Fonctions de base d'Unix concernant les signaux

©A. Dragut
Université Aix-Marseille I.U.T.d'Aix en Provence - Département Informatique
Dernière mise à jour :18/10/2012

Rappel structure des répertoires

- *allez dans sous-répertoire **TP***
- *allez dans le sous-répertoire **tpsystem***
- *vous deviez y avoir créé le sous-répertoire **tpfile**, correspondant à l'unité de TP précédente*
- *maintenant créez **tpsignal***
- *allez dans **tpsignal***
- *créez le répertoire **exo_01***
- *allez dans **exo_01***
- *copiez-y (avec l'option **-r**), le répertoire **include** du TP sur les fichiers*
- *créez le répertoire **dirsignal***
- *copiez dans **dirsignal** le fichier **Makefile** du TP sur les fichiers (donc depuis son `dir...`)*
- *pour chaque exercice, vous créérez un nouveau répertoire **exo_0x** avec **x** le numéro de l'exercice, et, sauf indication contraire, vous y copierez le répertoire **include** de l'exercice précédent.*
- *Pour faire de la place dans vos répertoires*
  Faites un petit coup de `make clean` dans le `dir*` de chaque `exo_*`. Compressez le contenu de chaque répertoire d'exercice avec `tar` et `gzip`, avec la commande suivante (exemple pour `exo_01`) :

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

  Attention aux espaces autour des tirets seuls, ces tirets demandent l'utilisation de l'entrée, respectivement sortie standard (au lieu d'utiliser des noms de fichiers). Ainsi on peut enchaîner ces commandes et tout déposer directement dans `exo_01.tar.gz` Par contre, le `-9` est bien ensemble, cette option demande la meilleure compression.

Sommaire du document

- ***Travail à faire***
- *Explications, rappels du cours, petites astuces, etc.*

Liste des fonctions et concepts système étudiés

Dans cette série de dix exercices on explore les différentes manières de générer, recevoir et traiter les signaux, à l'aide des fonctions ***, `kill()`, `sigaction()`, `sigsuspend()`, `select()`. On étudie les traitements possibles lors de l'arrivée d'un signal.***

Liste des exercices

- ***exo_01 liste des `signaux`, leur `déroutement`***
- ***exo_02 restauration automatique du traitant par défaut***
- ***OPTIONNEL exo_03 restauration de l'action précédente; blocage pendant l'exécution du traitant***
- ***exo_04 code sensible à protéger contre les signaux***
- ***exo_05 blocage des signaux pendant l'exécution du programme. section critique***
- ***OPTIONNEL exo_06 attendre avec `sigsuspend()` et noter l'arrivée d'un signal***
- ***OPTIONNEL exo_07 utilisation du signal SIGALRM et multiplexage des entrées/sorties avec `select()`***

Bon courage !

---

exo_01 liste des `signaux`, leur `déroutement`

Dans cet exercice

vous découvrez les trois traitements simples des signaux:

- le comportement standard (par défaut): `SIG_DFL`
- ignorer des signaux: `SIG_IGN`
- le déroutement vers un traitant de signal Derout en utilisant un wrapper simplifié de la fonction `sigaction()`

Vous les testez avec des signaux clavier (Ctrl+touches) et avec `kill` du shell.

Quoi de neuf

Les noms des signaux se trouvent dans un tableau de NTCTS appelé **`_sys_siglist`**, déclaré dans **`/usr/include/signal.h`**. On peut également utiliser **`strsignal()`** , déclarée dans **`<string.h>`**.
On déroute un signal vers un traitant de signal Derout avec la fonction `sigaction()`. Cette fonction met en place une structure de type `struct sigaction`

```
struct sigaction {
```

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

Les champs sa_handler et sa_sigaction ne s'utilisent pas simultanément. Les normes de programmation système POSIX disent que le champ sa_restorer est obsolète et ne doit pas etre utilisé.
Rappel de cours: un traitant de signal est de type **`sighandler_t`** et doit toujours prendre qu'un seul entier (le numéro du signal l'ayant fait appeler), et ne doit rien rendre.

Travail à faire

- **Saugardez le fichier modelmain.cxx en tant que `exo_01.cxx`.**
- **Mettez dans l'espace de noms anonyme un traitant de signal appelé `Derout()` qui affiche sur la sortie standard le nom et le numéro du signal reçu.**
- **Récupérez les fichiers nsSysteme.h et nsSysteme.cxx, étudiez bien le `wrapper` de `sigaction()`, au besoin en vous servant de *`man sigaction`***
- **Dans le fichier `nsSysteme.h` rajoutez le type `typedef void (*sighandler_t)(int)` vu dans le cours et regardez le profil de la fonction `Signal()`**
- **Écrivez le pseudo-wrapper simplifié `sighandler_t Signal(int numsig, sighandler_t Traitant)` dans `nsSysteme.cxx`, en utilisant `sigaction()` comme dans le cours.**
- **Dans le fichier `exo_01.cxx`, faites le `main()` prendre un seul argument, une lettre parmi 'P', 'I' et 'D' (particulier, ignorer ou défaut)**
- **Dans le fichier `exo_01.cxx` 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 déroutez les signaux vers le bon traitant de signal et annoncez votre choix sur la sortie standard:**
        - **si c'est 'P', déroutez `NumSig` vers le traitant particulier `Derout`**
        - **si c'est 'I', ignorez `NumSig` en utilisant `SIG_IGN`**
        - **si c'est 'D', remettez le traitement par défaut au signal `NumSig`**
        - **sinon annoncez que l'option est inconnue (par exemple)**
  - **se mettre dans une boucle infinie**
- **Compilez.**
- **Lancez `exo_01.run P` et testez avec des signaux en provenance du clavier. Identifiez les combinaisons de touches du clavier qui envoient**
  - **`SIGINT`**
  - **`SIGTSTP`**
  - **`SIGQUIT`**
- **Allez dans une autre fenêtre et écrivez**

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

  pour récupérer le PID de votre procéssus. On peut également faire

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

  pour récupérer directement seulement le PID.
- **Envoyez des signaux en faisant *`kill -<symboleDuSignal> <PID>`* , et pour terminer, envoyez `SIGKILL` .**
- **Lancez `exo_01.run I` et ressayez de lui envoyer des signaux.**

- **QUESTION: Votre programme affiche-t-il quelque chose pour marquer l'arrivée des signaux envoyés? Pourquoi? Comment peut-on le tuer?**

exo_02 restauration automatique du traitant par défaut

Dans cet exercice

Le but est d'apprendre à utiliser les drapeaux de la structure `sigaction`.

Quoi de neuf

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

Travail à faire

- **Dans le fichier `exo_02.cxx`, mettez 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 son fin.**
- **Dans le fichier `exo_02.cxx` dans le `try ... catch` du main()**
  - **préparez une structure (initialiser tous ses champs!) `struct sigaction Act;`**
    - **initialisez le champ `sa_flags` à la valeur `SA_RESETHAND`**
    - **initialisez le champ `sa_mask` à l'aide du wrapper de `sigemptyset()`**
    - **initialisez le champ `sa_handler` avec le traitant (c.à.d. le pointeur de fonction) `Derout`**
  - **faites un appel à `Sigaction()` pour mettre en place la structure `Act` et rendre effectif le déroutement du `SIGINT`**
  - **rentrez dans une boucle infinie**
  - **Compilez et testez en envoyant le signal `SIGINT` deux fois**
  - **QUESTION: Que se passe-t-il si vous envoyez plus de deux fois le signal? Pourquoi?**
  - **QUESTION: Lisez le man 2 sigaction(). Quels autres drapeaux peut-on mettre en place pour la fonction sigaction() ? Que font-t-ils?**

OPTIONNEL exo_03 restauration de l'action précédente; blocage pendant l'exécution du traitant

Dans cet exercice

On déroute `SIGINT` d'abord vers un traitant de signal, puis vers un autre, tout en bloquant des signaux pendant les exécutions des traitants. Les masques des signaux bloqués pendant les exécutions des traitants sont differents. À la fin, le programme restaure la première action, depuis la sauvegarde qu'il a bien faite.

Quoi de neuf

Lors de son appel, la fonction `sigaction()` fournit l'ancienne action qui était en place dans la zone pointée par son dernier paramètre. On peut ainsi la sauvegarder, pour une restauration ultérieure. On rappelle que bloquer un signal veut bien dire que le système note son arrivée, mais ne la dévoile au processus que lorsque le débloquage intervient.

Travail à faire

- **Dans le fichier `exo_03.cxx` dans l'espace de nom anonyme écrivez deux traitants de signal `Derout1(),Derout2()`.**
  - **tous les deux affichent un message spécifique de début, dorment 5 secondes, puis affichent un message de fin spécifique**
  - **à la fin du traitant `Derout2()` on restaure la valeur de l'ancienne action `OldAct` au moyen de la fonction `Sigaction()`, puis on affiche le message de fin. (`OldAct` doit etre une variable globale)**
- **Dans le fichier `exo_03.cxx` dans le `try...catch` du `main()`**
  - **préparez le masque du premier appel de `sigaction()` étant donne que:**
    - **le signal `SIGINT` est dérouté vers le traitant `Derout1`.**
    - **pendant l'exécution du traitant `Derout1` le signal `SIGTSTP` doit être bloqué (utilisez le champ `sa_mask` de l'action**
  - **appellez le wrapper `Sigaction()`de la fonction `sigaction()` pour mettre en place l'action desirée et récupérer l'ancienne action**
  - **attendez l'arrivée d'un signal avec la fonction `pause()`,**
  - **préparez le masque du deuxième appel de `sigaction()` étant donne que:**
    - **le signal `SIGINT` est dérouté vers le traitant `Derout2`.**
    - **pendant l'exécution du traitant `Derout2` le signal `SIGTSTP` NE doit pas être bloqué (utilisez le champ `sa_mask` de l'action**
  - **rentrez dans une boucle infinie.**
- **Compilez et testez en envoyant trois fois le signal `SIGINT` au processus.**
- **Vérifiez que le traitant du signal a bien été restauré.**
- **QUESTION: Quel type de variable est OldAct? Pourquoi?**
- **QUESTION: Quelle est la suite des instructions pour vérifier que `SIGTSTP` est bien bloqué pendant le premier déroutement, mais pas entre le second déroutement et la restauration?**

exo_04 -- exo_05 blocage des signaux pendant l'exécution du programme; section critique

Dans cet exercice

vous résolvez un problème d'interférence en bloquant le signal `SIGINT` avant d'entrer dans l'activité sensible et en le débloquant en sortant.

Quoi de neuf

On doit préparer une structure de type `sigset_t` pour pouvoir appeller `Sigprocmask()`. Cette structure est à manipuler avec les wrappers des fonctions: `sigemptyset(), sigaddset(), sigdelset(), sigismember()` La solution générale pour proteger une opération sensible est de la placer dans une **section critique** et de la protéger par un mécanisme dépendant du langage utilisé (verrous, sémaphores, moniteurs,

objets protégés ou tâches). Dans le cas présent, le fait de bloquer le signal suffit.

- `Sigprocmask(SIG_SETMASK,&Masque,&MasquePrecedent)`
- `// section critique`
- `Sigprocmask(SIG_SETMASK,&MasquePrecedent,0)`

ou bien

- `Sigprocmask(SIG_BLOCK,&Masque,&MasquePrecedent)`
- `// section critique`
- `Sigprocmask(SIG_UNBLOCK,&Masque,0)`

si on veut seulement rajouter notre blocage au traitement des signaux déjà mis en place par le `MasquePrecedent`.

Travail à faire

- **Récuperez le fichier `exo_04.cxx` et copiez le dans `exo_05.cxx`**
  Ce programme affiche dans une boucle infinie les éléments d'un vecteur d'entiers, avec un `sleep()` dans la boucle, pour temporiser. À chaque envoi de `SIGINT` son traitant rajoute au vecteur une valeur rentrée au clavier, et le trie.
- **Compilez et testez tous les cas possibles :**
  - **aucun signal envoyé : le contenu du vecteur doit s'afficher,**
  - **un signal `SIGINT` est envoyé pendant l'attente de 10 secondes : la saisie est effectuée et l'affichage est immédiat (la réception du signal a interrompu la fonction `sleep()`),**
  - **un signal `SIGINT` est envoyé pendant l'affichage**
- **QUESTION: Quel type de nombre doit-on saisir pour perturber l'affichage quand un signal `SIGINT` est envoyé pendant l'affichage?**
- **Il y a un problème d'interférence entre la saisie et l'affichage. On considere l'affichage une opération "sensible", qu'on ne desire pas qu'elle soit interrompue avant qu'elle soit terminée. On va la protéger.**
- **Ajoutez**
  - **la préparation d'un masque de signaux dans lequel on met à 1 le signal `SIGINT`. Le masque est une structure de type `sigset_t` à manipuler avec les wrappers: `Sigemptyset(), Sigaddset(), Sigdelset(), Sigismember()`**
  - **le blocage effectif du signal `SIGINT` en appellant la fonction `Sigprocmask()`.**
  - **le déblocage effectif du signal `SIGINT` au moyen de la fonction `Sigprocmask()` en sortant le signal `SIGINT` du masque des signaux à bloquer**
  - **Compilez et testez tous les cas possibles en envoyant le signal `SIGINT` au moyen de la commande `kill` à partir d'une autre fenêtre. :**
    - **aucun signal envoyé : au bout de 10 secondes, le contenu du vecteur doit s'afficher,**
    - **un signal `SIGINT` est envoyé pendant l'attente de 10 secondes : la saisie est effectuée et l'affichage est immédiat (la réception du signal a interrompu la fonction `sleep()`),**
    - **un signal `SIGINT` est envoyé pendant l'affichage : la saisie n'est possible qu'après terminaison de l'affichage. Le mécanisme fonctionne bien comme désiré.**
    - **un signal `SIGINT` est envoyé pendant la saisie: la saisie n'est pas affectée, mais le traitant est exécuté immédiatement après. En effet, le système bloque automatiquement un signal pendant l'exécution de son traitant, qui n'est donc pas récursif sauf si le *flag* `SA_NODEFER` est positionné. Cependant, l'arrivée du signal est mémorisée dans le masque des signaux pendants par**

un bit positionné à 1 (signal pendant), et traitée dès que possible.
    - **plusieurs signaux `SIGINT` sont envoyés pendant la saisie: un seul signal est traité après la fin de la saisie. Cela est dû au fait qu'un seul bit peut être positionné par le système lors de la réception du premier signal, qui ne peut pas mémoriser d'autres occurences du signal `SIGINT`. Toutes les autres occurrences sont donc définitivement perdues.**
- **QUESTION: Que se passe-t-il si on envoye plusiers signaux `SIGINT` pendant que le signal n'est pas bloqué?**

OPTIONNEL exo_06 attendre et noter l'arrivée d'un signal

Dans cet exercice

vous modifiez le programme de l'exercice précédent. Le nouveau traitant de signal enregistre l'occurrence des signaux, et c'est l'application qui décide du moment où elle veut les prendre en compte.

Quoi de neuf

La communication entre le traitant de signal et le reste du programme se fait au moyen d'une variable globale. Pour en assurer la cohérence, on a besoin d'un type spécial de variable, dont la valeur peut être accédée tout en ayant la garantie de ne jamais pouvoir être interrompus par un signal pendant l'accès. On utilisera le type `sig_atomic_t` . On a également besoin d'un qualifieur spécial qui empêche le compilateur d'optimiser trop -- `volatile` -- pour annoncer que la variable est susceptible d'être modifiée par un dispositif **externe au programme** (comme un traitant d'interruption ou de signal, etc.).
Enfin, on utilise aussi la fonction `sigsuspend(const sigset_t *)`. Un appel `sigsuspend(&Masque)` fait, **de manière atomique par rapport aux signaux, c.à.d ininterruptible par un signal**, l'équivalent de :

- `Sigprocmask(SIG_SETMASK,&Masque,&MasquePrecedent)`
- `pause()`
- `Sigprocmask(SIG_SETMASK,&MasquePrecedent,0)`

Travail à faire

- **Copiez le fichier `exo_05.cxx` dans `exo_06.cxx`.**
- **Ajoutez à l'espace de noms anonyme deux variables de type `sig_atomic_t`, qui joueront le rôle de variables booléennes, `Fin` et `Saisie`, qualifiées `volatile` et initialisées à 0 (faux). `Fin` et `Saisie` doivent être mis à 1 (vrai) respectivement lors de l'arrivée des signaux `SIGQUIT` et `SIGINT`.**
- **Ajoutez avant la boucle**
    - **déroutez les signaux `SIGQUIT` et `SIGINT`.**
    - **bloquez-les (étapes: préparez comme il faut un premier `Masque` masque de signaux et appeler la fonction `Sigprocmask()`)**
    - **préparez un nouveau masque de signaux, appelons-le `MasqueVide`, qui est vide. (si on veux tester avec autres signaux que SIGINT et SIGQUIT il faut bloquer les autres signaux)**
- **Modifiez la boucle infinie pour qu'elle se termine lorsque le "booléen" `Fin` est "vrai".**
- **A l'intérieur de la boucle, testez si aucun des deux signaux n'est arrivé de la manière suivante:**
    - **dans une autre petite boucle tant qu'aucun des deux signaux n'est arrivé**
        - **appelez le wrapper de `::sigsuspend(&MasqueVide)`**

    - **si `Saisie` faux on sort de la boucle (l'appel de `::Sigsuspend()` est revenu, donc au moins un des deux signaux est arrivé, donc Saisie faux implique Fin vrai)**
    - **mettez `Saisie` à faux**
    - **lisez du clavier l'entier**
    - **débloquez les signaux (étapes: préparez comme il faut un premier `Masque` masque de signaux et appeler la fonction `Sigprocmask()`)**
    - **faites le rajout**
    - **bloquez les signaux à nouveau avec le masque des signaux `Masque`**
    - **triez le veteur et faites l'affichage**
- **Compilez et testez.**
- **QUESTION: Que se passe-t-il si on envoye SIGINT et SIGQUIT pendant l'affichage?**

OPTIONNEL exo_07 SIGALRM et multiplexage des entrées/sorties avec `select()`

Dans cet exercice

vous écrivez un programme qui attend avec `select()` un événement clavier et qui l'affiche ensuite avec le temps écoulé. Si aucun événement clavier n'as pas lieu pendant un délai prefixé un signal **`SIGALRM`** sera renvoyé au programme.

Quoi de neuf

La fonction **`select()`** est appelée **fonction de multiplexage d'entrées/sorties**. Rappelons que les "fichiers" (au sens large d'Unix) peuvent être classés en deux catégories selon que la fonction système **`read()`** renvoie 0 lorsqu'ils sont vides (c'est le cas des fichiers disques "normaux") ou qu'elle bloque (normalement) le processus qui l'appelle lorsque le "fichier" est vide (c'est le cas du "fichier" clavier, des *sockets*, des *pipes*).
Rappelons aussi que la fonction **`select()`** est bloquante et donc le process l'appellant est endormi jusqu'à ce qu'au moins un événement attendu arrive, ou que le délai fixé en dernier paramètre arrive à expiration. Ce genre d'attente s'appelle **`une attente passive`**. Une description détaillée des arguments et du comportement se trouve dans le cours.

Travail à faire

- **Écrivez dans le fichier `nsSysteme.h` le *wrapper* de la fonction système `select()`.**
- **Dans le fichier `exo_07.cxx` dans le `try...catch` du `main()` mettez les actions suivantes :**
    - **déclarez un masque de descripteurs de fichier `fd_set` et initialiser le avec `FD_ZERO`**
    - **déclarez et initialisez une `struct timeval` initiale à cinq secondes.**
    - **dans une boucle de `nfois` tours**
        - **déclarez un autre `struct timeval` et initialisez-le avec celui initial** (sur beaucoup de distributions `select()` modifie ce paramètre)
        - **rajouter le descripteur de l'entrée standard dans le masque de descripteurs de fichiers à surveiller en lecture. Le masque est de type `fd_set`, donc pour un rajout on utilise `FD_SET`** (même raison -- `select()` modifie ce paramètre)
        - **faites un appel de `Select()` qui fait le processus attendre en dormant soit une frappe au clavier soit l'écoulement du délai**
            - **Si l'événement est une frappe au clavier** (à tester le retour de `Select()` et le masque de descripteurs de fichiers avec `FD_ISSET`)

        - **lisez le caractère frappé (soit avec ">>", soit avec la fonction membre `get()` de la classe `istream`)**
        - **afficher le caractère et délai restant** (il faut diviser par 1 000 000 le membre `tv_usec` pour l'additionner au `tv_sec`
    - **Si la sortie de la fonction `Select()` est due à l'épuisement du délai affichez un message et sortez la boucle.**
- **Compilez et testez.**

---

[1] On appelle *builtin command* du *c-shell* une commande intégrée au *shell*, qui est effectuée préférentiellement à une commande Unix de même nom. Par exemple, `kill` ou `login` sont à la fois des *builtin commands* du *c-shell* et des commandes Unix (de niveau 1).

Solutions

Solution exo_01

exo_01.cxx
Makefile
nsSysteme.cxx
CExc.h
INCLUDE_H
nsSysteme.h

Solution exo_02

exo_02.cxx
Makefile
nsSysteme.cxx
CExc.h
INCLUDE_H
nsSysteme.h

Solution exo_03

exo_03.cxx
Makefile
nsSysteme.cxx
CExc.h
INCLUDE_H
nsSysteme.h

Solution exo_04

exo_04.cxx
Makefile
nsSysteme.cxx

CExc.h
INCLUDE_H
nsSysteme.h

Solution exo_05

exo_05.cxx
Makefile
nsSysteme.cxx
CExc.h
INCLUDE_H
nsSysteme.h

Solution exo_06

exo_06.cxx
Makefile
nsSysteme.cxx
CExc.h
INCLUDE_H
nsSysteme.h

Solution exo_07

exo_07.cxx
Makefile
nsSysteme.cxx
CExc.h
INCLUDE_H
nsSysteme.h
