TP 6

Signaux

Durée prévue : 1 séance (1h30)

Avant-propos On fournit sur la page web du cours une archive contenant les fichiers nécessaires à ce TP, ainsi qu'un Makefile. Ne vous préoccupez pas des « warnings » avant d'avoir vous-même modifié le code !

EXERCICE 1 ▶ Signal périodique

Vous devez compléter le programme `signal_temps.cpp` (dans l'archive du TP). Dans ce programme, le processus principal entre dans une boucle infinie dans laquelle :

— il se met en attente de la réception d'un signal quelconque avec `pause()`,

— lorsqu'il reçoit le signal `SIGALRM`, il exécute un gestionnaire de signal qui affiche un caractère,

— il reprend son exécution après l'appel à `pause()`,

— il programme une nouvelle alarme (envoi du signal `SIGALARM`) après une seconde.

La réception du signal `SIGALRM` devra déclencher l'appel de la fonction `top()`. Vous utiliserez les primitives POSIX suivantes :

— `unsigned int alarm(unsigned int s)` qui permet de demander au système d'envoyer au processus appelant un signal `SIGALRM` au bout de `s` secondes.

— L'appel système `pause()`, qui met le processus en attente passive de la réception d'un signal (lire le manuel pour les détails). Cet appel système est obsolète, on devrait utiliser `sigsuspend()`, mais tant pis !

— La structure de donnée `struct sigaction` et son appel système associé `sigaction()`.

1) Vous avez deux choses à faire dans le `main()` :

— Au niveau du `TODO1`, créer un gestionnaire qui lie la réception de `SIGALRM` signal au lancement de la fonction `top()`, qui vous est fournie.

— Reprogrammer l'envoi du signal `SIGALRM` dans une seconde, au niveau du `TODO2` dans le code.

2) Compilez votre programme, puis testez : logiquement, si vous avez suivi scrupuleusement les consignes jusqu'ici, il ne se passe rien lors du lancement du programme. Pourquoi ? Comment faire depuis la ligne de commande pour que le programme « démarre » ?

3) Modifiez votre programme, pour qu'il démarre tout seul, puis testez à nouveau.

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
```