Exercices TP

Recopiez le contenu du répertoire `/Infos/lmd/2011/licence/ue/li219-2012fev/TP7` qui contient tous les fichiers nécessaires pour la réalisation de ce TP et en particulier le fichier `compte_rendu_TP7.txt`que vous devez compléter au fur et à mesure et soumettre à la fin de la séance.

Exercice 48 – Protection de section critique

Question 1

Reprenez le script `ecriture.sh` de la semaine 5 et modifiez le pour assurer que le contenu des fichiers soit cohérent lorsque plusieurs processus s'exécutent en parallèle avec les même fichiers en paramètres et quelle que soit la localisation des commutations. Votre solution doit permettre le plus de parallèlisme possible et ne pas bloquer les processus s'ils n'écrivent pas dans le même fichier.

```
#! /bin/bash
# ecriture.sh
if [ $# -lt 1 ] ; then
  echo Il faut au moins un parametre
  exit 1
fi

for elem in "$@" ; do
  if [ ! -e "$elem" ] ; then
    echo premier $$ > "$elem"
  else
    echo suivant $$ >> "$elem"
  fi
done
```

Exercice 49 – Accès concurrents à plusieurs fichiers

Soit une application nécessitant l'identification d'utilisateurs et le stockage d'informations les concernant. Le script `creation_utilisateur.sh` suivant est une première version du script de création d'un nouvel utilisateur.

```
#! /bin/bash
if [ "$#" -ne 3 ]; then
  echo Vous devez saisir trois parametres
  exit 1
fi
if [ -z "$1" ] ||  [ -z "$2" ] || [ -z "$3" ] ; then
  echo Vous devez saisir un login, un mot de passe et un nom non vide
  exit 1
fi

if [ -f login.txt ] && [ ! -z "`grep "^$1$" login.txt`" ] ; then
  echo Choisissez un login different de $1
  exit 1
fi

echo "$1" >> login.txt
echo "$2" >> pass.txt
echo "$3" >> nom.txt
```

Comme vous pouvez le constater, les informations sur les utilisateurs sont stockées dans trois fichiers différents :

1. le fichier `login.txt` contient le login de chaque utilisateur connu du système à raison d'un login par ligne,
2. le fichier `pass.txt` contient le mot de passe de chaque utilisateur, à raison d'un mot de passe par ligne,
3. le fichier `nom.txt` contient le nom de chaque utilisateur, à raison d'un nom par ligne.

Les informations sont associées en fonction de leur position dans les fichiers (les informations se trouvant à la ligne `i` de chaque fichier concernent le même utilisateur).

Question 1

Nous considérons que la base de fichiers est incorrecte si :

1. deux utilisateurs ont pu être créés avec le même identifiant (deux lignes identiques dans le fichier `login.txt`),
2. les informations concernant un utililisateur ne sont pas à la même ligne dans les trois fichiers.

Expliquez précisément comment chacun des cas précédents peut se produire.

Question 2

Quelles sont les ressources critiques du script `creation_utilisateur.sh` ? Justifiez votre réponse et déterminez les sections critiques.

Question 3

Modifiez le script `creation_utilisateur.sh` en utilisant un seul verrou pour protéger la(les) section(s) critique(s).

Question 4

Écrivez une nouvelle version du script `creation_utilisateur.sh` en utilisant 3 verrous différents pour permettre plus de parallélisme entre les prosessus. On veut que les processus ne se "doublent" pas tout en pouvant faire des actions en parallèle (un processus peut modifier le fichier `nom.txt` pendant qu'un autre modifie le fichier `login.txt`). Assurez-vous de toujours protéger les sections critiques. **Attention**, la solution de remplacer l'instruction de pose de l'unique verrou par les instructions de pose des trois verrous n'est pas bonne, elle n'assure pas plus de parallèlisme !

Exercice 50 – Problème de la piscine

ATTENTION, dans cet exercice, lorsque vous devrez poser un verrou, vous devrez utiliser la commande `lockfile -1 <nom_verrou>` pour que le processus n'attende qu'une seconde avant de pouvoir à nouveau essayer de poser le verrou.

Nous considérons un ensemble de processus représentant les usagers d'une piscine. La création d'un processus correspond à l'arrivée de l'usager à la piscine. L'usager doit alors :

1. trouver une cabine disponible,
2. prendre un panier pour y déposer ses vêtements,
3. se changer dans la cabine et la libérer.

L'usager peut alors nager. Une fois sa baignade terminée, le nageur doit :

1. trouver une cabine vide,
2. sortir ses affaires du panier et rendre celui-ci,
3. se changer puis libèrer la cabine.

Pour suivre la disponibilité des ressources, nous disposons d'un fichier par type de ressource (`/tmp/numéro_étudiant/nb_cabines` pour les cabines et `/tmp/numéro_étudiant/nb_paniers` pour les paniers où vous devez remplacer `numéro_étudiant` par votre propre numéro). Chacun des fichiers contient un entier qui correspond au nombre de ressources disponibles. Vous devrez créer le répertoire `/tmp/numéro_étudiant` et les fichiers qu'il contient.

– Pour prendre une ressource il faut :
  `lire le contenu du fichier` (pour savoir combien de ressources sont disponibles)
  `tant que la valeur lue n'est pas au moins égale à 1`
    `forcer une commutation` (1 seconde d'interruption en espérant la libération d'une ressource)
    `lire à nouveau le contenu du fichier`
  `fin tant que`
  `mettre à jour le contenu du fichier` (pour signaler qu'une ressource de moins est disponible)
– Pour libérer une ressource il faut :
  `mettre à jour le contenu du fichier` (pour signaler qu'une ressource de plus est disponible)

Remarque : nous utilisons des fichiers se trouvant dans le répertoire `/tmp`. Le répertoire `/tmp` est local à la machine sur laquelle vous êtes connecté, il est accessible à tous les utilisateurs se connectant sur l'ordinateur, c'est pour cela que vous devez créer un répertoire qui vous est propre. L'accès aux fichiers du répertoire `/tmp` se fait via le serveur de fichiers de la machine sur laquelle vous êtes connecté, ce qui n'est pas le cas pour les fichiers se trouvant dans votre `homedirectory`. Ne dépendre que du serveur de fichiers local réduit la possibilité que survienne un problème dû à la non-atomicité des écritures lors des redirections >. Ces cas ne sont pas ceux qui nous intéressent. Ce ne sont donc pas eux qu'il faut expliquer dans vos réponses.

Question 1
Donnez les instructions qui correspondent à la prise d'une cabine et celles qui correspondent à sa libération.

Question 2
Soit l'algorithme suivant réalisé par le script `usager.sh` qui représente les usagers de la piscine :

```
echo "Arrivee de l'usager $$"
instructions de prise d'une cabine
instructions de prise d'un panier
instructions de libération d'une cabine
echo "Usager $$ se baigne"
forcer une commutation et attendre 3 secondes (pour représenter le temps de la baignade)
instructions de prise d'une cabine
instructions de libération d'un panier
instructions de libération d'une cabine
echo "Fin de $$"
```

Ecrivez le script `usager.sh` correspondant à l'algorithme précédent et un script `lancement.sh` qui :

1. rend disponibles 5 paniers (le contenu du fichier `/tmp/numéro_étudiant/nb_paniers` doit être 5),
2. rend disponibles 3 cabines (le contenu du fichier `/tmp/numéro_étudiant/nb_cabines` doit être 3),
3. lance l'exécution en parallèle de 7 usagers,
4. attend la fin des usagers avant de se terminer (indice : tous les processus usagers sont des fils du processus de lancement).

Attention, votre solution doit permettre de facilement modifier le nombre de ressources disponibles et le nombre d'usagers créés.

Exécutez le script `lancement.sh` plusieurs fois, jusqu'à ce que le contenu des fichiers `/tmp/numéro_étudiant/nb_paniers` et `/tmp/numéro_étudiant/nb_cabines` ne soient pas cohérents avec le nombre initial des ressources. Comment expliquez-vous les valeurs contenues dans les fichiers `/tmp/numéro_étudiant/nb_paniers` et `/tmp/numéro_étudiant/nb_cabines` obtenues à la fin de l'exécution ?

Question 3
Pour éviter les problèmes identifiés en question 2, nous protégeons les instructions de prise d'une ressource (et de libération) par un verrou (un verrou par type de ressource) :

– Lors de la prise d'une ressource, nous choisissons de poser le verrou avant la première lecture dans le fichier associé et de le supprimer après la mise à jour du contenu du fichier.
– Lors de la libération d'une ressource, le verrou est posé avant la première instruction et supprimé après la dernière.

Ecrivez une nouvelle version du script `usager.sh`.

ATTENTION, avant chaque nouvelle exécution du script `lancement.sh` vous devez vous assurer que :
– tous les processus `usager.sh` et `lancement.sh` sont terminés (pour le vérifier utilisez la commande `ps`),
– les fichiers verrous ont tous été supprimés.

Exécutez plusieurs fois le script `lancement.sh` jusqu'à ce que vous constatiez un blocage. Pour faciliter l'apparition de ce blocage, ajoutez, dans le script `usager.sh`, l'instruction `sleep 5` juste après la première prise d'une cabine. Expliquez la cause de ce blocage (indice : regardez le contenu des fichiers `nb_cabines`, `nb_paniers` ainsi que les verrous correspondants) ? **Attention**, une explication n'est pas "L'exécution se bloque à cause de la valeur du contenu des fichiers". Elle doit expliquer l'enchaînement des actions qui a mené à la situation constatée.

Question 4

Ecrivez une nouvelle version du script `usager.sh` qui ne présente pas les problèmes identifiés dans les questions précédentes.

Question 5

Modifiez le script `usager.sh` en supprimant l'instruction `sleep 5` que vous avez ajoutée dans la question 3 et faites passer le temps de baignade à 15 secondes. Testez votre nouveau script pour 10 usagers, 5 paniers et 3 cabines (modifiez le script `lancement.sh`). Expliquez pourquoi l'exécution se bloque à nouveau (regardez le contenu des fichiers `nb_cabines` et `nb_paniers`). **Attention**, là encore, une explication n'est pas "L'exécution se bloque à cause de la valeur du contenu des fichiers". Elle doit expliquer l'enchaînement des actions qui a mené à la situation constatée.

Question 6

Que se passe-t-il, si on inverse l'ordre de prise du panier et de la cabine ? Un usager commence par prendre un panier avant de prendre une cabine pour se changer. Il libère alors la cabine, se baigne, prend à nouveau une cabine, se change puis libère le panier et la cabine.