TP 3 (1.5h) : Communication entre processus par signaux

1. alarm

La primitive **alarm**(**nb_sec**) programme un « réveil » qui enverra un signal SIGALRM au processus appelant dans **nb_sec** secondes. Tout appel de **alarm**() annule et remplace l'éventuelle alarme précédente. L'appel **alarm**(0) annule toute alarme précédente et ne programme pas de nouvelle alarme.

Écrivez un programme qui incrémente en boucle un compteur entier et affiche la valeur courante du compteur toutes les 2 secondes. Il faudra pour cela traiter le signal SIGALRM (cf. fonction **signal**).

2. Détecter la fin d'un processus fils sans l'attendre

Ecrire un programme qui lance un processus fils puis vaque à ses occupations à l'infini (par exemple, il affiche en boucle un compteur, entre deux **sleep**). Le processus fils se met en sommeil pendant un certain temps (20 sec) puis se termine avec un code quelconque.

Nous souhaitons que le père puisse faire son travail en parallèle du fils, mais qu'il affiche toutefois un message lorsque le fils se termine. Il ne peut donc pas utiliser la fonction **wait** car cela le bloquerait.

a) Traiter le signal SIGCHLD pour afficher le message quand le fils se termine

b) Dans le message, inclure le code de retour du fils (récupéré avec **wait** et **WEXITSTATUS**)

c) Que se passe-t-il si le fils est terminé par un signal (par exemple SIGTERM) ? Comparez ce comportement avec le cas où le fils est seulement suspendu (par SIGSTOP depuis un autre terminal) *avant sa fin.* Pourquoi dans ce cas le processus parent semble-t-il suspendu également ?

d) Remplacer l'appel à **wait** par :

```
waitpid(-1, &status, WNOHANG);
```

Cela marche-t-il mieux en cas de SIGSTOP sur le fils ? Expliquer pourquoi.