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

Écrire un programme dans lequel deux processus Unix **parents** (un processus et son fils) font des opérations en parallèle sur un compteur. L'un incrémente ce compteur de la valeur 1 un certain nombre de fois tandis que l'autre le décrémente de 1 le même nombre de fois. À la fin de l'exécution, chacun affiche la valeur du compteur.

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.

Variante 1

Chaque processus déclare le compteur **localement** dans la fonction représentant son traitement.

- Que constatez-vous ?

Variante 2

Le compteur est déclaré en tant que **variable globale**.

- Que constatez-vous ?
- Que concluez-vous ?

Partage d'informations entre processus – Variables partagées

IPC : segments de mémoire partagée

Présentation des segments de mémoire partagée via le support de TP associé.

Exercice (suite)

Variante 3

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

Variante 4

Modifier le code précédent pour que les processus **partagent** maintenant **deux informations** : le compteur entier (sur lequel les mêmes opérations sont faites) et une valeur réelle qui sera modifiée par un processus et affichée par l'autre (pour vérifier que le partage se fait bien).

Variante 5

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 (au moins).

- Le résultat de l'exécution est-il toujours cohérent ? Pourquoi ?
- En vous appuyant sur les notions étudiées en cours et TD, quelle solution (théorique pour l'instant) peut-on proposer pour obtenir une exécution toujours cohérente ?

Important

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

Voir commandes Unix : ipcs et ipcrm