

Linux 2 : Appels Systèmes - BINV2181 (tp05- signaux)

5.1. Appel système : sigprocmask & alarm

Le programme suivant va simuler un *heartbeat*¹.

- a) Le processus père crée un fils. Ils restent tous deux dans une boucle d'attente active. Le fils indique régulièrement à son père qu'il est toujours vivant. Pour ce faire, il envoie des signaux SIGUSR1 toutes les 3 secondes à son père. Cet envoi régulier de heartbeats est programmé grâce à une alarme.

Si le père ne reçoit pas de heartbeat (signal SIGUSR1) de son fils pendant 8 secondes, il affiche le message d'erreur « mon fils est down » et se termine avec l'exit code 5.

Au contraire, lorsque le père reçoit un heartbeat, il affiche le message « mon fils est toujours en vie » et réinitialise son minuteur pour une nouvelle période de 8 secondes.

Voici le squelette du programme :

```
pid_t childID = fork();

// PERE
if (childID > 0) {

    /* lancement d'un compte à rebours de 8 secondes */
    //TODO

    while (true) {
        sleep(10);
    }
}
// FILS
else {

    /* lancement d'un compte à rebours de 3 secondes */
    //TODO

    while (true) {
        sleep(10);
    }
}
```

Pour tester ce programme, simulez la mort du fils en lui envoyant un signal SIGKILL en ligne de commande.

¹ Il s'agit d'un protocole de communication dans lequel un ou plusieurs services d'une infrastructure informatique à haute disponibilité envoient périodiquement des messages « Oui, je suis toujours en vie » à leurs pairs. Ces derniers prennent des mesures s'ils ne reçoivent pas de message « Oui, je suis en vie » avant un délai déterminé, indiquant que le service est *down* (panne matérielle, logicielle ou autre).

- b) Dans une variante de ce programme, la mort du fils sera provoquée par Ctrl-C (signal d'interruption clavier SIGINT). Rappelons que Ctrl-C provoque l'émission d'un SIGINT au processus qui a la main sur le terminal mais aussi à tous ses processus fils (*foreground process group*).

Lorsque Ctrl-C est entré au clavier, le processus fils doit mourir mais pas son père !

Constatant après 8 secondes la mort probable de son fils, le père affichera le message d'erreur avant de lui-même se terminer.

5.2. Appels système : sigprocmask, sigsuspend & alarm

Ecrivez un programme « pingpong.c » où deux processus se renvoient mutuellement un signal SIGUSR1.

Le processus père effectue une boucle infinie où il envoie SIGUSR1 à son fils puis se met en attente du même signal. Le processus fils attend la réception du signal SIGUSR1 envoyé par son père et, à la réception de celui-ci, renvoie un signal SIGUSR1 à son père puis retourne en attente du signal.

Faites-en sorte que le processus père se termine après 5 secondes. Avant de se terminer, il affiche le nombre de signaux SIGUSR1 reçus et envoyés, puis envoie un signal SIGUSR2 à son fils pour que celui-ci se termine à son tour en affichant le nombre de signaux SIGUSR1 reçus et envoyés.

Voici un exemple d'exécution :

```
anthony@LAPTOP-GROTO:LAS_05_solutions$ ./pingpong
Durée de l'exécution limitée à 5 secondes
56777/56777 signaux envoyes/recus par pere
56778/56777 signaux envoyes/recus par fils
```

Pour résoudre cet exercice, utilisez les appels système `sigprocmask`, `sigsuspend` et `alarm`.

[Pour réviser] 5.3. Appel système : sigprocmask & sigsuspend

4		R			
3					
2					
1					
0					
Y/X	0	1	2	3	4

Tableau 1 : Une grille de 5 X 5 cellules

Imaginons un robot se trouvant dans une grille de 5 X 5 cellules (cf Tableau 1). Chaque cellule possède une coordonnée X et une coordonnée Y. Dans le Tableau 1, le robot ('R') se trouve dans la cellule X = 1 et Y = 4. Initialement, le robot se trouve en X = 0 et Y = 0.

Le robot est télécommandé par une manette qui possède 4 boutons :

1. Le bouton « U » (up) qui permet au robot d'aller vers le nord, qui incrémente Y.
2. Le bouton « D » (down) qui permet au robot d'aller vers le sud, qui décrémente Y.
3. Le bouton « L » (left) qui permet au robot d'aller vers l'ouest, qui décrémente X.
4. Le bouton « R » (right) qui permet au robot d'aller vers l'est, qui incrémente X.

Notez que le monde du robot est circulaire :

1. Si le robot est en « Y = 4 » et qu'on appuie sur le bouton « U », le robot ira en « Y = 0 ».
2. Si le robot est en « Y = 0 » et qu'on appuie sur le bouton « D », le robot ira en « Y = 4 ».
3. Si le robot est en « X = 0 » et qu'on appuie sur le bouton « L », le robot ira en « X = 4 ».
4. Si le robot est en « X = 4 » et qu'on appuie sur le bouton « R », le robot ira en « X = 0 ».

Dans la cadre de cet énoncé, le robot ne peut se déplacer qu'au nord et à l'est.

Ecrivez un programme « **robot.c** » qui crée un fils : le père représentera la manette et le fils le robot.

1. Le père lit des lignes au clavier. Les lignes simulent le fait d'appuyer sur un bouton. Elles peuvent prendre les trois formes suivantes :
 - a. « U\n » : dans ce cas, le père envoie un signal « SIGUSR1 » à son fils
 - b. « R\n » : dans ce cas, le père envoie un signal « SIGUSR2 » à son fils
 - c. « Q\n » : dans ce cas, le père envoie un signal « SIGKILL » à son fils
2. Le fils affiche la position initiale du robot et la grille correspondante. De plus, il gère le traitement des signaux « SIGUSR1, SIGUSR2 et SIGKILL » et affiche, à chaque déplacement du robot, la position de celui-ci et la grille. Il gère les signaux de la manière suivante :
 - a. Lorsqu'il reçoit un signal « SIGUSR1 », le robot se déplace vers le nord.
 - b. Lorsqu'il reçoit un signal « SIGUSR2 », le robot se déplace vers l'est.
 - c. Lorsqu'il reçoit un signal « SIGKILL », le fils est tué.

Consignes :

- La fonction d'affichage de la grille `printBoard` vous est fournie dans le fichier source **robot.c**.
- Les signaux doivent être bloqués (`sigprocmask`) avant la création du processus fils.
- Le fils doit armer ses gestionnaires (`sigaction`) avant de se mettre en attente de certains signaux (`sigsuspend`).
- Le traitement de(s) handler(s) de signaux doit être réduit au strict minimum.

Voici une exécution possible :

```
anthony@LAPTOP-GROTO:LAS_05_solutions$ ./robot
```

```
Robot commands:  
'U' to go up  
'R' to go right  
'Q' to quit
```

```
==> X = 0, Y = 0
```

```
      0  1  2  3  4  
-----  
4 |           | 4  
3 |           | 3  
2 |           | 2  
1 |           | 1  
0 |  R         | 0  
-----  
      0  1  2  3  4
```

```
R
```

```
==> X = 1, Y = 0
```

```
      0  1  2  3  4  
-----  
4 |           | 4  
3 |           | 3  
2 |           | 2  
1 |           | 1  
0 |   R        | 0  
-----  
      0  1  2  3  4
```

```
U
```

```
==> X = 1, Y = 1
```

```
      0  1  2  3  4  
-----  
4 |           | 4  
3 |           | 3  
2 |           | 2  
1 |   R        | 1  
0 |           | 0  
-----  
      0  1  2  3  4
```

```
Q
```

```
anthony@LAPTOP-GROTO:LAS_05_solutions$
```