Programmation Système, TD 3

Préambule

Rappel : Comme pour les TD précédents, un nouveau répertoire est à créer pour ce TD-ci, répertoire au sein duquel sera créé pour chaque exercice un nouveau répertoire associé nommé `Exo_xx` où `xx` désignera le numéro de l'exercice. Chacun de ces répertoires aura à contenir deux nouveaux répertoires :

— l'un, appelé `Include`, sera destiné à recevoir scripts, en-têtes, fichiers de paramètrage
— l'autre, appelé `WorkDir`, sera le répertoire de travail contenant codes sources et `makefile`

Une archive servant de canevas pour ce TD, intitulé `CanevasTD_Signaux.tar.gz`, est téléchargeable. Afin de ne pas occuper inutilement de la place dans vos répertoires, lorsque vous finirez ce TD

1. penser à effacer tous les exécutables et `.o`
   — soit *via* la commande `make clean` accessible à partir du Makefile de votre répertoire de travail ;
   — soit à partir de la simple commande `rm */*/*.{o,a,exec}`, toujours dans votre répertoire de travail ; cette procédure sera répétée à chaque TD.
2. compresser le contenu de chaque répertoire associé à un exercice sous la forme d'une archive `.tar.gzip` *via* (par exemple) la commande

   ```
   tar cvf - exo_01 | gzip -9 - > exo_01.tar.gz
   ```

   pour l'exercice 01.

Des questions vous sont posées tout au long du TP : Les réponses sont à rendre sur une feuille de papier libre comportant votre nom et votre groupe.

Affichages des signaux standard et déroutement

On considère ici les trois formes possibles de comportement associées aux signaux :

— le comportement standard établi par défaut avec la directive `SIG_DFL`
— la possibilité d'ignorer l'arrivée d'un signal avec la directive `SIG_IGN`
— le déroutement vers un traitant de signal `Derout` ; un tel déroutement est opéré avec la fonction système `sigaction()` qui met en place la `struct`

```
struct sigaction {
  void (* sa_handler)(int);
  void (* sa_sigaction) (int, siginfo_t *, void *);
  sigset_t sa_mask;
  int sa_flags;
  void (* sa_restorer) (void);
}
```

On notera que les champs `sa_handler` et `sa_sigaction` ne s'utilisent pas simultanément, et que les normes de programmation POSIX déclarent le dernier champs

```
void (* sa_restorer) (void);
```

obsolète et ne devant pas être utilisé (voir `man 2 sigaction`)

Exercice 1. Cet exercice permet l'affichage des signaux standard et le déroutement de signaux à partir d'un wrapper simplifié de la fonction `sigaction()` simulant le fonctionnement de la fonction système `signal()`, plus ancienne, et dont l'emploi littéral est déprécié. L'affichage du rôle associé à chacun des signaux peut être obtenu à partir d'un tableau de NTCTS appelé `_sys_siglist`, déclaré dans `/usr/include/signal.h`. La bibliothèque `<string.h>` met par ailleurs à disposition la fonction `strsignal()` qui interface l'accès au tableau en question. *Question 1 : Quelle sont les commandes à taper dans le shell, permettant d'obtenir une description de cette fonction ?*

Dans l'ordre :

1. ajouter dans l'espace de noms anonyme d'`exo_01.cxx` un traitant de signal appelé `Derout()` qui affiche sur la sortie standard le numéro et le nom du signal reçu
2. à partir de la description de `sigaction()`, étudier le wrapper correspondant déclaré et défini respectivement dans les fichiers `nsSysteme.h` et `nsSysteme.cxx`
3. dans le fichier `nsSysteme.h` ajouter le type `typedef void (*sighandler_t)(int)` vu dans le cours et étudier le profil de la fonction `Signal()`
4. écrire alors le pseudo-wrapper simplifié

   ```
   sighandler_t Signal(int numsig, sighandler_t Traitant)
   ```

   dans `nsSysteme.cxx` en utilisant `sigaction()` comme dans le cours
5. dans le fichier `exo_01.cxx`, faire prendre au `main()` un unique argument qui ne pourra être qu'une des lettres 'P', 'I' et 'D' (signifiant respectivement Particulier, Ignorer ou Défaut)
6. dans le `try ... catch` du `main()`, pour tous les signaux `NumSig` de `1` à `31` :
   si `NumSig` n'est pas un des signaux interdits pour le déroutement (`SIGKILL`, `SIGSTOP`, `SIGCONT`) selon la valeur de l'argument ('P', 'I' ou 'D') dérouter les signaux vers le bon traitant de signal et afficher le choix opéré sur la sortie standard :
   — si l'argument passé est 'P', dérouter `NumSig` vers le traitant particulier `Derout`
   — si l'argument passé est 'I', ignorer `NumSig` en utilisant la directive `SIG_IGN`
   — si l'argument passé est 'D', rétablir le traitement par défaut associé au signal `NumSig`
   — sinon annoncer que l'option est inconnue
7. à la suite du traitement précédent, ouvrir une boucle infinie

Après compilation et édition de liens, lancer `exo_01.exec P` et tester avec des signaux en provenance du clavier. *Question 2 : Quelles combinaisons de touches au clavier envoient respectivement* `SIGINT`*,* `SIGTSTP`*,* `SIGQUIT` *?*

Ouvrir un autre terminal et taper y les commandes

```
ps x | head -1; ps x | grep exo_01.exec | grep -v grep
```

pour récupérer le PID du processus. On peut alternativement taper

```
ps x | grep exo_01.exec | grep -v grep | awk '{print $1;}'
```

pour récupérer uniquement le PID. Envoyer ensuite des signaux en tapant dans le shell `kill -<symboleDuSignal>` et pour terminer, envoyer `SIGKILL` Relancer alors `exo_01.exec I` et essayer de lui envoyer des signaux. *Question 3 : Le programme affiche-t'il quelque chose pour marquer l'arrivée des signaux envoyés ? Pourquoi ? Question 4 : Comment procéder pour le tuer ?*

Restauration automatique du traitant par défaut

La fonction `sigaction()` permet une manipulation très fine des aspects liés à la réception des signaux. En l'occurence, la valeur `SA_RESETHAND` utilisée dans le champ `sa_flags` de la struct `sigaction` installée avec la fonction `sigaction()` permet d'effectuer la restauration automatique du traitant par défaut ("reset handler" en anglais).

Exercice 2. Dans cet exercice, on manipule les drapeaux de la struct `sigaction`

- dans ans le fichier `exo_02.cxx`, placer dans l'espace de noms anonyme un traitant de signal appelé Derout() qui affiche sur la sortie standard son début, le nom et le numéro du signal reçu et sa fin
- dans le try ... catch du `main()` déclarer une structure `struct sigaction Act` et initialiser tous ses champs
- initialiser le champ `sa_flags` à la valeur `SA_RESETHAND`
- initialiser le champ `sa_mask` à l'aide du wrapper de `sigemptyset()`
- initialiser le champ `sa_handler` avec le traitant (c'est-à-dire le pointeur de fonction) `Derout`
- réaliser un appel à `Sigaction()` pour mettre en place la structure `Act` et rendre effectif le déroutement de `SIGINT`
- à la suite du traitement précédent, ouvrir une boucle infinie

Après compilation et édition de liens, tester en envoyant le signal `SIGINT` deux fois. *Question 4 : Que se passe-t-il si le signal est envoyé plus de deux fois ? Pourquoi ? Question 5 : Quels autres drapeaux peut-on mettre en place pour la fonction* `sigaction()` *? A quoi servent-ils ?*