TP 6
Signaux

EXERCICE 1 ▶ Un premier gestionnaire de signal

Créez le fichier `tstsig.cpp`.
La commande `g++ tstsig.cpp -o tstsig` permettra de le compiler; la commande `./tstsig` de l'exécuter.

1) Le processus principal (c'est-à-dire la fonction d'entête `int main(void)`) doit exécuter la boucle infinie suivante :

```
while(true) {
    cout << "mise en sommeil" << endl << flush;
    sleep(60);
}
```

À l'exécution, votre programme ne doit *a priori* jamais se terminer, et ainsi rester bloqué dans la boucle infine. Tout ce qu'il va faire, c'est afficher « `mise en sommeil` » une première fois, puis toutes les 60 secondes.

2) Dans la question suivante, nous allons mettre en place un gestionnaire pour le signal `SIGUSR1`. Pour lancer ce signal à l'exécutable `tstsig`, il suffit d'appeler la commande shell suivante :

```
kill -USR1 $(pidof tstsig)
```

Essayez de faire cette opération une première fois, même si pour le moment ce signal n'est pas traité par l'exécutable.

3) Vous allez modifier votre programme, pour installer dans le procesus principal, un gestionnaire pour le signal `SIGUSR1` :

(a) vous devez utiliser la fonction système `sigaction()` pour mettre en place le gestionnaire pour le signal `SIGUSR1`,

(b) ce gestionnaire de signal doit permettre l'appel de la procédure `handler` à l'émission du signal `SIGUSR1`,

(c) cette procédure `handler`, gestionnaire de `SIGUSR1`, doit afficher simplement « `signal SIGUSR1 reçu` »,

(d) et il vous est demandé de ne pas utiliser d'appel à `exit()`,

(e) enfin, notons que pour faciliter la programmation de la question suivante, vous utiliserez des variables globales qui seront utilisées dans la fonction `main()` et la procédure gestionnaire de signal `handler`.

4) Modifier la procédure `handler` pour faire en sorte qu'au bout de 5 réceptions du signal `SIGUSR1`, le processus se termine en affichant « `fin du programme` ».

EXERCICE 2 ▶ Synchronisation entre deux processus

Lorsqu'un processus reçoit le signal `SIGSTOP`, il s'endort (son exécution est stoppée). Pour le réveiller il suffit de lui envoyer `SIGCONT`. On se propose d'utiliser ces deux signaux pour synchroniser l'exécution d'un processus fils sur celle de son père.

Vous devez écrire un programme `ping-pong.cpp`, dans lequel le processus principal crée un processus fils. Après s'être endormi 1 seconde, le père envoie le signal `SIGCONT` à son fils, puis affiche un message, et recommence ainsi 10 fois. Le fils lui s'endort en s'envoyant à lui-même le signal `SIGSTOP`, se réveille quand il reçoit le signal `SIGCONT`, affiche un message, et recommence ainsi 10 fois. De façon plus détaillée,

- le père doit effectuer 10 fois les actions suivantes :
  - il s'endort 1 seconde avec `sleep(1)`,
  - il envoie le signal `SIGCONT` à son fils avec la primitive `kill()`,
  - il affiche « `père : ping` »
- le fils exécute 10 fois les actions suivantes :
  - il se met en état de sommeil en s'envoyant le signal `SIGSTOP` (il ne reprendra son exécution qu'après avoir reçu `SIGCONT`),
  - il affiche « `fils : pong` »
- le père attend la terminaison de son fils avant de se terminer lui-même.

À l'exécution, vous devez donc voir s'afficher sur la sortie standard les deux lignes suivantes affichées en tout 10 fois :

```
père : ping
fils : pong
```