message because the target message is not empty.
./shm-client.out:shm-client.c:169: Unable to copy the message.
$
```

Ici, le premier message est correctement envoyé mais ce n'est pas le cas du deuxième message. Maintenant, essayons les options `-n` et `-x` :

```
$ ./shm-client.out -n "My message name" -x "My message text"
proj_id = "1"
pathname = "file.ftok"
2019-02-04 16:27:08: My message name: My message text
$
```

Nous voyons que ces options permettent respectivement de personnaliser le nom et le texte du message à envoyer. Lorsque le nom ou le texte du message est trop long, une erreur doit être affichée :

```
$ ./shm-client.out -x "This message text is a bit too long to be accepted. That is sad."
./shm-client.out:shm-message.c:59: Unable to set the "This message text is a bit too long to be accepted. That is sad." message text because its "65" size is greater than "64".
./shm-client.out:shm-client.c:134: Unable to set the "This message text is a bit too long to be accepted. That is sad." message text.
$
```

Toutes les options que nous venons de voir peuvent être combinées et ceci dans n'importe quel ordre. D'autre part, chaque option courte possède sa version longue.

Le message

Voici le contenu du fichier d'en-tête `shm-message.h` contenant les déclarations pour le message :

```c
/**
 * \file shm-message.h
 */
#ifndef SHM_MESSAGE_H
#define SHM_MESSAGE_H
/**
 * The size of the message name.
 */
#define SHM_MESSAGE_NAME_SIZE 16
/**
 * The size of the message text.
 */
#define SHM_MESSAGE_TEXT_SIZE 64
/**
 * A message.
 */
typedef struct
{
        char name[SHM_MESSAGE_NAME_SIZE]; /**< The message name. */
        char text[SHM_MESSAGE_TEXT_SIZE]; /**< The message text. */
} shm_message_t;
/**
 * Copies a source message to a target message.
 * \param message_source The source message.
 * \param message_target The target message.
 * \return -1 on error (i.e., if the target message is not empty, the shm_message_set_name function returns -1 or the shm_message_set_text function returns -1), else 0.
 */
int shm_message_copy(shm_message_t message_source, shm_message_t *message_target);
/**
 * Empties a message (i.e., sets the message name and the message text to an empty string).
 * \param message The message.
 */
void shm_message_empty(shm_message_t *message);
/**
 * Checks if a message is empty (i.e., checks if the message name and the message text are an empty string).
 * \param message The message.
 * \return 1 if the message is empty, else 0.
 */
int shm_message_is_empty(shm_message_t message);
```

```
40 /**
41  * Prints a message using the "YYYY-MM-DD HH:MM:SS: name: text" format.
42  * \param message The message.
43  */
44 void shm_message_print(shm_message_t message);
45 /**
46  * Sets the name of a message.
47  * \param message The message.
48  * \param name The name.
49  * \return -1 on error (if the message name size is greater than SHM_MESSAGE_NAME_SIZE), else 0.
50  */
51 int shm_message_set_name(shm_message_t *message, const char *name);
52 /**
53  * Sets the text of a message.
54  * \param message The message.
55  * \param text The text.
56  * \return -1 on error (if the message text size is greater than SHM_MESSAGE_TEXT_SIZE), else 0.
57  */
58 int shm_message_set_text(shm_message_t *message, const char *text);
59 #endif
```