TP programmation système en C et communications

Dans chacun des exercices ci-dessous, vous prendrez soin de gérer correctement les erreurs (utiliser systématiquement `perror` *après chaque fonction système) et compiler avec l'option -Wall pour afficher tous les messages d'alerte du compilateur*

Exercice 1- *Manipulations de fichiers*

Écrire un programme C qui demande une liste de renseignements pour créer des comptes utilisateurs sur un cluster de machines: nom du compte, répertoire de login, groupe. les informations correspondantes sont stockées dans un fichier à raison d'une ligne par compte.

Écrire ce programme:

1. En utilisant les sorties formatées (`fprintf`), les informations sur chaque ligne sont séparées par des doubles points ":".
2. En utilisant les opérations d'E/S de bas niveau (`open`, `read` et `write`, `close`). Est-il nécessaire d'avoir un séparateur de champs ?
3. En utilisant les opérations d'E/S de haut niveau (`fopen`, `fread` et `fwrite`, `fclose`). Est-il nécessaire d'avoir un séparateur de champs ?

Exercice 2- *Fork et fichier*

Écrire un programme C dans lequel on lance un programme fils (par fork). Le processus père et le processus fils doivent chacun écrire dans un même fichier cinq messages successifs de la forme:

"message numero xxx du processus père de pid yyy"

ou

"message numero xxx du processus fils de pid yyy".

1. Comment sont gérées l'ouverture et la fermeture du fichier?
2. Vérifier que les zones tampon (buffer) du fichier sont transmises du père au fils au moment du fork.

Exercice 3- *Signaux*

i) Faire afficher la liste des signaux disponibles par `trap -l` (voir également le fichier `signal.h` de la bibliothèque système).

ii) Écrire deux programmes shell, l'un affichant ping l'autre affichant pong de manière à ce qu'ils se coordonnent pour afficher ping-pong-ping....

iii) Réécrire la commande Pong en C en utilisant l'interface de signaux non fiable `signal` (voir un exemple d'utilisation sur le serveur).

Exercice 4- *Timer*

Écrire un programme C qui affiche une chaîne à l'ecran et toutes les 5 secondes affiche le message bip. Pour cela, mettre en place un compte à rebours périodique par la primitive `setitimer` après avoir remplis une structure `timeval` (voir l'aide sur setitimer), ce qui a pour effet d'envoyer le signal SIGALRM avec la périodicité définie.

Exercice 5- *Tubes*

i) Tester la capacité d'un tube, c'est-à-dire combien d'octets peuvent être écrits dans le tube si personne ne lit de données dedans. Quel est le comportement de la primitive `write` quand la capacité est atteinte?

ii) En utilisant un tube, écrire un programme C qui crée un fils chargé de demander des nombres à l'utilisateur et les transmet à son père qui calcule et affiche la somme et la moyenne des nombres saisis jusqu'à présent.

Exercice 6- *Files de messages*

À partir des exemples présents sur le serveur (Essmsg_send.c et Essmsg_rec.c), écrire un serveur qui communique avec 2 clients séparément en utilisant une seule file de messages.

Exercice 7- *Sémaphores* Soient T1, T2, T3, T4, T5 et T6 des tâches qui doivent être exécutées selon l'ordre suivant :

- T1 est la tâche initiale
- T2, T3, et T4 ne commencent que lorsque T1 est terminée
- T5 ne commence que lorsque T2 et T3 sont terminées
- T6 ne commence que lorsque T4 et T5 sont terminées

Résoudre ce problème d'ordonnancement en utilisant des sémaphores en C.

N.B. Pour les exercices 6 et 7, penser à regarder/détruire les sémaphores, files de messages créées à l'aide des commandes `ipcs` et `ipcrm`

Exercice 8- *Sockets* En vous inspirant des fichiers socketClient.c et socketServer.c en exemple :

i) Créer un serveur qui, après avoir tiré un nombre au hasard, l'envoie au client (fonctions utiles `rand`, `srand` et `time`).

ii) Créer un client qui, sur réception d'un nombre, affiche le nombre à l'ecran.

iii) Modifier le client pour qu'il multiplie le nombre reçu par 2, puis l'envoie au serveur. Le serveur réceptionne le nombre et vérifie que la multiplication est correcte.

iv) Modifier client et serveur pour qu'ils réalisent l'échange précédent une centaine de fois, avec une pause d'une seconde entre chaque échange.

v) Modifier le serveur pour qu'il soit duplicatif (fork). Montrez que plusieurs clients peuvent y accéder simultanément.