Travaux Dirigés Programmation Système: Feuille 5

Informatique 2ème année. ENSEIRB 2014/2015

—*Mathieu Faverge* - `mfaverge@enseirb.fr` —

Signaux et mémoire partagée

▸**Exercice 1.** *Création d'un gestionnaire capturant le signal SIGUSR1*

Écrire un programme qui :

- *affiche son propre pid,*
- *installe un gestionnaire de signaux avec `signal(2)` pour le signal SIGUSR1 qui affiche le numéro du signal reçu.*
- *se met en attente d'un signal*
- *restaure le gestionnaire par défaut.*
- *** N'attendre que le signal SIGUSR1 à l'aide de l'interface avancée (`sigaction`,`sigsuspend`)*

Lancer le programme et tester l'effet des différents signaux avec la commande shell `kill(1)` sur l'exécution du programme. Essayer notamment l'enchaînement SIGSTOP, SIGCONT, SIGUSR1, ou le signal SIGKILL. Que se passe t'il ?

▸**Exercice 2.** *Communication par `mmap` :*

Le but de cet exercice est de reprendre l'exercice précédent pour faire communiquer 2 processus au travers d'une zone mémoire partagée par `mmap`

- *Créez un fichier texte contenant au moins 128 caractères.*
- *Reprenez le code de l'exercice précédent dans un programme `receveur.c` qui fait un `mmap(2)` de ce fichier, écrit son PID au début de la zone mémoire partagée, et se met en attente avec `pause()` (ou avec `sigsupsend` si implémenté) sur le signal SIGUSR1. Lorsque le signal est reçu, il affiche le contenu du segment de mémoire partagée en plus du message prédéfini.*
- *Testez ce programme en envoyant le signal SIGUSR1 à l'aide de la commande `kill(1)`*
- *Ecrivez un second programme `expediteur.c` qui fait un `mmap(2)` de ce fichier, lit le PID au début de la zone mémoire partagée, puis écrit dans le segment de mémoire ce qu'il lit sur l'entrée standard et envoie au processus dont il a lu le PID le signal SIGUSR1.*
- *Que se passe-t-il si plus de 128 caractères sont écrits dans le segment de mémoire partagée ? Quelle est la taille de ce segment ? (écrivez dedans jusqu'à ce qu'une erreur se produise).*

▸**Exercice 3.** *Protections de `mmap` anonyme*

1. *Écrire un programme `exo3.c` qui :*
   - *Appelle `mmap(NULL, sizeof(int), PROT, MAP_PRIVATE | MAP_ANON, -1, 0)` et stocke la valeur de retour dans un pointeur d'entier `p`.*
   - *Imprime sur l'erreur standard le pointeur avec le formatage `%p` de `fprintf(3)`.*
   - *Imprime sur l'erreur standard la valeur de `*p` avec le formatage `%d`.*
   - *Affecte la valeur 42 à la destination du pointeur.*
   - *Imprime sur l'erreur standard la valeur du pointeur avec le formatage `%d` avant de terminer.*

   On passera la variable de pre-processeur PROT en argument du compilateur de la façon suivante :

   ```
   gcc -Wall -DPROT=PROT_NONE -o exo3 exo3.c
   gcc -Wall -DPROT=PROT_READ -o exo3_r exo3.c
   gcc -Wall -DPROT=PROT_WRITE -o exo3_w exo3.c
   gcc -Wall -DPROT=PROT_EXEC -o exo3_x exo3.c
   gcc -Wall -DPROT="PROT_EXEC | PROT_WRITE" -o exo3_wx exo3.c
   ```

   Exécuter ces programmes et expliquer ce qu'il se passe en vous aidant de la page de manuel de `mmap(2)`. Expliquer ce qui se passe.

▸ **Exercice 4.** *Partage de fonctions*

On souhaite partager entre plusieurs processus le code d'une fonction, sans que cette fonction fasse partie de l'exécutable des processus. On va pour cela reproduire le fonctionnement d'une bibliothèque partagée, chargée dynamiquement (`man dlopen, dlsym`*). Pour cela, on écrira d'abord un programme qui copie une fonction dans une zone mémoire partagée, puis un autre programme qui partage cette zone mémoire et appelle la fonction :*

1. *Ecrire un programme* `load.c` *avec une fonction* `add(int, int)` *qui renvoie la somme de ses deux argument.*
2. *Dans le main, créer avec* `write(2)` *un fichier* `libadd.a` *de 2048 octets (s'il n'existe pas), mapper ce fichier dans une zone mémoire partagée à l'aide de* `mmap(2)`*. Pour cela, on utilisera les flags* `PROT_WRITE|PROT_EXEC` *et* `MAP_SHARED`*. Puis copier à l'aide de la fonction* `memcpy(3)` *les 2048 premiers octets correspondant à la fonction* `add` *dans la zone de mémoire partagée créée à l'aide de* `mmap(2)`*.*
3. *Écrire un programme* `use.c` *qui crée une zone mémoire partagée, synchronisée avec le fichier* `libadd.a`*. On utilisera les flags* `PROT_EXEC` *et* `MAP_SHARED` *pour pouvoir exécuter le code de cette zone mémoire. Affecter un pointeur* `f` *de type* `int (*)(int, int)` *avec l'adresse de cette zone mémoire.*
4. *Appeler la fonction* `f(42,12)` *et afficher le résultat. Que se passe-t-il si vous lancez plusieurs programmes* `use` *dans différents shells ?*