Signaux (suite)

Ressources : pour ce TP, comme pour les suivants, vous pourrez vous appuyer sur

- Le polycopié intitulé « Systèmes d'exploitation : Unix », qui fournit une référence généralement suffisante sur la sémantique et la syntaxe d'appel des différentes primitives de l'API Unix.
- Les pages du manuel en ligne (commande `man`), et plus particulièrement les sections 2 et 3.

Rendu à la fin de la séance

Comme pour les TPs précédents, vous archiverez à la fin de la séance votre travail en utilisant la commande `make archive`. Le fichier nommé `minishell-votreidentifiant.tar` sera alors à charger sous Moodle dans la section rendu, dans la zone qui correspond à votre groupe de TD. Cette archive nous permet de voir votre progression au cours des séances de TP et de détecter d'éventuel problème que nous n'aurions pas vu en séance.

Minishell : contrôle du fils par les commandes au clavier

Pour cette partie, nous souhaitons pouvoir contrôler les processus fils en utilisant l'appui au clavier de `ctrl-C` (touches `control` et `C` appuyées en même temps) et `ctrl-Z`. Le traitement normal est d'arrêter ou suspendre le processus en avant-plan.

Etape 10 (Test de la frappe au clavier de `ctrl-C` et `ctrl-Z`) Terminez l'étape 10, si ce n'est pas déjà fait. Lorsqu'on appuie sur `ctrl-C` (respectivement `ctrl-Z`), le signal `SIGINT` (respectivement `SIGTSTP`) est envoyé au processus en cours. La terminaison (respectivement la suspension) du processus `minishell` est alors demandée. Testez ce comportement.
Que se passe-t-il lorsque le `minishell` a lancé :

- une commande en avant plan, par exemple `sleep 50` ?
- une commande en arrière plan, par exemple `sleep 50 &` ?

Etape 11 (Gestion de la frappe au clavier de `ctrl-C` et `ctrl-Z`) Comme nous avons pu le remarquer, lorsqu'on frappe `ctrl-C` (ou `ctrl-Z`), les signaux sont transmis à la fois à `minishell` et à l'ensemble de ses fils. La première chose que nous allons traiter est la non-terminaison ou la non-suspension du processus `minishell`. Plusieurs solutions s'offrent à nous. **Elles sont toutes à réaliser et à tester.**

Etape 11.1 (Changer le traitement des signaux `SIGINT` et `SIGTSTP`) Définissez un traitement, qui réaffiche le prompt par exemple, et associez le aux signaux `SIGINT` et `SIGTSTP`. Testez la frappe de `ctrl-C` puis de `ctrl-Z`, avec un processus en arrière plan et un processus en avant plan.

Etape 11.2 (Ignorer les signaux `SIGINT` et `SIGTSTP`) Ignorer un signal revient à associer le traitement `SIG_IGN` à ce signal. Pour reprendre le traitement par défaut, il suffit d'associer le traitement `SIG_DFL`. Testez ici aussi la frappe de `ctrl-C` puis de `ctrl-Z`, avec un processus en arrière-plan et un processus en avant-plan.

> **Remarque :** Vous pouvez ici utiliser la primitive C `void *signal(int sig, void (*traitement)(int));` dont le rôle est d'affecter un traitement à un signal. Son utilisation doit d'ailleurs être limitée aux traitements `SIG_DFL` et `SIG_IGN` (voir `man signal` sous Linux). Il est par contre préférable d'utiliser `sigaction` dans tous les autres cas. Si les actions liées à tous les signaux utilisés sont les mêmes à l'exception du traitement, pour simplifier l'affectation du traitement aux signaux, il est possible de définir une fonction `int set_signal(int sig, void (*traitement)(int))` qui utilise la primitive `sigaction`.

Etape 11.3 (Masquer les signaux `SIGINT` et `SIGTSTP`) La dernière solution possible consiste à masquer les signaux. La primitive `sigprocmask` réalise cette opération :

```
int sigprocmask(int how, const sigset_t *set, sigset_t *old_set);
```

La primitive `sigprocmask` permet de consulter et d'affecter le masque des signaux du processus appelant, qui indique l'ensemble des signaux que le processus peut recevoir et ceux qui n'auront pas d'impact. `how` peut prendre une des trois valeurs suivantes :

- `SIG_BLOCK` : `set` est ajouté au masque courant ;
- `SIG_UNBLOCK` : `set` est retiré du masque courant ;
- `SIG_SETMASK` : `set` remplace le masque courant.

`old_set` permet de récupérer le masque précédent mais peut valoir `NULL`.

Si `set` vaut `NULL`, `how` n'est pas significatif, le masque courant reste inchangé : `sigprocmask (SIG_BLOCK, NULL, &masq_courant)` permet de récupérer le masque courant dans l'ensemble `masq_courant`.

Pendant un masquage, seule une occurrence de chaque signal masqué est mémorisée ; si le signal est démasqué, cette occurrence est délivrée

Pour rappel, l'ensemble des signaux à masquer est défini par le type abstrait de données `sigset_t`. Ce type dispose des fonctions :

- `int sigemptyset(sigset_t *set)` : `set` $\leftarrow \emptyset$
- `int sigfillset(sigset_t *set)` : `set` $\leftarrow \{1, ..., \texttt{NSIG}\}$
- `int sigaddset(sigset_t *set, int signal)` : `set` $\leftarrow$ `set` $\cup \{$ `signal` $\}$
- `int sigdelset(sigset_t *set, int signal)` : `set` $\leftarrow$ `set` $\setminus \{$ `signal` $\}$
- `int sigismember(sigset_t *set, int signal)` : retourne 1 si `signal` $\in$ `set`, 0 sinon.

Utilisez `sigprocmask` pour masquer les signaux `SIGINT` et `SIGTSTP`. Lorsque le masque est vide, le processus peut recevoir tous les signaux. Testez une nouvelle fois la frappe de `ctrl-C` puis de `ctrl-Z`, avec un processus en arrière plan et un processus en avant plan.

Etape 12 (Détacher les processus fils en arrière plan) À ce stade, le processus `minishell` n'est plus sensible à la frappe sur les touches `ctrl-C` et `ctrl-Z` mais tous les processus reçoivent les signaux `SIGINT` et `SIGTSTP`. Or, lors de la frappe, seul le processus en avant plan doit recevoir le signal. En réalité, les signaux sont transmis à tous les processus du même groupe que le processus `minishell`. La solution que nous proposons est de mettre les processus en arrière plan dans un autre groupe de processus. Nous utiliserons ici la primitive `int setpgrp()` qui associe une nouveau groupe au processus appelant. Testez une dernière fois, la frappe de `ctrl-C` puis de `ctrl-Z`, avec un processus en arrière plan et un processus en avant plan.