TP2 : Programmation Système (partie 2)

Blottière Paul <blottiere.paul@gmail.com>

Table of Contents

1. Généralités 1
    1.1. embsys 1
    1.2. Simulateur GPS 1
    1.3. Mémoire partagée : shm_writer 2
2. Exercice 1 : mémoire partagée et sémaphore 2
    2.1. Les questions 2
    2.2. Ce qu'il faut retenir 4
3. Exercice 2 : thread et mutex 4
    3.1. Les questions 5
    3.2. Ce qu'il faut retenir 6
4. Exercice 3 : sockets, timerfd et plugin 6
    4.1. Les questions 6
    4.2. Ce qu'il faut retenir 7

1. Généralités

1.1. embsys

L'ensemble des cours, exemples, PDF et TP sont disponibles sur le dépôt github https://github.com/pblottiere/embsys.

Si vous voulez cloner entièrement le dépôt :

```
$ git clone https://github.com/pblottiere/embsys
```

Si vous voulez cloner le dépôt mais avoir simplement les labs dans votre répertoire de travail :

```
$ git clone -n https://github.com/pblottiere/embsys --depth 1
$ cd embsys
$ git checkout HEAD labs
```

Si vous souhaitez mettre à jour un dépôt que vous avez préalablement cloné :

```
$ cd embsys
$ git pull
```

Si vous n'avez pas **git**, téléchargez le ZIP sur la page d'acceuil de **embsys**.

Le TP d'aujourd'hui se trouve ici : https://github.com/pblottiere/embsys/labs/2_sysprog_part2.

1.2. Simulateur GPS

Dans le cadre de ce TP, nous allons réutiliser le simulateur GPS étudié lors du premier TP. Tout d'abord, il faut compiler le simulateur :

```
$ cd labs/gps
$ make ok
```

Nous lancerons le simulateur plus tard dans le TP grâce au script **run.sh**.

1.3. Mémoire partagée : shm_writer

Comme vu dans le TP précédent, le simulateur GPS envoie les données sur un port virtuel à travers /dev/pts/X. Le binaire **shm_writer** présent dans le répertoire 2_sysprog_part2/src/shm_writer lit les données envoyées par le simulateur GPS sur le port virtuel et écrit en mémoire partagée les informations **latitude**, **longitude** et **time**.

Pour compiler le binaire **shm_writer** :

```
$ cd labs/2_sysprog_part2/src/shm_writer
$ make
```

Les données en mémoire partagée sont représentées via la structure SHDATA définie dans le header shm_writer/shdata.h :

```
struct SHDATA
{
    float latitude;
    float longitude;
    int time;
};
```

2. Exercice 1 : mémoire partagée et sémaphore

Dans le cadre de ce premier exercice, nous allons naviguer dans le code source des répertoires 2_sysprog_part2/src/shm_writer et 2_sysprog_part2/src/shm_reader.

Le but de l'exercice est de lire la mémoire partagée remplie par le binaire **shm_writer** et d'afficher ces informations à l'écran toutes les secondes en modifiant le code de **shm_reader**.

2.1. Les questions

Tout d'abord, ouvrez un terminal et lancez le simulateur GPS :

```
$ cd embsys/labs/gps
$ sh run.sh
PTTY: /dev/pts/X
```

Ouvrez un second terminal et lancez le binaire **shm_writer** :

```
$ cd embsys/labs/2_sysprog_part2/src/shm_writer
$ ./shm_writer -p /dev/pts/X -s myshm -l lock
```

Question 1 : *Selon vous, à quoi correspond le paramètre **myshm** indiquée via l'option **-s** de **shm_writer**? Et **lock**?*

Question 2 : *Où peut-on trouver la représentation du segment de mémoire partagée sur le système de fichiers?*

Question 3 : *Faites un schéma bloc des différents éléments mis en jeu.*

Placez vous dans le répertoire 2_sysprog_part2/src/shm_writer.

Question 4 : *En étudiant la fonction **hndopen** implémentée dans le fichier **handler.c**, décrivez les fonctions utilisées pour gérer le segment de mémoire partagée.*

Question 5 : *Quelle fonction utilise le paramètre **myshm** passé en ligne de commande?*

Question 6 : *Quel flag en particulier indique une **création** de segment et pas seulement une ouveture en lecture/écriture?*

Placez vous maintenant dans le répertoire 2_sysprog_part2/src/shm_reader.

Question 7 : *Modifiez la fonction **hndopen** implémentée dans **handler.c** pour ouvrir le segment de mémoire partagée en lecture/écriture. Les champs **shm**, **shmfd** et **shdata** de la structure **handlers** passée en paramètre doivent être mis à jour.*

En voulant compiler le binaire **shm_reader**, vous devez obtenir ceci :

```
$ make
/tmp/ccawsZJY.o: In function `hndopen':
handler.c:(.text+0x66): undefined reference to `shm_open'
collect2: error: ld returned 1 exit status
```

Pour utiliser les fonctions liées à la mémoire partagée, la librairie **realtime** est nécessaire.

Question 8 : *En s'inspirant de **shm_writer/Makefile**, modifiez le fichier **shm_reader/Makefile** pour que la compilation passe.*

Une fois la compilation réalisée avec succès, exécutez **shm_reader**. Vous devez obtenir :

```
$ make
* ./shm_reader -s myshm -l lock
time : XXXXXX

time : XXXXXX

...
```

Question 9 : *Expliquez l'évolution de la valeur **time** affichée à l'écran.*

Dans le main de **shm_reader**, le paramètre **handlers** est défini en tant que variable globale.

Question 10 : *Quelle est la particularité d'une variable globale? Comment fait-on pour définir une telle variable?*

Question 11 : *Dans le **main** de shm_reader.c, complétez la boucle **while** de la fonction shmreader afin que les champs **latitude** et **longitude** du segment de mémoire partagée **handlers.shdata** soient affichés en même temps que le champs **time**.*

Question 12 : *Inspirez vous de la fonction **decode_frame** de shm_writer/shm_writer.c pour lever/ baisser le sémaphore lors de la lecture du segment de mémoire partagée dans la fonction **shmreader** de shm_reader.c.*

Recompilez et exécutez **shm_reader**. Vous devez obtenir :

```
$ make
$ ./shm_reader -s myshm -l sem
time : XXXX
latitude : XXXX
longitude : XXXX

time : XXXX
latitude : XXXX
longitude : XXXX

...
```

Modifiez la fonction **sem_open** utilisée dans la fonction **hndopen** et définie dans **shm_writer/handler.c** de cette manière :

```
// handlers->sem = sem_open(opts.sem, O_RDWR|O_CREAT, S_IRUSR|S_IWUSR, 1);
handlers->sem = sem_open(opts.sem, O_RDWR|O_CREAT, S_IRUSR|S_IWUSR, 0);
```

Recompilez **shm_writer**. Relancez **shm_writer** puis **shm_reader**.

Question 13 : *Que se passe t-il côté **shm_reader**? Pourquoi? Quel effet a eu la modification précédente?*

Rétablissez l'appel à **sem_open** du fichier **shm_writer/handler.c** ainsi :

```
handlers->sem = sem_open(opts.sem, O_RDWR|O_CREAT, S_IRUSR|S_IWUSR, 1);
```

Lorsque l'utilisateur interrompt le processus via Ctrl-C, la fonction **hndclose** n'est jamais appellée et les handlers d'entrées/sorties ne sont pas fermés correctement.

Question 14 : *Mettez en place un gestionnaire de signaux qui appelle la fonction **hndclose** lors d'une interruption. Inspirez vous de **shm_writer/shm_writer.c**.*

Question 15 : *Comparez la fonction **hndclose** définie dans **shm_writer/handler.c** avec celle définie dans **shm_reader/handler.c**. Quelle différence voyez vous? Expliquez.*

2.2. Ce qu'il faut retenir

- la notion de variable globale
- les fonctions permettant de gérer les segments de mémoire partagée : **shm_open**, **ftruncate**, **mmap** et **shm_unlink**.
- l'utilisation de **sem_open**, **sem_wait** et **sem_post**.
- la mise en place d'un gestionnaire de signaux via **sigaction**.

3. Exercice 2 : thread et mutex

Dans le cadre de ce deuxième exercice, nous allons naviguer dans le code source du répertoire 2_sysprog_part2/src/converter.

Les coordonnées latitude et longitude écrites en mémoire partagée par le binaire **shm_writer** sont définies en degré et minute (norme NMEA). Le but est ici de mettre en place un thread chargé de convertir ces coordonnées au format decimal et de les rendre disponibles à travers une structure globale accessible par tous les threads du binaire **converter** et protégée par mutex.

3.1. Les questions

Le simulateur **gps** et **shm_writer** doivent toujours être actifs pour la suite de l'exercice.

Placez vous dans le répertoire 2_sysprog_part2/src/converter.

Dans un premier temps, compilez et exécutez **converter** :

```
$ cd 2_sysprog_part2/src/converter
$ make
$ ./converter -s myshm -l lock
time : 0
latitude : 0
longitude : 0

time : 0
latitude : 0
longitude : 0
...
```

La fonction affichant periodiquement les coordonnées decimales est exécutée dans un premier thread par la fonction **display** définie dans converter.c.

Dans la suite de l'exercice, le but est de mettre en place un deuxième thread chargé de mettre à jour les coordonnées decimales qui sont affichées par le premier thread, le tout vérouillé par un mutex.

Question 16 : *Faites un schéma bloc des éléments à mettre en jeu pour atteindre notre but (simulateur GPS et shm_writer compris).*

Question 17 : *Décrivez les deux fonctions utilisées dans converter.c pour la mise en place du thread1 affichant les coordonnées.*

Question 18 : *Utilisez la fonction **pthread_create** afin d'exécuter la fonction **convert** dans un deuxième thread.*

Comme dans l'exercice précédent, la structure projetée en mémoire partagée est accessible à travers la variable globale **handlers.shdata**.

Question 19 : *Modifiez la fonction **convert** afin que la variable globale **decimal_coord** soit mise à jour avec les coordonnées decimales. Vous pouvez utiliser la fonction **to_decimal** pour la conversion.*

Compilez et exécutez :

```
$ make
$ ./converter -s myshm -l lock
time : XXXX
latitude : XXXX
longitude : XXXX
```

```
time : XXXX
latitude : XXXX
longitude : XXXX
...
```

Question 20 : *Utilisez la variable mutex **mut** passée en paramètre des fonctions **convert** et **display** pour verrouiller l'accès à **decimal_coord** entre les 2 threads.*

3.2. Ce qu'il faut retenir

- les fonctions de base permettant de gérer les threads : **pthread_create**, et **pthread_join**.
- les mutex avec **pthread_mutex_lock** et **pthread_mutex_unlock**.

4. Exercice 3 : sockets, timerfd et plugin

Le but de l'exercice est d'envoyer periodiquement les cordonnées de latitude et longitude sur le réseau. L'envoi se fera par UDP.

4.1. Les questions

Le simulateur **gps** et **shm_writer** doivent toujours être actifs pour la suite de l'exercice. Pour cet exercice, on doit aussi compiler et exécuter le binaire **net_reader** dans un terminal séparé :

```
$ cd 2_sysprog_part2/src/net_reader
$ make
$ ./net_reader -p 8888
```

Dans le cadre de ce troisième exercice, nous allons naviguer dans le code source du répertoire **2_sysprog_part2/src/forwarder**.

Question 21 : *À quoi correspond ici le paramètre 8888? Quel outil pouvez vous utiliser pour vérifier qu'une connexion réseau est bien ouverte?*

Question 22 : *En analysant le fichier handler.c, indiquez les fonctions liées à la programmation de socket.*

Question 23 : *Toujours dans handler.c, quels flags indiquent une connexion UDP? Quelle est la particularité de l'UDP par rapport à du TCP?*

Question 24 : *Encore dans handler.c, à quoi sert le premier paramètre de la fonction **inet_aton**?*

Question 25 : *Dans la fonction **forward** du fichier forwarder.c, utilisez la fonction **sendto** pour envoyer la variable **message** sur le réseau.*

Compilez puis exécutez correctement le binaire **forwarder**. Vous devriez obtenir ceci du côté **net_reader** :

```
$ ./net_reader -p 8888
received: 000000/49.3055/8.3795
received: 223222/49.3065/8.3805
...
```

Question 26 : *Faites un schéma bloc de tous éléments mis en jeu pour arriver ici.*

4.2. Ce qu’il faut retenir

- les fonctions liées à la programmation réseau