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° 3 - Les tubes

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

04/02/2019

Les tubes permettent la communication inter-processus. Pour ce TP, nous avons un processus parent et un processus fils qui communiquent entre eux à travers un tube. Ce tube peut contenir un ou plusieurs messages écrits par le fils qui seront lus par le parent.

Le travail à rendre

L'archive à rendre doit contenir exactement ces fichiers :

```
$ tar tf tp3-mauro-gaio-samson-pierre.tar.xz | sort
tp3-mauro-gaio-samson-pierre/
tp3-mauro-gaio-samson-pierre/Makefile
tp3-mauro-gaio-samson-pierre/pipe-common.c
tp3-mauro-gaio-samson-pierre/pipe-common.h
tp3-mauro-gaio-samson-pierre/pipe-main.c
tp3-mauro-gaio-samson-pierre/pipe-message.c
tp3-mauro-gaio-samson-pierre/pipe-message.h
$
```

Les prénoms `mauro` et `samson` ainsi que les noms `gaio` et `pierre` sont à remplacer par les vôtres. Le fichier `Makefile` contient les règles dont se sert le programme `make`. Les fichiers `pipe-message.c` et `pipe-message.h` contiennent respectivement les définitions et les déclarations pour le message. Le fichier `pipe-main.c` contient les définitions pour le parent et le fils. Les fichiers `pipe-common.c` et `pipe-common.h` contiennent respectivement les définitions et les déclarations pour ce qui n'est pas spécifique au parent, au fils et au message.

La compilation du projet doit se passer ainsi :

```
$ make
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -c -o pipe-common.o pipe-common.c
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -c -o pipe-message.o pipe-message.c
gcc -std=c89 -pedantic -Wall -Werror -D_GNU_SOURCE -g -o pipe-main.out pipe-main.c pipe-common.o
pipe-message.o
$
```

Vous êtes autorisés à utiliser les fonctions suivantes :

```
int close(int fd);
void exit(int status);
pid_t fork(void);
int fprintf(FILE *stream, const char *format, ...);
int getchar(void);
int getopt_long(int argc, char * const argv[], const char *optstring, const struct option
*longopts, int *longindex);
pid_t getpid(void);
struct tm *localtime(const time_t *timep);
int pipe2(int pipefd[2], int flags);
int printf(const char *format, ...);
ssize_t read(int fd, void *buf, size_t count);
unsigned int sleep(unsigned int seconds);
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);
pid_t wait(int *status);
ssize_t write(int fd, const void *buf, size_t count);
```

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 `pipe-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 26/04/2019 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 3 - 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 programme

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

```
$ ./pipe-main.out -h
Usage: ./pipe-main.out [OPTION]...
Send and receive messages through a pipe.

Options:
        -h, --help
                display this help and exit
        -r, --seconds-for-receiving=SECONDS
                set the seconds between each try to receive a message (the default value is "1", a value less than or equal to 0 enables the interactive mode where the input stream is read)
        -s, --seconds-for-sending=SECONDS
                set the seconds between each try to send a message (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-for-receiving=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)
        -u, --times-for-sending=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 :

```
$ ./pipe-main.out -v
pipe-main 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 programme est lancé sans option, il doit créer un tube, créer un processus fils et attendre l'arrivée d'un message en lisant à intervalles réguliers dans le tube :

```
$ ./pipe-main.out
```

Ce processus fils, quant à lui, doit écrire un message dans le tube et l'afficher. Lorsque le parent trouve un message dans le tube, il doit l'afficher :

```
$ ./pipe-main.out
written: 2019-02-04 08:48:05: 1331: This is the default message text
read: 2019-02-04 08:48:06: 1331: This is the default message text
```

Le processus fils s'est arrêté après avoir envoyé son message. Cependant, le processus parent est toujours actif. Pour arrêter le programme, nous devons lui envoyer le signal `SIGINT`. Pour cela, nous pouvons presser les touches `Ctrl` et `C` :

```
$ ./pipe-main.out
written: 2019-02-04 08:48:05: 1331: This is the default message text
read: 2019-02-04 08:48:06: 1331: This is the default message text
^C
$
```

Une fois le programme arrêté, nous essayons l'option `-r` :

```
$ ./pipe-main.out -r 10
written: 2019-02-04 09:54:56: 1353: This is the default message text
read: 2019-02-04 09:55:06: 1353: This is the default message text
```

Le temps entre chaque message reçu est maintenant de `10` secondes. Puisque nous n'avons qu'un seul message écrit par le fils, nous ne pouvons pas comparer le temps entre chaque message pour vérifier le fonctionnent. Afin de voir l'intérêt de l'option `-r`, nous devons la combiner avec l'option `-u` :

```
$ ./pipe-main.out -r 10 -u 2
written: 2019-02-04 10:57:18: 1361: This is the default message text
written: 2019-02-04 10:57:19: 1361: This is the default message text
read: 2019-02-04 10:57:28: 1361: This is the default message text
read: 2019-02-04 10:57:38: 1361: This is the default message text
```

Ainsi, nous voyons qu'il y a un intervalle de `10` secondes entre les `2` messages reçus. 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 :

```
$ ./pipe-main.out -r 0
Press the Enter key to read...
written: 2019-02-04 11:11:02: 1423: This is the default message text

read: 2019-02-04 11:11:06: 1423: This is the default message text
Press the Enter key to read...

Press the Enter key to read...
```

Dans ce mode, le parent attend que l'utilisateur presse la touche `Entrée`. Une fois la touche pressée, le parent essaye de lire un message. Avant la première pression de cette touche, le fils a écrit un message dans le tube ce qui explique pourquoi un message écrit est affiché. Après la première pression de cette touche, le parent a lu un message dans le tube ce qui explique pourquoi un message lu est affiché. Entre la première et la deuxième pression de cette touche, aucun message n'est écrit par le fils ce qui explique pourquoi rien n'est affiché. Une fois le programme arrêté, nous essayons l'option `-s` :

```
$ ./pipe-main.out -s 10
written: 2019-02-04 12:22:58: 1448: This is the default message text
read: 2019-02-04 12:22:59: 1448: This is the default message text
```

Le temps entre chaque message envoyé est maintenant de `10` secondes. Puisque nous n'avons qu'un seul message écrit par le fils, nous ne pouvons pas comparer le temps entre chaque message pour vérifier le fonctionnent. Afin de voir l'intérêt de l'option `-s`, nous devons la combiner avec l'option `-u` :

```
$ ./pipe-main.out -s 10 -u 2
written: 2019-02-04 13:24:55: 1457: This is the default message text
read: 2019-02-04 13:24:56: 1457: This is the default message text
written: 2019-02-04 13:25:05: 1457: This is the default message text
read: 2019-02-04 13:25:05: 1457: This is the default message text
```

Ainsi, nous voyons qu'il y a un intervalle de `10` secondes entre les `2` messages envoyés. 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 :

```
$ ./pipe-main.out -s 0
written: 2019-02-04 14:50:25: 1533: This is the default message text
Press the Enter key to write...
read: 2019-02-04 14:50:26: 1533: This is the default message text
```

Dans ce mode, le fils attend que l'utilisateur presse la touche `Entrée`. Une fois la touche pressée, le fils va s'arrêter puisque nous n'avons qu'un seul message à envoyer. Cependant, si nous combinons cette option avec l'option `-u`, le programme pourra éventuellement envoyer un nouveau message une fois la touche pressée. Une fois le programme arrêté, nous essayons l'option `-t` :

```
$ ./pipe-main.out -t 3
written: 2019-02-04 15:55:56: 1554: This is the default message text
read: 2019-02-04 15:55:57: 1554: This is the default message text
$
```

Le programme se comporte de la même façon que si l'option n'avait pas été utilisée. La raison est que malgré les `3` tentatives pour recevoir, nous n'avons qu'un seul message envoyé dans le tube. Afin de voir l'intérêt de l'option `-t`, nous devons la combiner avec l'option `-u` :

```
$ ./pipe-main.out -t 3 -u 3
written: 2019-02-04 16:56:10: 1556: This is the default message text
written: 2019-02-04 16:56:11: 1556: This is the default message text
read: 2019-02-04 16:56:11: 1556: This is the default message text
written: 2019-02-04 16:56:12: 1556: This is the default message text
read: 2019-02-04 16:56:12: 1556: This is the default message text
$
```

Nous pouvons constater que seulement deux messages sont lus sur les `3` messages écrits. Le fils a pourtant bien écrit trois fois dans le tube. Cependant, lorsque le parent lit pour la première fois dans le tube, il n'y trouve aucun message parce qu'il a été plus rapide que le fils. Ainsi, les messages lus affichés correspondent à la deuxième et à la troisième lecture dans le tube par le parent. Il reste donc un message non lu dans le tube. Maintenant essayons l'option `-u` :

```
$ ./pipe-main.out -u -1
written: 2019-02-04 17:00:50: 1566: This is the default message text
read: 2019-02-04 17:00:51: 1566: This is the default message text
written: 2019-02-04 17:00:51: 1566: This is the default message text
read: 2019-02-04 17:00:52: 1566: This is the default message text
written: 2019-02-04 17:00:52: 1566: This is the default message text
read: 2019-02-04 17:00:53: 1566: This is the default message text

```

Comme indiqué dans l'aide du programme, puisque nous donnons une valeur négative à cette option, le fils envoie des messages dans le tube pour une durée infinie. Le parent, quant à lui, lit ces messages et les affiche. Dans un autre terminal, grâce à la commande `ps`, nous pouvons vérifier quel processus est le parent et quel processus est le fils :

```
$ ps -HC pipe-main.out
  PID TTY          TIME CMD
 1565 pts/1    00:00:00 pipe-main.out
 1566 pts/1    00:00:00   pipe-main.out
$
```

Maintenant, essayons l'option `-x` :

```
$ ./pipe-main.out -x "My message text"
written: 2019-02-04 18:06:22: 1586: My message text
read: 2019-02-04 18:06:23: 1586: My message text

```

Nous voyons que cette option permet de personnaliser le texte du message à envoyer. Lorsque ce texte est trop long, une erreur doit être affichée :

```
$ ./pipe-main.out -x "This message text is a bit too long to be accepted. That is sad."
./pipe-main.out:pipe-message.c:31: 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".
./pipe-main.out:pipe-main.c:185: 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 :

```
$ ./pipe-main.out -x "My message text" -u 3 -t 3 -s 10 -r 10
written: 2019-02-04 19:25:21: 1665: My message text
written: 2019-02-04 19:25:31: 1665: My message text
read: 2019-02-04 19:25:31: 1665: My message text
written: 2019-02-04 19:25:42: 1665: My message text
read: 2019-02-04 19:25:42: 1665: My message text
$
```

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

```
$ ./pipe-main.out --message-text="My message text" --seconds-for-receiving=10
--seconds-for-sending=10 --times-for-receiving=3 --times-for-sending=3
written: 2019-02-04 20:28:42: 1672: My message text
written: 2019-02-04 20:28:52: 1672: My message text
read: 2019-02-04 20:28:52: 1672: My message text
read: 2019-02-04 20:29:02: 1672: My message text
written: 2019-02-04 20:29:02: 1672: My message text
$
```

Le message

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

```c
/**
 * \file pipe-message.h
 */
#ifndef PIPE_MESSAGE_H
#define PIPE_MESSAGE_H
#include <sys/types.h> /* for pid_t */
/**
 * The size of the message text.
 */
#define PIPE_MESSAGE_TEXT_SIZE 64
/**
 * A message.
 */
typedef struct
{
        pid_t pid; /**< The process ID. */
        char text[PIPE_MESSAGE_TEXT_SIZE]; /**< The message text. */
} pipe_message_t;
/**
 * Prints a message using the "prefix: YYYY-MM-DD HH:MM:SS: pid: text" format.
 * \param message The message.
 * \param prefix The prefix.
 */
void pipe_message_print(pipe_message_t message, const char *prefix);
/**
 * Sets the text of a message.
 * \param message The message.
 * \param text The text.
 * \return -1 on error (if the message text size is greater than PIPE_MESSAGE_TEXT_SIZE), else 0.
 */
int pipe_message_set_text(pipe_message_t *message, const char *text);
/**
 * Sets the process ID of a message.
 * \param message The message.
 * \param pid The process ID.
 * \return -1 on error (if the message pid value is less than 0), else 0.
 */
int pipe_message_set_pid(pipe_message_t *message, pid_t pid);
#endif
```