Université Paul Sabatier
31062 Toulouse Cedex 09

Master 1 Informatique - MCPR

Modèles et Concepts du Parallélisme et de la Répartition
Travaux pratiques n° 1

Rappels

Un processus peut **créer dynamiquement** un autre processus en appelant la primitive : pid_t **fork**(void).

La **valeur retournée** par le fork représente, dans le contexte du père, le **pid** du fils créé (ou -1 en cas d'erreur) et, dans le contexte du fils, vaut **0**.

Un processus peut choisir de **terminer son exécution** à n'importe quel moment en appelant la primitive :
void **exit**(int valCR).

Le paramètre est un compte-rendu d'exécution que le fils souhaite communiquer à son père. Par convention, il vaut 0 si le fils est satisfait de son exécution et est strictement positif si le fils veut signaler qu'il a eu un problème.

Un processus père peut **attendre la terminaison** de l'un de ses fils en appelant la primitive : int **wait**(int *valCRDecalee)

Elle retourne -1 en cas d'erreur ou si aucun fils n'est en cours d'exécution, le pid du fils qui s'est terminé sinon.

La fonction main peut être paramétrée : int **main** (int argc, char *argv[]]

où argc représente le **nombre d'arguments** se trouvant sur la ligne de commande, et argv est un tableau de chaînes de caractères contenant la **valeur** de ces différents arguments (argv[0] représente le nombre de la commande exécutée).

Exercice

On souhaite programmer une application dans laquelle deux processus Unix font des opérations en parallèle sur un compteur commun. L'un incrémente ce compteur de la valeur 1 un certain nombre NB de fois tandis que l'autre le décrémente de 1 le même nombre de fois.

Variante 1

Dans un premier temps, les processus sont parents : l'un est le père de l'autre. On écrira une fonction pour représenter le traitement de chacun de ces processus qu'on utilisera pour créer ces processus dans le programme principal. Ce programme sera **paramétré** par le nombre d'incrémentations/décrémentations à réaliser. Le compteur sera implanté grâce à une variable globale.

Que constatez-vous ?

Variables partagées et synchronisation sous Posix

IPC : segments de mémoire partagée et sémaphores Posix

Présentation des IPC via le support de TP associé.

Exercice

Variante 2

Modifier le code précédent pour utiliser maintenant un segment de **mémoire partagée** pour implanter ce compteur.

Variante 3

On désire maintenant que ces processus n'aient **aucun lien de parenté** et que leur exécution puisse être lancée dans des fenêtres différentes, voire qu'un processus soit exécuté par un utilisateur et l'autre par un autre utilisateur. Que faut-il modifier dans le code précédent ?

Faire exécuter ce programme en faisant **varier le nombre d'itérations** entre 4000 et 10000.

Le résultat de l'exécution est-il toujours cohérent ? Pourquoi ?

Quelle solution peut-on proposer ?

Assurer la cohérence de l'exécution

Implanter la solution proposée.

Important

Assurez-vous avant de quitter votre session que les IPC utilisés sont bien détruits !

Voir commandes Unix : ipcs et ipcrm