UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence 3 Informatique
Info0601

TP n°4

Processus et tubes

Le but de ce TP est de manipuler les appels systèmes relatifs à la gestion de processus et aux tubes nommé et anonyme.

Pour l'ensemble des questions suivantes, vous devez utiliser les fonctions C qui ont été introduites dans les cours correspondants. À noter que la gestion d'erreurs doit être correctement mise en place pour tous les appels systèmes.

1 Création de processus

L'objectif de cet exercice est de manipuler les commandes `fork`, `wait` (ou `waitpid` seulement si nécessaire) et `execve`.

1. Écrivez un programme qui prend en argument un entier n et qui crée ensuite n processus fils. Chaque fils génère un nombre aléatoire. Il affiche ensuite son numéro de création (de 1 à n) et le nombre généré. Le père attend la fin de chaque fils : à chaque fois qu'un fils s'arrête, il affiche son PID.

Vous devez utiliser la procédure `srand` pour initialiser le générateur de nombres pseudo-aléatoires. Elle prend en paramètre une graine. Habituellement, on utilise le retour de la fonction `time`. La fonction `rand` permet ensuite de générer des entiers aléatoires dans l'intervalle $[0; RAND\_MAX]$.

2. Modifiez votre programme pour que chaque fils attende un temps aléatoire (vous pouvez utiliser `sleep`). À la fin de chaque fils, le père doit indiquer le numéro de création et le PID du fils qui s'est arrêté. Pour le numéro de création, vous utiliserez la valeur de retour envoyée par `exit`.
3. Comment faire pour éviter d'utiliser la valeur retournée par `exit` ?

Dans le père, la fonction `fork` retourne le PID du fils qui a été créé...

4. Écrivez un premier programme qui prend en argument un entier i. Il affiche alors "Bonjour ; je vais attendre X secondes" (où X est la valeur de i) puis se met en pause pendant i secondes. Écrivez ensuite un deuxième programme qui prend en argument le nom du premier programme, un entier et qui exécute le premier programme à l'aide de `execve` en lui transmettant l'entier spécifié. Le deuxième programme doit attendre la fin du premier programme puis afficher un message de fin.

2 Les tubes anonymes

Dans cet exercice, nous utiliserons les tubes anonymes (pour rappel, ils sont créés à l'aide de `pipe`).

Prenez un soin particulier à fermer systématiquement tous les descripteurs des tubes non utilisés, aussi bien dans les fils que dans le père.

Questions

1. Écrivez un programme qui prend en argument un entier n et qui crée ensuite n tubes et n processus fils. Une fois tous les fils créés, le père lit depuis le clavier un numéro (entre 1 et n) et une chaine de caractères. Il envoie ensuite la chaîne de caractères au fils spécifié qui l'affiche. Quand l'utilisateur saisit -1, le père envoie un code de fin aux fils qui s'arrêtent. Une fois tous les fils terminés, le père s'arrête.
2. Modifiez votre programme pour que chaque fils envoie la chaîne « X bien reçu » (où X est le numéro du processus) au père à chaque fois qu'il reçoit un message.

3 Les tubes nommés

Nous souhaitons reprendre l'exercice précédent mais en utilisant les tubes nommés. Nous avons besoin de deux programmes : le programme principal qui correspondant au père de l'exercice précédent et le programme client qui correspond aux fils.

Questions

1. Le programme principal prend en argument un entier n et crée ensuite n tubes nommés dont le nom de fichier est « totoX » où X est un numéro entre 1 et n. Une fois qu'un client pour chaque tube est exécuté, le programme principal lit depuis le clavier un numéro (entre 1 et n) et une chaine de caractères. Il envoie ensuite la chaîne de caractères au client associé qui l'affiche. Quand l'utilisateur saisit -1, le programme principal envoie un code de fin aux clients qui s'arrêtent.
2. Que se passe-t-il si deux clients sont exécutés sur le même tube ?
3. Modifiez vos programmes pour que chaque client envoie « X bien reçu » (où X est le numéro du client) au programme principal à chaque fois qu'il reçoit un message.