ESIEE AMIENS	TRAVAUX PRATIQUES N°1 Matière : Programmation multitâches Remis par : S.Pomportes / Y. Guerroudj J. Olive / C. Sarroche	Année : I4 GRIT/RIOC Temps : 4h

OBJECTIFS

Comprendre le fonctionnement des signaux ainsi que leur utilisation.

REMARQUES

Vous placerez l'ensemble de vos fichiers sources dans le répertoire PMTP2 de votre répertoire personnel (commande mkdir $HOME/PMTP2).
Vous rédigerez un compte rendu au format openoffice ou txt.
Chaque fichier (compte rendu ou code) doit comporter vos noms au début.

Les signaux

1. Présentation des signaux

Les signaux ont des origines diverses, ils peuvent être :

1. retransmis par le noyau : division par zéro, overflow, instruction interdite.
2. envoyés depuis le clavier par l'utilisateur (touches : *<CTRL>Z*, *<CTRL>C*, *<CTRL>*, ...).
3. émis par la commande *kill* depuis le shell ou depuis le C par l'appel à la primitive *kill*

Emission :
kill (num_du_processus, num_du_signal) en C.
kill -num_du_signal num_du_processus, en Shell.

Remarque : l'émetteur ne sait pas si le destinataire a reçu ou non le signal.

Réception :
Les comportements possibles du destinataire du signal sont décrits dans le tableau ci-dessous.

Ignorer le signal	`signal(num_du_signal,SIG_IGN)`
Repositionner le traitement par défaut	`signal(num_du_signal,SIG_DFL)`
Définir un traitement spécifique	`signal(num_du_signal,fonction)`

2. Ignorer les signaux

Ecrivez un programme qui ignore tous les signaux ignorables et qui attend.

1. A l'aide de la fonction *strsignal* (utilisez le *man* pour avoir de plus amples informations sur cette fonction), affichez les noms des NSIG premiers signaux. Combien de signaux utiles y a-t-il (on ne compte pas les signaux temps réels) ? On note N ce nombre.
2. Ignorez ces N signaux à l'aide de la fonction *signal*. Parmi ces signaux, lesquels ne sont pas ignorables ? (pensez à utiliser la valeur de retour de la fonction). N'hésitez pas à utiliser les commandes "man 2 signal" pour en savoir plus sur la fonction *signal* et "man 7 signal" pour en savoir plus sur les signaux. D'après vous, pourquoi certains signaux ne sont pas ignorables?
3. A la fin du programme, effectuez une boucle infinie sur un sleep(1) de manière à pouvoir tester les envois de signaux vers le programme.
4. Faire *<CTRL>C* dans la fenetre où tourne le programme. Que se passe-t-il ? Pourquoi ? envoyez également des signaux vers ce programme en utilisant *kill* depuis une autre fenetre. Quels signaux ont un effet visible sur le programme ? Expliquez ces effets. Une fois le programmé *stoppé*, pouvez-vous le réveiller ?

3. Traitement spécifique des signaux

Modifiez le programme précédent : les signaux ne seront plus ignorés, mais traités par une fonction spéciale, que l'on appellera `Traite_Sig`. Cette fonction affichera *"`Fonction Traite_Sig : j'ai reçu le signal xx`".*

1. Testez la valeur de retour de la fonction signal pour relever les (rares) signaux pour lesquels on ne peut définir un traitement spécifique.
2. Dans une autre fenetre, envoyez des signaux à l'aide de la commande *kill* pour vérifier le fonctionnement de votre programme.

4. Utilisation de SIGUSR1 et SIGUSR2

Ecrivez un programme qui :

- Affiche son numéro (pid) via l'appel à `getpid()`.
- Traite tous les signaux traitables (sauf les signaux utilisateurs) par une fonction `fonc` qui se contente d'afficher le numéro du signal reçu.
- Traite le signal *SIGUSR1* par une fonction `fonc1` et le signal *SIGUSR2* par `fonc2`.
    1. `fonc1` affiche le numéro du signal reçu et la liste des utilisateurs de la machine (appel à `who` par `system("who")`).
    2. `fonc2` affiche le numéro du signal reçu et l'espace disque utilisé sur la machine (appel à `df .` par `system("df .")`).

Lancez votre programme et envoyez-lui des signaux (dont *SIGUSR1* et *SIGUSR2*) depuis une autre fenêtre, à l'aide de la commande *kill*. Lors de l'envoi des signaux *SIGUSR* 1 et 2, que se passe-t-il ? Pourquoi ? N'hésitez pas à lire le *man* de la commande *system*.

5. Signaux et sleep

1. Reprenez le programme précédent en ignorant le signal *SIGUSR1* et en traitant le signal *SIGUSR2* par une fonction affichant « J'ai reçu le signal ».
2. Dans la bouche infinie à la fin du programme, remplacez *sleep(1)* par *sleep(10)*;*printf("je dors\n");* de manière à ce que le programme affiche "je dors" toutes les 10 secondes.
3. Envoyez des signaux *SIGUSR1* et *SIGUSR2* au programme. Que pouvez-vous remarquer ? pourquoi ?

6. Traitement de SIGFPE

Ecrire un programme ayant une fonction `Traite_FPE` permettant de traiter *SIGFPE* (par exemple obtenu lors d'une division par zéro).

Votre programme doit attendre une seconde (à l'aide d'un sleep) , afficher : "Attention à la division par zéro ! " puis effectuer cette division.

1. Vérifier qu'on passe bien par `Traite_FPE`. `Traite_FPE` doit afficher « Détection d'une erreur ». Expliquez le comportement du programme.
2. Ensuite, en utilisant *sigsetjmp* et *siglongjmp*, reprendre l'exécution juste AVANT le sleep. Utiliser le `man` pour voir comment utiliser ces fonctions. Avant le sleep, afficher « Reprise sur errreur » si tel est le cas.

Nb : Les fonctions *sigsetjmp* et *siglongjmp* sont utilisées intensivement par les noyaux des systèmes multitâches afin d'effectuer les changements de contexte. Ces fonctions différent de leurs homologues *setjmp* et *longjmp* par la façon dont elles gèrent les signaux.

Pour se détendre pendant la pause (uniquement la pause !) et se préparer a affronté l'ultime exercice de ce TP :

telnet towel.blinkenlights.nl

7. Le combat ultime : SkyWalker contre DarkVador

Il s'agit d'une simulation d'un combat au sabre entre SkyWalker (le bon) et DarkVador (le méchant), héros de la célèbre saga...

Le bon

Le méchant

L'envoi d'un coup de sabre est simulé par l'émission d'un signal vers l'adversaire (SIGUSR1). Chacun des combattants se protège (comportement par défaut) en ignorant le signal qui lui est destiné. Dès qu'ils attaquent leur adversaire, les protagonistes baissent leur garde (deviennent sensibles aux signaux, donc peuvent recevoir des coups). Vous devez matérialiser le temps pendant lequel le coup est porté. La fin de l'attaque correspond bien évidemment à l'émission d'un signal, sauf si un coup a été reçu entre temps. Dans ce cas l'attaque est annulée. Entre deux attaques, chaque opposant attend en jaugeant son adversaire entre 1 et 5 secondes.

SkyWalker met 2 secondes pour attaquer, alors que DarkVador met 3 secondes pour attaquer. D'un autre coté, les coups portés par SkyWalker provoquent 1 point de dégats alors que ceux portés par DarkVador provoquent 3 points de dégat. Les coups sont portés à des moments aléatoires (fonction *rand*). Chacun des adversaires a le même capital en points de vie au début du combat (six étant valable pour un combat durant quelques dizaines de secondes). A chaque coup porté, le capital diminue du nombre de points de dégats reçus. Lorsque le capital d'un protagoniste est nul ou négatif, ce protagoniste meurt et son opposant est déclaré vainqueur. Le programme s'achève alors.

Qui va donc gagner ce combat de titans ? La galaxie sera-t-elle sauvé ? C'est ce que vous saurez en réalisant un merveilleux programme.

Vous rendrez un source, dans lequel vous tiendrez compte du fait que DarkVador est le père de SkyWalker. Ne vous trompez pas de saga, et évitez les zombies. Votre source sera commenté ; à la fin vous expliquerez ce que vous avez effectivement réalisé.

Note : La main ne doit être rendue au *bash* que lorsque le combat est terminé ! (Aucun printf ne doit apparaître une fois le prompt réapparu).

Facultatif : A chaque instant, Dieu (vous !) peut interrompre le combat.

Ps : May the force be with you