Compte Rendu TP2 : Programmation Système (partie 2)

Contents

1 Exercice 1 : mémoire partagée et sémaphore

Réponse 1 :

lock correspond à l'identifiant du sémaphore protégant la mémoire partagée.

myshm est l'identifiant de la mémoire partagée.

Réponse 2 : En rentrant dans la console

```
ls -l /dev/shm/myshm
```

On retrouve une représentation de la mémoire partagée (*ici on affiche ses droits*).

Réponse 3 : *Faîtes un schéma bloc des différents éléments mis en jeu.*

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

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

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

Réponse 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.

Réponse 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.

Réponse 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

...
```

Réponse 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.

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

Réponse 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**.*

Réponse 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**.

Réponse 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.

Réponse 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**.*

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

1.1 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**.

2 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.

2.1 Les questions

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

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.

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

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

Réponse 18 : *Utilisez la fonction **pthread_create** et **pthread_detach** 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**.

Réponse 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
...
```

Réponse 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.*

2.2 Ce qu'il faut retenir

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

3 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 soit par TCP soit par UDP via une option passée en ligne de commande. La sélection du mode TCP ou UDP impliquera le chargement dynamique de la librairie correspondante (notion de plugin).

3.1 Les question

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

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

Dans un premier temps, compilez et exécuter