Systèmes - M1 - Janvier 2010

Exercice 1

Écrivez une fonction qui :
1) tout d'abord, crée *n* processus,
2) ensuite, attend la terminaison, des n processus créés, dans **l'ordre inverse** de leur création.

Exercice 2

Écrivez une commande qui prend en argument : une chaîne représentant un nombre *n* et une suite d'arguments *args* représentant l'invocation d'une sous-commande avec ses arguments.
La commande doit faire en sorte de lancer l'exécution d'une sous-commande dont l'exécutable est spécifié par le premier argument de la suite *args* et ses propres arguments par le reste de la suite *args*,

1) soit la sous-commande n'existe pas, auquel cas la commande doit afficher un message indiquant cette erreur,
2) soit la sous-commande termine avant *n* secondes, auquel cas la commande doit afficher un message indiquant que tout s'est correctement passé, le tout accompagné de la valeur de retour de la sous-commande,
3) soit la sous-commande ne termine pas dans les n secondes imparties et la commande doit alors forcer sa terminaison et afficher un message l'indiquant.

Exercice 3

Répondez avec concision aux questions suivantes :
1) expliquez quelle différence il y a entre un processus en avant-plan et un processus en arrière-plan,
2) pourquoi l'exécution de la commande *vi* (qui est un éditeur en mode texte fonctionnant sur un terminal ordinaire) en arrière-plan provoque t-elle la réception du message "job stopped by tty output" ?
3) qu'est-ce qu'un processus zombi ?
4) écrivez un programme permettant de faire apparaître un processus zombi pendant une durée suffisamment longue que l'on puisse l'observer par l'exécution de la commande *ps*
5) quelle différence faites-vous entre une page logique et une page physique (une frame) ?
6) à quoi sert une zone de swap ?
7) pourquoi lorsqu'un processus tente d'accéder à une adresse *a priori* interdite, le système réagit parfois en laissant faire et d'autres fois en délivrant le signal SIGSEGV ?
8) que signifie "temps partagé" lorsqu'on dit qu'un système est à temps partagé ?
9) quelle différence faites-vous entre un mutex (qui permet de réaliser une exclusion mutuelle) et un sémaphore ?

Problème

Fort de votre nouveau diplôme, vous voilà embauché par une nouvelle société PixArt qui produit des films en format numérique. Aujourd'hui son système de numérisation utilise un mécanisme simple mais trop lent permettant d'obtenir la version compressée d'un flux vidéo brut : la vidéo captée brut est enregistrée dans un fichier, puis lorsque la captation est terminée, un processus de compression lit les données dans la mémoire partagée, compresse les données lues dans sa propre mémoire et écrit le résultat dans un autre fichier.
Vous avez donc été embauché pour paralléliser la production du flux compressé. Voici comment il vous est suggéré de procéder.
Vous disposez d'une mémoire partagée dans laquelle la partie (maintenant indépendante) de lecture du flux brut écrit ses données et depuis laquelle la partie (aussi indépendante) de compression puise ses données, les compresse et les écrit dans le fichier de sortie.
L'idée est que la partie qui écrit les données dans la mémoire le fait incrémentalement et par paquet : elle lit un paquet de taille fixe de données brutes et elle écrit ce paquet à la suite du paquet précédent dans la mémoire. Attention, la mémoire étant de taille fixe et plus petite que la longueur totale du flux, il faudra considérer le cas où le paquet à écrire ne peut être écrit à la suite du précédent (fin de la mémoire partagée) et devra donc être écrit en début (la mémoire est donc considérée comme un anneau de paquets de données)...
La partie qui lit, compresse et écrit les données en sortie effectue aussi sa lecture par paquet. On notera que la compression d'un paquet est assez longue en temps, en tous cas bien plus longue que l'écriture d'un paquet du flux.
Comme cela sera implanté dans une machine bi-cœur, le gain sera évident.

1) quels sont les problèmes sérieux seront à résoudre ? (liste des problèmes)
2) qu'allez-vous employer de ce que vous connaissez des systèmes Unix pour les résoudre ? (chaque problème, sa résolution).
3) écrivez le code du processus chargé de l'écriture en mémoire du flux brut (vous supposerez l'existence d'une fonction - prototype à définir vous-même - capable de lire un paquet de données brute).
4) écrivez le code du processus chargé de la lecture en mémoire d'un paquet brut, de sa compression et de son écriture dans un fichier de sortie (vous supposerez l'existence d'une fonction - prototype à définir vous-même - capable de compresser un paquet.
5) écrivez le code de la commande qui permet de lancer correctement les deux processus précédents.

Votre chef de service est très content de vous car vous avez réussi! Bien entendu, il a bien l'intention de vous presser à mort et de tirer le meilleur de vous... Il a donc songé à utiliser plus de deux cœurs pour effectuer le travail. Il a constaté que si la lecture ne peut être faite que par un seul processus (deux ne feraient pas le travail plus vite!) la compression, puisqu'elle prend plus de temps, pouvait être partagée. Les mesures effectuées lui laisse penser qu'il doit être possible de compresser sept/huit paquets en parallèle pour aller aussi vite que l'écriture du flux brut. Cela change évidemment certaines choses.

6) quels nouveaux problèmes apparaissent ?
7) qu'allez-vous employer d'Unix pour les résoudre ?
8) décrivez avec suffisamment de détail les algorithmes à employer pour régler les problèmes...