Examen - Systèmes et Réseaux 1

Licence 3 Informatique

Durée : 2h. Documents personnels autorisés. Le barème est indicatif.

Exercice 1: Shell/Awk (4 pts)

1. Réecrire le commande `$CMD < $FIC` sans utiliser le symbole '<'.
2. Écrire une commande shell/awk qui affiche tous les noms de fichiers du répertoire courant dont le groupe propriétaire est identique au propriétaire.
3. Soit la ligne de commande suivante :

   ```
   tail -n 10000 "$ERRFILE" > "$T" && mv -f "$T" "$ERRFILE" || rm -f "$T"
   ```

   (a) Traduire en langage Français cette ligne de commande.

   (b) Donner, en expliquant, les résultats possibles de l'exécution de cette ligne.

   (c) Expliquer l'utilité d'une telle ligne de commande.

Exercice 2: Structure du SF (3 pts)

Soit un système de fichiers de type ext2 avec des blocs de 4 Ko et des numéros de blocs codés sur 32 bits. Chaque inœud contient 7 entrées pour des blocs directs, 1 entrée pour un bloc indirect et 1 entrée pour un bloc doublement.

1. Quelle est la taille maximum d'un fichier dans ce SF ? La taille maximum d'un fichier qui n'utilise pas de double indirection ?
2. Quelle est la taille maximum d'une partition ?
3. Indiquer, en justifiant, le nombre de blocs qui doivent être lus pour accéder à l'octet n°32560 du fichier `exam.tex`, après avoir chargé l'inœud de ce fichier.

Exercice 3: Ordonnancement de commandes (3 pts)

Soit le fichier C `exo.c` qui fait appel à la fonction `lire_entier()` qui est définie dans le fichier `lire.c`.

1. Écrire un fichier Makefile qui permet d'obtenir l'exécutable `exo` à partir des codes `exo.c` et `lire.c`.
2. On suppose maintenant que la fonction `lire_entier()` fait appel à la fonction `ecrire_entier()` définie dans le fichier `ecrire.c` et que l'on veut construire plusieurs "versions" `exo1`, `exo2`, ... basées sur les fichiers `exo1.c`, `exo2.c`, ... Écrire un fichier Makefile générique qui permet d'obtenir l'exécutable `exo`i en lançant la commane `make exo`i.
3. On suppose que l'on a exécuté la commande `make exo1` avec succès une première fois puis que l'on modifie `ecrire.c`. Quelles commandes de compilation seront lancées si l'on exécute de nouveaux `make exo1` ?

Exercice 4: QCM (3 pts)

Répondre aux affirmations suivantes par Vrai ou Faux. Toute réponse fausse vous pénalise car elle annule une réponse juste.

Q1 Les variables d'un processus sont partagées par tous ses threads.

Q2 Les threads d'un même processus partagent la même zone de pile.

Q3 La primitive `signal()` du langage C permet d'envoyer un signal à un processus.

Q4 La primitive `accept()` du langage C est bloquante par défaut.

Q5 La primitive `send()` du langage C peut être utilisée pour envoyer des données en mode non connecté.

Q6 Dans le modèle producteur/consommateurs, un consommateur et un producteur ne peuvent pas accéder au tampon partagé en même temps.

Exercice 5: Communications (7 pts)

On souhaite écrire une application sous Linux pour la gestion d'une salle d'exposition pouvant contenir au plus 20 personnes. Cette salle comporte deux portes d'entrée et une porte de sortie. L'application est constituée de 4 processus :

- Le processus *salle* qui effectue la gestion des personnes présentes dans la salle. Pour cela, il dispose d'un compteur des personnes présentes dans la salle, reçoit les indications de chaque porte sur les entrées ou sorties de personnes et envoie aux portes d'entrée des autorisation pour faire entrer une personne quand une place se libère.
- 2 processus *porte d'entrée* (1 pour chaque porte d'entrée), disposant chacun d'un compteur des personnes en attente sur cette porte. Les arrivées des personnes sont décidées par l'utilisateur (via un menu), une seule personne à la fois.
- un processus *porte de sortie* qui indique simplement au processus salle quand quelqu'un quitte la salle.

1. Quel moyen de communication entre les processus vous semble le mieux adapté ? Justifier et indiquer pour chacun des moyens non retenus, la raison principale.

   Décrire précisément la solution que vous proposez pour programmer cette application (schéma bienvenu).
2. Écrire en shell le morceaux de code de *salle* correspondant au lancement des processus de portes.
3. Écrire en shell ou en C le programme qui gère une porte d'entrée (les deux processus porte d'entrée correspondent au même programme).
4. Écrire (en shell ou en C) la boucle principale de *salle*. Lorsqu'il y a des personnes en attente à l'une des portes d'entrée et qu'une place se libère, le programme doit faire entrer quelqu'un. Si des personnes sont en attente aux deux portes, le programme choisira alternativement l'une des deux portes.
5. On souhaite maintenant avoir accès de façon distante et à tout moment au nombre de personnes qui sont passées dans la salle depuis l'ouverture (le lancement de l'application). Comment allez vous mettre en place cette nouvelle fonctionalité ? En particulier, expliciter les techniques utilisées et les modifications à apporter aux codes.