

ISS - Initiation aux Systèmes d'exploitation et au Shell

LU2IN020

TP 10 – Concurrence, incohérence et synchronisation

Julien Sopena

décembre 2022

Le but de cette dixième semaine est de mettre évidence que la commutation entre processus peut être à l'origine d'incohérence sur des ressources partagées. Nous y verrons aussi comment répondre à ce problème en utilisant un mécanisme de verrouillage.

Exercice 1 : Attention commutation

Dans cet exercice on considère le script suivant

ecriture.sh

```
#!/bin/bash
# écriture.sh
if [ $# -lt 1 ] ; then
    echo Il faut au moins un paramètre
    exit 1
fi
for elem in "$@" ; do
    if [ ! -e "$elem" ] ; then
        echo premier $$ > "$elem"
    else
        echo suivant $$ >> "$elem"
    fi
done
```

Question 1

Pour commencer, chargez l'archive LU2IN020-TP_10.tgz à l'URL suivante : http://julien.sopena.fr/LU2IN020-TP_10.tgz. Vous y trouverez l'exécutable `iss_synchro` nécessaire à la réalisation de ce TP, ainsi que celui de la semaine suivante.

Avant d'aller plus loin, assurez que cet exécutable soit accessible depuis n'importe quel terminal ou script que vous lancerez.

Question 2

Exécutez deux fois de suite la commande suivante et expliquez le contenu des fichiers `a`, `b` et `c` obtenus.

```
moi@pc /home/moi $ ./ecriture.sh a b c
```

Question 3

Soit le script `lancement_ecriture.sh` suivant qui permet d'exécuter en concurrence deux processus `ecriture.sh` :

`lancement_ecriture.sh`

```
#!/bin/bash
# lancement_ecriture.sh
if [ -f f1 ] ; then
    rm f1
fi
if [ -f f2 ] ; then
    rm f2
fi
if [ -f f3 ] ; then
    rm f3
fi
./ecriture.sh f1 f2 f3 & ./ecriture.sh f1 f2 f3
```

Exécutez le script `lancement_ecriture.sh` jusqu'à ce que le contenu des fichiers `f1`, `f2` et `f3` obtenus soit différent de celui obtenu à la question précédente (la différence ne doit bien sûr pas se limiter à la valeur des PID). Expliquez le résultat obtenu et dites pourquoi il n'est pas satisfaisant.

Question 4

Modifiez le script `ecriture.sh` pour que l'exécution de la commande `lancement_ecriture.sh` donne obligatoirement un résultat non satisfaisant (vous devez forcer une commutation à un moment judicieusement choisi).

Question 5

Pour que l'exécution de la commande précédente ne puisse pas produire de résultats non satisfaisants, nous proposons la solution suivante dont le principe est qu'un processus qui souhaite accéder à un fichier bloque tous les autres processus tant que nécessaire. Voici les étapes à respecter :

- Avant d'accéder pour la première fois à un fichier, un processus doit envoyer un signal `SIGSTOP` aux autres processus qui souhaitent accéder à un fichier (quel qu'il soit), de manière à bloquer leur exécution.
- Pour que les identités de ces processus soient connues, un processus qui veut accéder à un fichier doit "s'inscrire", c'est-à-dire écrire son identité (PID) dans un fichier commun à tous les processus (`fic_PID`) et dédié aux inscriptions.
- Une fois que le processus a fini de travailler sur le fichier, il réveille les processus stoppés en leur envoyant un signal `SIGCONT`. De cette façon, tous les processus sauf un sont bloqués lors des accès aux fichiers.

Chaque processus doit effectuer son inscription avant toute autre instruction. De plus, vous devrez veiller à ce qu'un processus ne s'envoie par un signal à lui-même.

Modifiez le script `ecriture.sh` et vérifiez que le résultat de l'exécution de la commande `lancement_ecriture.sh` est correct.

Question 6

Cette "correction" n'est pas un bon moyen de résoudre les problèmes mis en évidence dans cet exercice. Quels sont les défauts que vous identifiez ?

Question 7

Proposer une solution basée sur le verrouillage qui permette d'assurer que le contenu des fichiers soit cohérent lorsque plusieurs processus s'exécutent en parallèle avec les mêmes fichiers en paramètres et quelques soit les commutations.

Votre solution doit permettre le plus de parallélisme possible et ne pas bloquer les processus s'ils n'écrivent pas dans le même fichier. Pour simplifier, votre test sera lancé par script qui générera préalablement tous les verrous nécessaires.

Exercice 2 : Ressources multiples

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.

`creation_utilisateur.sh`

```
#!/bin/bash
if [ "$#" -ne 3 ]; then
    echo "Vous devez saisir trois paramètres"
    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 différent 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 utilisateur 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 processus. 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 parallélisme !