Programmation Système Travaux Pratiques (4), Licence 2 Informatique Gestion de processus (2/2)

Les exercices suivants ont pour but de vous familiariser avec les appels système de base relatifs à la gestion des processus.

Il vous est recommandé de consulter les pages man de ces primitives pour de plus amples informations sur leur syntaxe, leur sémantique et les éventuelles options qu'elles offrent.

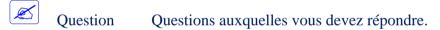
Les instructions des exercices se repèrent par des icônes, qui sont les suivantes :

i	Information	Information concernant l'usage ou le rôle d'une commande, par exemple. Dans certains cas, il s'agit d'une information sur ce que vous êtes en train de faire ou sur ce qui se passe.

	Exemple	Exemple d'utilisation.
--	---------	------------------------

	Contrôle	Vérifier le résultat d'une (ou plusieurs) action(s).
--	----------	------------------------------------------------------





De plus, un texte en police courier correspond soit à une sortie écran soit à des noms spécifiques (menus, fenêtre, icône, processus, commandes...).

Un **texte en police times gras** correspond à ce que l'utilisateur doit introduire comme valeur de paramètre, ou encore, est utilisé pour attirer l'attention de l'utilisateur.

Le recouvrement

Les primitives exec()

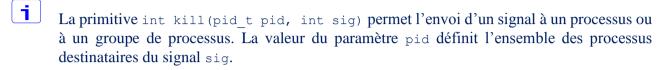
Les primitives de la famille exec() permettent de charger en mémoire, en vue de leur exécution, de nouveaux programmes binaires.

Les primitives de la famille <code>exec()</code> se différencient par la manière dont les arguments sont transmis. Ces arguments sont transmis soit sous forme d'un tableau (famille <code>execv()</code>), soit sous forme de liste (famille <code>execl()</code>) selon que la primitive utilisée a un nombre fixe ou variable de paramètres.

- Écrire un programme qui charge un nouveau programme binaire et dont les arguments sont transmis selon les deux modes précédents.
- Les primitives de la famille exec() se différencient également par la manière dont le programme à charger est recherché dans le système de fichiers. Soit la recherche est relative au répertoire courant, soit elle l'est par rapport aux répertoires spécifiés via la variable PATH.
- Écrire un programme qui charge un nouveau programme binaire qui est recherché selon les deux modes précédents.
- Les primitives de la famille exec() se différencient enfin par l'environnement conservé par le processus après recouvrement. Soit l'environnement reste inchangé, soit un nouvel environnement est transmis en paramètre.
- Écrire un programme qui charge un nouveau programme binaire et qui dans un premier cas conserve le même environnement et dans un second cas acquière un nouvel environnement.
- Au cours d'un recouvrement, les caractéristiques suivantes ne sont pas conservées selon les conditions spécifiées :
 - **propriétaire effectif** : si le set-uid bit est positionné sur le fichier exécutable chargé, le propriétaire de ce fichier devient propriétaire effectif du processus,
 - **groupe effectif** : si le set-gid bit est positionné sur le fichier exécutable chargé, le groupe propriétaire de ce fichier devient groupe propriétaire effectif du processus,
 - **descripteur de fichier ouvert** : si le bit FD_CLOEXEC d'un descripteur a été positionné à l'aide de la primitive fcntl(), ce descripteur est fermé après un recouvrement.
- Écrire des programmes qui vérifient les affirmations précédentes.
- Au cours d'un recouvrement, les attributs suivants sont hérités par le nouveau programme :
 - identifiant de processus (ID) et identifiant du processus parent (PID),
 - identifiant utilisateur réel (UID) et identifiant de groupe réel (GID),
 - répertoire de travail courant,
 - masque de création de fichier,
 - verrous sur les fichiers.
 - masque des signaux (cf. partie suivante),
 - signaux pendants (cf. partie suivante).
- Écrire des programmes qui vérifient les affirmations précédentes.

Gestion des signaux

Émission d'un signal





- Que devient le processus destinataire ? Expliquer !
- Une valeur nulle (0) du paramètre sig correspond au test d'existence du processus mentionné.
- Vérifier l'affirmation précédente.
- 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.
- Écrire un programme qui s'envoie plusieurs signaux différents.
- Décrire le résultat de la précédente exécution ? Expliquer !

Attente d'un signal

- La primitive int pause (void) permet de se mettre en attente de l'arrivée de signaux.
- Écrire un programme dont le processus correspondant se met en attente de l'arrivée de signaux. Émettre à ce processus les signaux SIGUSR1 et SIGSTOP.
- Que devient le processus après la délivrance du signal SIGUSR1 ?

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

Installation d'un gestionnaire de signaux : le gestionnaire ANSI C de signaux

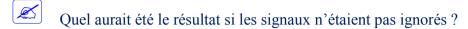
Le gestionnaire ANSI C de signaux

void (*signal(int signo, void (*func)(int)))(int);

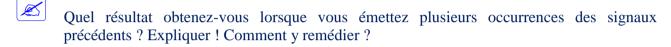
permet d'installer un nouveau gestionnaire func pour le signal signo. Le gestionnaire peut être soit une fonction spécifique de l'utilisateur, soit l'une des constantes <code>SIG_IGN</code> ou <code>SIG_DFL</code>.







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.



Fonctions POSIX de manipulation de signaux

Installation d'un gestionnaire de signaux



- Écrire un programme 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.
- Quelle est la différence par rapport à l'utilisation de la fonction signal()?
- Modifier le programme précédent de façon que la fonction signation () ait le comportement par défaut de la fonction signal ().
- Vérifier que le comportement de la fonction signation() est bien celui par défaut de la fonction signal().
- Écrire un programme de façon qu'un signal capté (gestionnaire spécifique associé) soit bloqué pendant l'exécution du gestionnaire.
- Réécrire le programme précédent de façon que le signal capté (gestionnaire spécifique associé) ne soit pas bloqué pendant l'exécution du gestionnaire.
- Vérifier l'exécution des deux programmes précédents en émettant (à nouveau) le signal capté pendant l'exécution du gestionnaire (utiliser la fonction sleep() dans le gestionnaire). Indiquer bien l'entrée et la sortie du gestionnaire ainsi que l'occurrence de l'appel au gestionnaire.



Écrire un programme comprenant deux fonctions gest 1 et gest 2 de façon que le gestionnaire d'un signal donné soit tour à tour (à chaque nouvelle réception de ce signal) gest 1 et gest 2.

Fonctions de manipulation d'ensembles de signaux



La norme POSIX a défini le type de données sigset t pour contenir un ensemble de signaux et les cinq fonctions suivantes pour manipuler les ensembles de signaux :

```
int sigemptyset(sigset t *set);
int sigfillset(sigset t *set);
int sigaddset(sigset t *set, int signo);
int sigdelset(sigset t *set, int signo);
                  Toutes retournent 0 si OK, -1 sinon
     sigismember(const sigset t *set, int signo);
```





Écrire un programme qui gère un ensemble de signaux SIG SET, initialement vide, et qui vous propose le menu interactif suivant :

- Ajout d'un signal donné à l'ensemble SIG SET,
- Suppression d'un signal donné de l'ensemble sig set,
- Test si un signal donné appartient à l'ensemble SIG SET,
- Affichage des signaux appartenant à l'ensemble SIG SET.

Blocage et déblocage de signaux



La fonction sigprocmask() permet de modifier la liste des signaux actuellement bloqués. La fonction sigpending () retourne l'ensemble des signaux bloqués pour la délivrance et qui sont pendants pour le processus appelant.



Écrire un programme qui :

- bloque le signal SIGQUIT,
- s'endort n secondes (le temps de lui émettre le signal SIGQUIT),
- teste si le signal SIGQUIT est pendant (et affiche le résultat),
- débloque le signal SIGQUIT,
- se met en attente de réception d'un quelconque signal.



Quel comportement obtenez-vous de l'exécution du programme ?

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.



Écrire 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 donné (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 requêtes qu'il a fallu ainsi que le temps écoulé avant son introduction.

Contrôle du point de reprise



Les fonctions sigsetjmp() et siglongjmp() permettent de sauvegarder et de restaurer un environnement du processus. L'environnement se caractérise par une position dans la pile d'exécution du processus et d'un masque des signaux bloqués.



Écrire un programme qui procède à une division par (rand_nb - nombre). Le nombre rand_nb est tiré au hasard par le programme et est compris entre 0 et 9. nombre vous est sollicité par le programme est appartient au même intervalle précédent. Le programme se termine lorsque vous aurez (pour une même valeur de rand_nb) introduit successivement 5 nombres de valeur différente à rand_nb. Sinon, le programme reprend (suite à la division par zéro) avec un nouveau tirage au hasard de la valeur de rand_nb.



random est la fonction permettant de générer au hasard un nombre entier.



Écrire un programme dans lequel une fonction est appelée récursivement un nombre de fois donné (n, par exemple). Au n-ième appel, le programme vous propose d'introduire la profondeur (dans les appels récursifs) à laquelle vous désirez retourner. De façon que l'exécution soit plus "parlante", chaque exécution de la fonction doit ajouter une information à la donnée qu'elle aura reçue en paramètre et qu'elle passera elle même (information + donnée) à l'appel suivant.

Communication par tubes ordinaires



Écrire un programme qui permet à un processus de communiquer des données à un processus fils via des tubes ordinaires.

Modifier le programme précédent de façon que les deux processus communiquent dans les deux sens. Utiliser deux tubes et chaque processus doit garder ouvert l'ensemble des descripteurs de tubes.

Quel comportement obtenez-vous de l'exécution du programme précédent si les deux processus commencent chacun par une lecture ? Quelles solutions à ce problème ?

Communication par tubes nommés

Les tubes nommés permettent la communication entre processus, même sans lien de parenté.

Écrire un programme qui permet à un processus de communiquer des données à un processus fils via des tubes nommés.