Programmation Unix
TP
Les signaux

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affichés avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

NB : Les parties dont le texte est rédigé en gris sont facultatives

Question 1 Emisssion d'un signal `kill ()`

La primitive `int kill(pid_t pid, int sig)` permet l'envoi d'un signal à un processus ou à un groupe de processus. La valeur du paramètre `pid` définit l'ensemble des processus destinataires du signal `sig`.

a) Ecrire un programme qui envoie différents signaux au(x) processus selon les valeurs possibles des deux paramètres `pid` et `sig`.

b) Que devient le processus destinataire ? Expliquer !

Une valeur nulle (0) du paramètre `sig` correspond au test d'existence du processus mentionné.

c) Vérifier l'affirmation précédente.

d) Dans le cas précédent, y a-t-il émission de signal ? Que devient le processus destinataire ?

La fonction `int raise(int sig)` fait partie de l'interface standard du langage C et permet l'envoi du signal `sig` au processus courant.

e) Ecrire un programme qui s'envoie plusieurs signaux différents.

f) Décrire le résultat de la précédente exécution ? Expliquer !

Question 2 Attente d'un signal `pause ()`

La primitive `int pause(void)` permet de se mettre en attente de l'arrivée de signaux.

a) Ecrire un programme dont le processus correspondant se met en attente de l'arrivée de signaux. Emettre à ce processus les signaux `SIGUSR1` et `SIGSTOP`.

b) Que devient le processus après la délivrance du signal `SIGUSR1` ?

c) Que devient le processus après la délivrance du signal `SIGSTOP` ? Que se passe-t-il s'il reçoit à la suite le message `SIGCONT` ?

Question 3 Le gestionnaire de signal ANSI C `signal ()`

La primitive `void (*signal(int signo, void (*func)(int)))(int);` permet d'installer un nouveau gestionnaire `func` pour le signal `signo`. Le gestionnaire peut être soit une fonction spécifique de l'utilisateur, soit l'une des constantes `SIG_IGN` ou `SIG_DFL`.

a) Ecrire un programme qui ignore la réception des signaux `USR1` et `USR2`.

b) Vérifier que le processus ignore bien les signaux utilisateur.

c) Quel aurait été le résultat si les signaux n'étaient pas ignorés ?

d) Réécrire le programme précédent de façon à installer un gestionnaire spécifique pour les deux signaux `USR1` et `USR2`. Sur réception de chacun de ces deux signaux, le gestionnaire doit afficher la nature (`USR1` ou `USR2`) du signal reçu ainsi que le nombre total d'occurrences reçues de ce même signal.

e) Ecrire un programme qui envoie un grand nombre de signaux au programme précédent. Quel résultat obtenez-vous lorsque vous émettez plusieurs occurrences des signaux précédents ? Expliquer !

Question 4 Le gestionnaire de signal POSIX `sigaction ()`

La primitive
```
int sigaction(int signo, const struct sigaction *act,
                                    struct sigaction *oldact);
```
permet de modifier l'action effectuée par un processus lors de la réception d'un signal spécifique

Question d'examen (de a à b)

a) Ecrire un programme **`ring.c`** qui se lance de la façon suivante : **`./ring`** ***`n m`***. L'exécution de ce programme créera *n* processus de *P1* à *Pn* (*n*>1) dans cet ordre. Chaque processus *Pi* envoie le signal SIGUSR1 au processus *P(i-1)* puis attend à son tour l'arrivé de SIGUSR1. Chaque Pi répète cette séquence *m* (*m*>1) fois puis se termine.

- Le précèdent de *P1* est *Pn*.
- *P1* est le résultat de **`./ring`** ***`n m`***.

b) Est-il possible de faire circuler SIGUSR1 en sens inverse, c.à-d. de Pi vers P(i+1). Justifiez votre réponse.

c) Modifiez le programme écrit en a) afin que chaque processus affiche :

1. Le masque courant au tout début de son exécution
2. Le masque des signaux pendants au tout début de son exécution
3. Le masque courant juste avant l'envoi de SIGUSR1
4. Le masque des signaux pendants et le masque courant lorsqu'il attend SIGUSR1
5. Le masque courant lorsqu'il a reçu SIGUSR1

d) Si vous avez écrit le programme en a) avec `pause`, écrivez le avec `sigsuspend`. Si vous l'avez écrit avec `sigsuspend`, écrivez le avec `pause`:

Question 5 La primitive : `alarm()`

L'appel à la primitive `unsigned int alarm(unsigned int sec) ;` correspond à une requête au système d'envoyer au processus appelant le signal `SIGALARM` dans `sec` secondes.

a) Ecrire un programme qui lit cinq données depuis son entrée standard et qu'il sauvegarde dans les cinq entrées d'un vecteur. Pour chaque donnée, il doit faire une requête à l'utilisateur en affichant un message lui demandant de l'introduire. Si la donnée n'est pas introduite par l'utilisateur au bout d'un temps d'attente (2 secondes, par exemple) le programme réitère sa requête jusqu'à ce qu'elle le soit. A chaque nouvelle requête, pour une même donnée, le temps d'attente est augmenté d'une seconde. A la fin du programme, pour chaque donnée, afficher le nombre de tentatives de requêtes qu'il a fallu ainsi que le temps écoulé avant son introduction.

b) Définissez un point de reprise (`sigsetjmp`) juste avant la requête demandant à l'utilisateur d'introduire une donnée. Reprenez l'exécution du programme à ce point de reprise (`siglongjmp`) chaque fois que l'utilisateur n'a pas introduit la donnée au bout du temps d'attente