DIU-EIL : Exercices pratiques de systèmes d'exploitation

Jonathan Lejeune

Juin-Juillet 2019

Introduction

Ce document est un fascicule d'exercices de difficulté croissante illustrant les différents mécanismes des systèmes d'exploitation avec le langage de commande `bash`. En ce qui concerne le système d'exploitation, le programme du lycée tel qu'il est connu à ce jour est le suivant :

- **en première** :
  - ◇ Identifier les fonctions d'un système d'exploitation.
  - ◇ Utiliser les commandes de base en ligne de commande.
  - ◇ Gérer les droits et permissions d'accès aux fichiers.

  Les élèves utilisent un système d'exploitation libre et il ne s'agit pas d'une étude théorique des systèmes d'exploitation
- **en terminal** :
  - ◇ Gestion des processus et des ressources par un système d'exploitation.
  - ◇ Décrire la création d'un processus
  - ◇ l'ordonnancement de plusieurs processus par le système
  - ◇ Mettre en évidence le risque de l'interblocage (deadlock)

  À l'aide d'outils standard, il s'agit d'observer les processus actifs ou en attente sur une machine. Une présentation débranchée de l'interblocage peut être proposée.

Ce document comporte 4 séries d'exercices :

- Prise en main du shell
- Manipulation de fichiers
- Processus
- Synchronisation

Un complément du cours sur la synchronisation est donné en annexe de ce document. Il est nécessaire d'avoir lu et compris cette annexe pour commencer la quatrième série d'exercices.

N.B. : Une majorité d'exercices présentés dans ce fascicule sont issus de l'unité d'enseignement *"Introduction aux systèmes d'exploitation"* dispensée en deuxième année de licence informatique de Sorbonne Université.

Prise en main du shell

Exercice 1 – Paramètres et début des alternatives

Notions travaillées :

- **paramètre d'un script**
- **structure de contrôle conditionnelle**

Question 1

Écrivez un script `param.sh` qui affiche le nombre de paramètres avec lequel le script a été appelé ainsi que la liste des paramètres.

Question 2

Modifiez votre script pour que la liste des paramètres ne soit affichée que si elle n'est pas vide (il doit y avoir au moins un paramètre).

Question 3

Modifiez votre script pour qu'il affiche la liste des paramètres sans les deux premiers. Il devra en plus faire les tests suivants :

- si le nombre de paramètres est inférieur strictement à 3, il s'interrompt après avoir affiché un message d'erreur,
- si le premier paramètre n'est pas un fichier, il s'interrompt après avoir affiché un message d'erreur,
- si le deuxième paramètre n'est pas un répertoire, il s'interrompt après avoir affiché un message d'erreur.

Exercice 2 – Alternatives

Notions travaillées :

- **test en shell**
- **structure de contrôle conditionnelle**

Question 1

Écrivez un script `alternatives.sh` qui prend en paramètre un entier. Si le nombre de paramètre est différent de 1, une erreur est affichée et le script s'arrêtera. Sinon il devra afficher :

- `Vous voyagez gratuitement`, si l'âge est inférieur ou égal à 3,
- `Vous payez demi tarif`, si l'âge est supérieur ou égal à 4 et inférieur ou égal à 11,
- `Vous payez plein tarif`, si l'âge est supérieur ou égal à 12 et inférieur ou égal à 59,
- `Vous bénéficiez de 25 pourcent de réduction`, si l'âge est supérieur ou égal à 60.

Exercice 3 – À la recherche du texte perdu

Notions travaillées :

- **méta-caractères**
- **contenu d'un fichier**

Le répertoire `MetaCaracteres` vous est fournit. Avec la commande `ls MetaCaracteres`, vous constaterez alors que ce répertoire contient beaucoup d'éléments. Les méta-caractères permettent de déterminer des motifs à partir desquels une sélection des éléments affichés peut être effectuée. Voici 3 éléments de motifs possibles :

`*` : représente n'importe quelle suite de caractères (vide ou non),

`?` : représente n'importe quel caractère (exactement un),

`[liste de caractères]` : représente n'importe quel caractère de la liste (exactement un). La liste peut-être un intervalle (`a-z` par exemple), le code ASCII des caractères est pris en considération pour définir les caractères appartenant à l'intervalle.

Question 1

Déterminez les éléments du répertoire `MetaCaracteres` dont le nom correspond à chacun des motifs suivants et expliquez brièvement les caractéristiques des éléments affichés.

```
Prompt% ls *
Prompt% ls  ?
Prompt% ls [a-c]?
Prompt% ls [ac]?
Prompt% ls [a-c]*
Prompt% ls *.txt
```

Question 2

L'énoncé de cette question se trouve dans le fichier `MetaCaracteres/questionTP1`. La commande `cat` suivie d'un nom de fichier vous permettra d'afficher le contenu du fichier, et donc de pouvoir le lire et de suivre les indications que vous y trouverez.

Exercice 4 – Ré-écriture de la commande `which`

Notions travaillées :

- **Variable `PATH`**
- **Droits d'exécution**

On souhaite écrire un script `which.sh` qui a le même comportement que la commande `which`. Ce script prend un ou plusieurs paramètres. Pour chaque paramètre, le script affiche le nom absolu du premier fichier exécutable trouvé, lors du parcours de la variable `PATH`, dont le nom est celui du paramètre. Le script affiche que le fichier est introuvable s'il n'apparaît dans aucun des répertoires de la variable `PATH`.

Une première étape consiste à identifier les différents répertoires composant la variable `PATH`. Attention, un nom de répertoire peut contenir un espace.

Le script `repPATH.sh` suivant affiche la liste des répertoires de `PATH`, un par ligne.

exercices/which/repPATH.sh

```
#! /bin/bash
# script repPATH.sh

# "for" separe ses arguments sur les espaces, donc on transforme ':'
# en ' ' (la commande grep fait des substitutions de carcateres).
# D'autre part les repertoire de PATH peuvent contenir des espaces,
# mais pas de caractere':' donc pour ne pas troubler "for" on transforme
# ' ' en ':', puis dans la boucle on fait l'operation inverse.

for rep in `echo "$PATH" | tr ": " " :"`
do
  rep=`echo "$rep" | tr : " "`
  echo $rep
done
```

Question 1

Écrivez le script `which.sh`. La condition `-x nom_fichier` est vraie si `nom_fichier` a les droits positionnés en exécution. Attention, cette condition ne permet pas de différencier les fichiers des répertoires.

Exercice 5 – Commande `shift`

Notions travaillées :

- commande shift
- tests sur les types de fichier

La commande `shift` permet de supprimer le(s) premier(s) paramètre(s). Elle entraîne un décalage des paramètres qui restent. Elle est très utile lorsque les premiers paramètres d'un script ont une signification particulière qui fait qu'ils ne sont pas traités de la même manière que tous les autres.

Question 1

Écrivez un script `rep_fichier.sh` qui :

- prend au moins deux paramètres,
- vérifie que le nombre de paramètres est correct, arrête son exécution si ce n'est pas le cas,
- vérifie que le premier paramètre est un répertoire, arrête son exécution si ce n'est pas le cas,
- affiche pour chacun des paramètres suivant s'il est ou non un fichier du répertoire passé en premier paramètre.

Question 2

Écrivez un script `supprimer.sh` qui supprime certains fichiers du répertoire passé en paramètre. Ce script prend au moins un paramètre (le répertoire). Les paramètres suivants indiquent les extensions des fichiers à supprimer (les caractères qui suivent le point en fin de nom). S'il y a un seul paramètre, tous les fichiers du répertoire sont à supprimer. Dans tous les cas, le script ne supprime que des fichiers et pas de répertoire.

Votre script doit vérifier les paramètres (nombre, que le premier correspond bien à un répertoire). Voici un exemple d'exécution dans laquelle `repTest` et `rep1` sont les seuls répertoires, tous les autres éléments sont des fichiers.

```
Prompt% ls repTest
fichier1  fichier2  image1.jpg  image2.jpg  musique1.mp3  rep1
Prompt% ./supprimer.sh repTest jpg mp3
Prompt% ls repTest
fichier1  fichier2
Prompt% ./supprimer.sh repTest
Prompt% ls repTest
rep1
Prompt%
```

Pour tester votre script, vous pouvez créer des fichiers en utilisant la commande `touch nom_fichier`.

Manipulation de fichiers

Exercice 6 – Affichage du nom absolu des fichiers d'un répertoire

Notions travaillées :

- **parcours de répertoire**
- **chemin absolu**

Nous souhaitons écrire un script `absolu.sh` qui prend en argument une liste de noms de répertoires. Le script prend en compte les différents appels possibles (corrects ou non).

Le but de ce script est d'afficher le nom absolu de tous les fichiers contenus dans les répertoires passés en arguments. Nous faisons l'hypothèse (ce ne sera donc pas à tester) que les noms des répertoires passés en paramètres ne contiennent aucun '.' (pas de référence au répertoire courant ni au répertoire père) et sont donnés en nom relatif.

Voici un algorithme permettant de réaliser ce script :

```
Si le nombre de paramètres n'est pas correct
   afficher un message d'erreur et terminer le script
Fin si
Si il existe un paramètre qui ne correspond pas à un répertoire
   afficher un message d'erreur et terminer le script
Fin si

Pour chaque répertoire r passé en paramètre
   Pour chaque sous-fichier f de r
      si f est un fichier
         Afficher le nom absolu du fichier
      Fin si
   Fin pour
Fin pour
```

Question 1

Écrivez le script `absolu.sh` réalisant l'algorithme précédent.

Exercice 7 – Compter les éléments d'un répertoire

Notions travaillées :

- **parcours de répertoire**
- **parcours d'une liste de paramètres**

Nous souhaitons écrire un script `compte.sh` qui prend en argument une liste de noms de répertoires. Le script prend en compte les différents appels possibles (corrects ou non).

Le but de ce script est d'afficher le nombre total de fichiers ainsi que le nombre total de répertoires contenus dans l'ensemble des répertoires passés en paramètre. La commande n'est pas récursive : on n'examinera pas les sous-répertoires.

Voici un algorithme permettant de réaliser ce script. Les variables `totalFichiers` et `totalRepertoires` permettent de stocker respectivement le nombre de fichiers et le nombre de répertoires :

```
Si le nombre de paramètres n'est pas correct
  afficher un message d'erreur et terminer le script
Fin si
Initialiser la variable totalFichiers à 0
Initialiser la variable totalRepertoires à 0
Pour chaque paramètre p
  si p est un répertoire
     Pour chaque sous-fichier f du répertoire
      Si f est un fichier
        Augmenter la variable totalFichiers de 1
      sinon
        Augmenter la variable totalRepertoires de 1
      Fin si
     Fin pour
  sinon
     Afficher un message d'erreur
  Fin si
Fin pour
Afficher la valeur des variables totalFichiers et totalRepertoires
```

Question 1

Écrivez le script `compte.sh` réalisant l'algorithme précédent.

Question 2

Écrivez un script `compte_max.sh` qui prend en argument une liste de noms de répertoires.

Le but de ce nouveau script est d'afficher le nom du répertoire contenant le plus de fichiers et celui du répertoire contenant le plus de répertoires. Ce script prend en compte les différents appels possibles (corrects ou non).

Exercice 8 – Déplacer des fichiers

Notions travaillées :

- **parcours de répertoire**
- **modification d'arborescence de fichiers**

Nous souhaitons écrire un script `deplacer.sh` qui prend au moins trois arguments, les deux premiers sont des noms de répertoires et les suivants des noms de fichiers.

Le but de ce script est de déplacer du premier répertoire vers le deuxième répertoire les fichiers donnés en paramètre.

Voici un algorithme permettant de réaliser ce script. On utilise une variable `repSource` pour stocker le nom du répertoire dans lequel se trouvent les fichiers à déplacer et une variable `repDest` pour stocker le nom du répertoire dans lequel doivent être déplacés les fichiers.

Voici un algorithme permettant de réaliser ce script. On utilise une variable `repSource` pour stocker le nom du répertoire dans lequel se trouvent les fichiers à déplacer et une variable `repDest` pour stocker le nom du répertoire dans lequel doivent être déplacés les fichiers.

```
Si le nombre de paramètres n'est pas correct
  afficher un message d'erreur et terminer le script
Fin si
```

```
Donner sa valeur à repSource
Si repSource n'est pas un répertoire
   afficher un message d'erreur et terminer le script
Fin si

Donner sa valeur à repDest
Si repDest n'est pas un répertoire
  afficher un message d'erreur et terminer le script
Fin si

Supprimer le nom des répertoires de la liste des paramètres

Pour chaque paramètre p
  si p est un fichier de repSource
      si repDest ne contient aucun élément de même nom que le fichier
           déplacer le fichier de repSource vers repDest
      Fin si
  sinon
      afficher un message d'erreur
  Fin si
Fin pour
```

Question 1

Écrivez le script `deplacer.sh` réalisant l'algorithme précédent.

Question 2

Nous souhaitons écrire un script `deplacer2.sh` qui a un comportement identique au script `deplacer.sh` et qui lorsque le fichier à déplacer existe déjà dans le répertoire `repDest`, demande à l'utilisateur s'il souhaite écraser le fichier existant. Si l'utilisateur répond `oui`, le fichier du répertoire `repDest` est détruit et celui du répertoire `repSource` est déplacé vers le répertoire `repDest`. Si l'usager répond autre chose que oui, rien n'est modifié.

Exercice 9 – Exploration d'une arborescence

Notions travaillées :

- **parcours d'arborescence**
- **contenu de fichier**

Question 1

Écrivez le script `explorer.sh` qui prend un argument. Son but est de descendre le long d'une branche de l'arbre des répertoires et fichiers à partir de l'élément passé en paramètre. À chaque niveau, le script affiche le contenu du répertoire et demande à l'utilisateur quel élément il veut explorer ou s'il veut s'arrêter (saisie de la chaîne de caractères "fin"). Si l'utilisateur choisit un fichier, le script affiche son contenu et s'arrête. Si l'utilisateur choisit un répertoire vide le script s'arrête, si le répertoire n'est pas vide alors l'exploration recommence à partir de ce répertoire.

Exercice 10 – Lire un fichier

Notions travaillées :

- **ouverture de fichier**

- lecture de fichier

Question 1

Écrivez un script `lecteur_ligne.sh` qui prend un nom de fichier texte en paramètre et qui le lit ligne par ligne. Pour chaque ligne lue on affichera le numéro de la ligne suivi du contenu de la ligne

Question 2

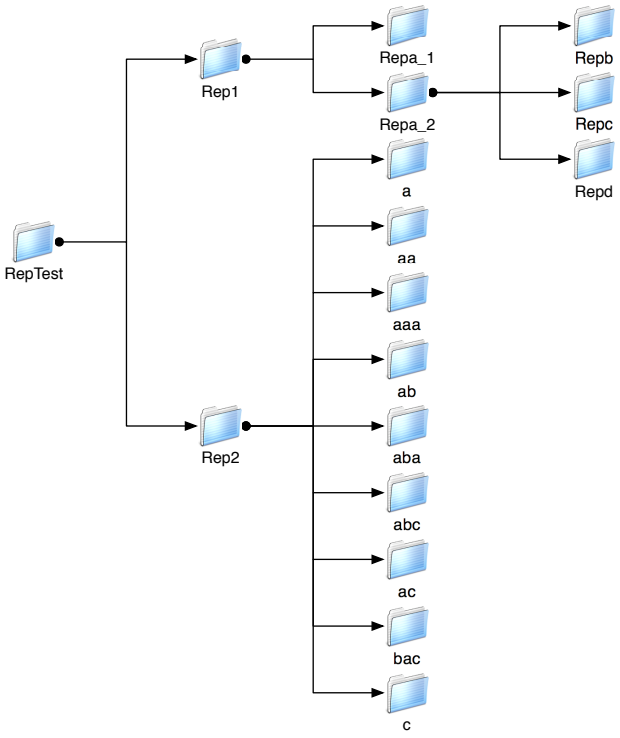
Écrivez un script `lecteur_mot.sh` qui prend un nom de fichier texte en paramètre et qui le lit mot par mot. Pour chaque mot lu on affichera le numéro de ce mot suivi du mot lui même.

Exercice 11 – Manipulation d'une arborescence de fichiers

Notions travaillées :

- construction d'une arborescence de fichiers
- lister les éléments d'une arborescence de fichiers

En répondant aux questions suivantes, vous allez regrouper dans un fichier nommé `creation.sh` les commandes permettant de construire l'arborescence donnée ci-dessous :



Question 1

Écrivez dans le fichier `creation.sh` la liste des commandes (une par ligne) permettant la création de l'arborescence qui vous est demandée (elle ne contient que des répertoires). Sauvegardez le fichier et exécutez le script ainsi créé.

Question 2

Vérifiez que l'arborescence créée est conforme à celle qui vous a été demandée en exécutant la commande :

```
Prompt% ls -R RepTest
```

Question 3

Les commandes suivantes sont à exécuter dans l'ordre et dans le répertoire `RepTest/Rep1/Repa_2`. Pour chacune d'elle dites quel est son effet :

1. `Prompt% mv Repb Repe`
2. `Prompt% mv Repe Repc/`
3. `Prompt% mv Repd Repc`

Question 4

Déplacez le répertoire `Repa_2` dans le répertoire `RepTest/Rep2` en le renommant `Rep2_1`.

Question 5

Listez les éléments de `RepTest` dont le nom commence par `Rep`. Que constatez vous ?

Question 6

Placez vous dans le répertoire `RepTest/Rep2`.

Question 7

Exécutez la commande vous permettant de vérifier que vous êtes bien dans le répertoire souhaité.

Question 8

Listez les éléments de `RepTest/Rep2` dont le nom :

1. se termine par `a`,
2. commence par `b`,
3. est composé de 2 caractères exactement,
4. contient le caractère `a` en deuxième position.

Exercice 12 – Rapport sur arborescence

Notions travaillées :

- paramètres
- commandes de manipulation d'arborescence de fichiers

Question 1

Écrivez un script `contenuRepertoires.sh` qui :

- prend zéro, un ou plusieurs paramètres,
- s'il n'y a pas de paramètre, affiche le nom absolu du répertoire courant,
- s'il y a au moins un paramètre, pour chacun des paramètres
  - ◇ s'il s'agit d'un répertoire, affiche son nom relatif en disant qu'il s'agit d'un répertoire,
  - ◇ s'il s'agit d'un fichier, affiche son nom relatif en disant qu'il s'agit d'un fichier,
  - ◇ s'il s'agit ni d'un fichier, ni d'un répertoire, affiche un message d'erreur et passe au paramètre suivant.

Voici un exemple d'exécution :

```
Prompt% pwd
/users/jon/Quest1
Prompt% ls -Rl

total 8
drwxr-xr-x  5 jon  staff  170  3 Nov 15:00 Rep1
drwxr-xr-x  6 jon  staff  204  3 Nov 15:00 Rep2
-rwxr-xr-x  1 jon  staff  810  3 Nov 15:51 contenuRepertoires.sh

./Rep1:
total 0
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 a
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 b
```

```
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 c

./Rep2:
total 0
drwxr-xr-x  4 jon  staff  136  3 Nov 15:00 Rep2_1
-rw-r--r--  1 jon  staff    0  3 Nov 15:00 d
-rw-r--r--  1 jon  staff    0  3 Nov 15:00 e
-rw-r--r--  1 jon  staff    0  3 Nov 15:00 f

./Rep2/Rep2_1:
total 0
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 a
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 b
```

```
Prompt% ./contenuRepertoires.sh Rep1 Rep2/Rep2_1 b Rep1/a
Rep1 est un repertoire
Rep2/Rep2_1 est un repertoire
b n est ni un fichier ni un repertoire
Rep1/a est un fichier
Prompt% ./contenuRepertoires.sh
Repertoire courant : /users/jon/Quest1
```

Question 2

Écrivez le script `contenuRepertoires1.sh` pour qu'en plus de ce que fait le script `contenuRepertoires.s` il affiche le nombre d'éléments contenus dans un répertoire lorsqu'il affiche qu'un élément est un répertoire (et aussi dans le cas du répertoire courant).
Voici un exemple d'exécution :

```
Prompt% pwd
/users/jon/Quest2
Prompt% ls -Rl
```

```
total 8
drwxr-xr-x  5 jon  staff  170  3 Nov 15:00 Rep1
drwxr-xr-x  6 jon  staff  204  3 Nov 15:00 Rep2
-rwxr-xr-x  1 jon  staff  810  3 Nov 15:51 contenuRepertoires1.sh

./Rep1:
total 0
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 a
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 b
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 c

./Rep2:
total 0
drwxr-xr-x  4 jon  staff  136  3 Nov 15:00 Rep2_1
-rw-r--r--  1 jon  staff    0  3 Nov 15:00 d
-rw-r--r--  1 jon  staff    0  3 Nov 15:00 e
-rw-r--r--  1 jon  staff    0  3 Nov 15:00 f

./Rep2/Rep2_1:
total 0
```

```
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 a
-rw-r--r--  1 jon  staff  0  3 Nov 15:00 b

Prompt% ./contenuRepertoires1.sh Rep1 Rep2/Rep2_1 b Rep1/a
Rep1 est un repertoire : 3 elements
Rep2/Rep2_1/ est un repertoire : 2 elements
b n est ni un fichier ni un repertoire
Rep1/a est un fichier
Prompt% ./contenuRepertoires1.sh
Repertoire courant : /users/jon/Quest2 : 3 elements
```

Exercice 13 – Création d'une arborescence à partir d'un fichier

Notions travaillées :

- **lecture de fichier**
- **création d'une arborescence**

Nous souhaitons écrire un script `creationArboFichier.sh` qui prend un nom de fichier en paramètre. Le script doit signaler les appels incorrects (nombre de paramètres différent de un, ou un paramètre qui n'est pas un fichier).

Le fichier en paramètre doit contenir une liste de noms (sans espace), à raison d'un nom par ligne. Vous supposerez que le contenu du fichier respecte ces contraintes, ce n'est pas à tester.

Le but du script est de créer dans le répertoire courant, pour chaque nom lu dans le fichier, un répertoire de ce nom. Si un élément de même nom existe déjà (répertoire ou fichier), un message est affiché pour le signaler.

Voici un algorithme permettant de réaliser ce script :

```
Tester le nombre de parametres
  si le nombre n'est pas correct
    afficher un message d'erreur et terminer le script.

Tester si le parametre est un fichier
  si ce n'est pas le cas
    afficher un message d'erreur et terminer le script.

Debut boucle
# La boucle doit permettre, a chaque tour, de recuperer
#   le nom suivant dans le fichier passe en parametre.
  S'il ne s'agit pas du nom d'un  element du repertoire courant
    Creer le repertoire avec le bon nom
    Afficher un message
  Sinon
    Afficher un message et continuer
  Fin si
Fin boucle
```

Voici une exécution possible du script `creationArboFichier.sh` :

```
Prompt% cat ficNoms
rep1
rep2
```

```
rep3
Prompt% ls -l
-rwxr-xr-x 1 usr1 usr1 378 8 sep 15:28 creationArboFichier.sh
-rw-r-r- 1 usr1 usr1 15 9 sep 15:18 ficNoms
-rw-r-r- 1 usr1 usr1 0 9 sep 15:17 rep2
Prompt% ./creationArboFichier.sh ficNoms
Le repertoire rep1 a ete cree
Un element de nom rep2 existe deja
Le repertoire rep3 a ete cree
Prompt% ls -l
-rwxr-xr-x 1 usr1 usr1 378 8 sep 15:28 creationArboFichier.sh
-rw-r-r- 1 usr1 usr1 15 9 sep 15:18 ficNoms
drwxr-xr-x 2 usr1 usr1 68 9 sep 15:20 rep1
-rw-r-r- 1 usr1 usr1 0 9 sep 15:17 rep2
drwxr-xr-x 2 usr1 usr1 68 9 sep 15:20 rep3
```

Question 1

Écrivez le script shell réalisant l'algorithme précédant.

Question 2

Nous souhaitons écrire un script `creationArboFichier2.sh` qui comme précédemment prend un nom de fichier en paramètre. Chaque ligne du fichier est de la forme `val1 val2`. Si `val1` vaut `d`, le script crée un répertoire de nom `val2`, si `val1` vaut `f`, le script crée un fichier de nom `val2`, pour toute autre valeur de `val1`, un message d'erreur est affiché. De même, si un élément de nom `val2` existe déjà dans le répertoire courant, un message est affiché.

Voici une exécution possible du script `creationArboFichier2.sh`.

```
Prompt% cat ficNoms
d rep5
f fic1
f fic2
d fic1
f rep5
e rep8
d rep7
Prompt% ls -l
-rwxr-xr-x 1 usr1 usr1 378 8 sep 15:28 creationArboFichier.sh
-rwxr-xr-x 1 usr1 usr1 628 8 sep 17:00 creationArboFichier2.sh
-rw-r-r- 1 usr1 usr1 49 9 sep 15:18 ficNoms
Prompt% ./creationArboFichier2.sh ficNoms
Le repertoire rep5 a ete cree
Le fichier fic1 a ete cree
Le fichier fic2 a ete cree
Un element de nom fic1 existe deja
Un element de nom rep5 existe deja
e : type inconnu
Le repertoire rep7 a ete cree
Prompt% ls -l
-rwxr-xr-x 1 usr1 usr1 378 8 sep 15:28 creationArboFichier.sh
-rwxr-xr-x 1 usr1 usr1 628 8 sep 17:00 creationArboFichier2.sh
```

```
-rw-r-r- 1 usr1 usr1 0 9 sep 16:41 fic1
-rw-r-r- 1 usr1 usr1 0 9 sep 16:41 fic2
-rw-r-r- 1 usr1 usr1 49 9 sep 16:40 ficNoms
drwxr-xr-x 2 usr1 usr1 68 9 sep 16:41 rep5
drwxr-xr-x 2 usr1 usr1 68 9 sep 16:41 rep7
```

Processus

Exercice 14 – Passage de paramètres et exportation de variables

Notions travaillées :

- **création de processus**
- **transmission de données processus père vers processus fils**

Question 1

- Écrivez un script `pere_para.sh` qui prend en paramètre un entier et crée un nombre égal à cet entier de processus fils, en transmettant **en paramètres** à chacun sa position dans l'ordre de création et le nombre total de fils qui doivent être créés.
- Écrivez le script `fils_para.sh` exécuté par chaque processus fils et qui affiche le numéro (i.e. la position du processus) et le nombre total de processus créés.

Voici un exemple d'exécution :

```
Prompt% ./pere_para.sh 3

Je suis le fils 1 sur 3
Je suis le fils 2 sur 3
Je suis le fils 3 sur 3
```

Question 2

Nous souhaitons maintenant avoir deux nouveaux scripts `pere.sh` et `fils.sh` avec le même comportement que dans la question précédente mais le père ne doit transmettre **aucun paramètre** à ses fils. Écrivez les scripts `pere.sh` et `fils.sh`.

Exercice 15 – Récupérer un résultat

Notions travaillées :

- **variable `$?`**
- **communication processus père - processus fils**

Question 1

Écrivez un script `affiche_result.sh` qui :

- un paramètre de type entier
- affiche, en fonction des cas :
  - ◇ `Tous est repertoire` si le paramètre est égal à 0.
  - ◇ `1 fichier n est pas un repertoire` si le paramètre est égal à 1
  - ◇ `x fichiers ne sont pas des repertoire` (où `x` correspond à la valeur du paramètre).

Soit le script `cpt_repertoire.sh` suivant :

exercices/valRetour_aff_TP/Scripts/cpt_repertoire.sh

```
#! /bin/bash
# cpt_repertoire.sh

nb=0

for rep in "$@"; do
  if [ ! -d "$rep" ]; then
    nb=$(($nb + 1))
  fi
done
```

Question 2

Modifiez `cpt_repertoire.sh` pour qu'il renvoie son résultat exclusivement une valeur de retour. Écrivez un script `test1.sh` qui appelle le script `cpt_repertoire.sh` modifié puis `affiche_result.sh` en passant en paramètre le nombre calculé par celui-ci.

Question 3

Même question mais en faisant en sorte que `cpt_repertoire.sh` renvoie son résultat exclusivement par un affichage.

Question 4

Même question en utilisant le script `cpt_repertoire.sh` tel qu'il vous a été donné.

Exercice 16 – Valeur de retour d'un processus

Notions travaillées :

- **variable** `$?`
- **communication processus père - processus fils**

Dans cet exercice, vous devez arrêter l'exécution des processus si le nombre de paramètres n'est pas correct.

Question 1

Écrivez un script `petit_fils.sh` qui prend un paramètre et qui affiche le texte`Je suis le processus` suivi de la valeur de ce paramètre. Voici un exemple d'exécution : `Prompt% ./petit_fils.sh 7`

```
Je suis le processus 7
```

Question 2

Le script `alea.sh` (qui vous est fourni) affiche une valeur tirée aléatoirement entre `0` et `9`. Écrivez un script `fils.sh` qui prend un paramètre (son identité). Le script récupère une valeur tirée aléatoirement (en utilisant le script `alea.sh`) et crée un nombre égal à cette valeur de processus `petit_fils.sh`. Avant de créer ses petits-fils, il affiche son identité et le nombre de petits-fils à créer. Chaque processus `petit_fils.sh` reçoit un paramètre composé de son ordre de création concaténé à l'identité de son père. Voici un exemple d'exécution :

```
Prompt% ./fils.sh 3
Je suis le fils 3 et je dois creer 4 processus
```

```
Je suis le processus 3.1
Je suis le processus 3.2
Je suis le processus 3.3
Je suis le processus 3.4
```

Question 3

Écrivez un processus `pere.sh` qui prend un paramètre et qui crée un nombre de processus `fils.sh` égal à la valeur de ce paramètre. Voici un exemple d'exécution :

```
Prompt% ./pere.sh 3
Je suis le fils 1 et je dois creer 2 processus
Je suis le processus 1.1
Je suis le processus 1.2
Je suis le fils 2 et je dois creer 4 processus
Je suis le processus 2.1
Je suis le processus 2.2
Je suis le processus 2.3
Je suis le processus 2.4
Je suis le fils 3 et je dois creer 1 processus
Je suis le processus 3.1
```

Écrivez une nouvelle version des scripts pour que le script `pere.sh` affiche le nombre total de processus créés (le nombre de processus `fils.sh` + le nombre de processus `petit_fils.sh`). Votre solution **ne doit pas utiliser** de fichier. Voici une exécution :

```
Prompt% ./pere.sh 3
Je suis le fils 1 et je dois creer 2 processus
Je suis le processus 1.1
Je suis le processus 1.2
Je suis le fils 2 et je dois creer 4 processus
Je suis le processus 2.1
Je suis le processus 2.2
Je suis le processus 2.3
Je suis le processus 2.4
Je suis le fils 3 et je dois creer 1 processus
Je suis le processus 3.1
10 processus ont ete crees
```

Exercice 17 – Redirection des entrées-sorties

Notions travaillées :

- **ouverture de fichiers**
- **écriture dans un fichier**

Dans cet exercice, lorsqu'un processus doit transmettre une information à son fils, il le fera par l'intermédiaire d'un paramètre.

Soit le script de calcul suivant :

exercices/io_TP/Scripts/calcul.sh

```
#! /bin/bash
# calcul.sh
if [ $# -ne 3 ]
then
```

```
  echo Il faut trois parametres
  exit 1
fi
if [ "$2" != "+" ] && [ "$2" != "-" ]
then
  echo "l'operateur doit etre + ou -"
  exit 2
fi
echo $(($1 $2 $3))
```

Question 1

Écrivez un script `ensemble_calculs.sh` qui utilise le script `calcul.sh` pour exécuter plusieurs calculs. L'exécution

```
Prompt% ./ensemble_calculs.sh "1 + 2" "5 - 7" "8 - 1" "8 - 8"
```

doit produire l'affichage suivant :

```
3
-2
7
0
```

Question 2

Nous souhaitons maintenant écrire une nouvelle version du script `ensemble_calculs.sh`. Les résultats des calculs doivent maintenant être écrits dans un fichier dont le nom sera passé en premier paramètre du script `ensemble_calcul.sh`. Si ce premier paramètre correspond à un répertoire existant, l'exécution doit s'arrêter. S'il correspond à un fichier existant, le contenu de ce dernier doit être remplacé par le résultat des calculs. Voici un exemple d'exécution.

```
Prompt% ls
ensemble_calculs.sh
Prompt% ./ensemble_calculs.sh fic_res "1 + 2" "5 - 7" "8 - 1" "8 - 8"
Prompt% cat fic_res
3
-2
7
0
```

Vous devez proposer deux versions du nouveau script `ensemble_calculs.sh` (`ensemble_calculs1.sh` et `ensemble_calculs2.sh`). Une des deux solutions ne doit provoquer qu'une seule ouverture du fichier. L'autre solution doit provoquer plusieurs ouvertures (une par écriture).

Question 3

Nous souhaitons maintenant modifier les scripts précédents pour que l'écriture des résultats positifs se fasse dans un fichier différent de celui des résultats négatifs ou nuls. Si un des deux fichiers est un sous-répertoire du répertoire courant, l'exécution doit s'arrêter. Si un des fichiers existe déjà, son contenu doit être remplacé par le résultat des calculs. Si `fic_pos` et `fic_neg` ne correspondent pas à un sous-répertoire du répertoire courant, vous devez obtenir l'exécution suivante :

```
Prompt% ls
ensemble_calculs.sh
Prompt% ./ensemble_calculs.sh fic_neg fig_pos "1 + 2" "5 - 7" "8 - 1" "8 - 8"
Prompt% cat fic_neg
```

```
-2
0
Prompt% cat fic_pos
3
7
```

Vous devez proposer deux solutions, une dans laquelle les instructions d'écriture dans les fichiers se trouvent dans le nouveau script `ensemble_calculs.sh` et l'autre dans laquelle les instructions d'écriture dans les fichiers se trouvent dans le nouveau script `calcul.sh`. Dans chacun des cas, vous créerez de nouveaux scripts de manière à ne pas écraser vos réponses aux questions précédentes.

Exercice 18 – Ignorer les signaux

Notions travaillées :

- **signaux**
- **commande** `trap`

Question 1

Écrivez un script `saisie.sh` qui demande la saisie de trois entiers. Le premier entier doit être comprise entre 1 et 31, le deuxième entre 1 et 12 et le troisième entre 1960 et 1980. Si un entier n'est pas dans l'intervalle attendu, la saisie de la valeur est répétée tant que les contraintes ne sont pas vérifiées. Vous devez donc faire 3 boucles, une pour la lecture de chaque entier.

Question 2

Écrivez un nouveau script `saisie1.sh` qui a le même comportement que le script `saisie.sh` mais qui assure en plus qu'un processus qui l'exécute ignore le signal `SIGINT` lors de l'exécution de la deuxième boucle de saisie. Le processus a le comportement par défaut avant ou après cette boucle. Pour tester ceci taper `Ctrl-C`

- lors de la première et la troisième boucle ce qui tuera le processus
- lors de la deuxième boucle ce qui n'aura aucun effet

Exercice 19 – Ping pong avec signaux

Notions travaillées :

- **envoi de signaux**
- **redéfinition de comportement de signaux**

Le but de cet exercice est de coder deux scripts `ping.sh` et `pong.sh` qui afficheront alternativement les messages `ping` et `pong` respectivement. Pour ceci, nous utiliserons les signaux. Ainsi chaque processus enverra alternativement un signal `SIGUSR1`. Á la réception de ce signal, le message sera affiché et le signal sera renvoyé. On pourra faire une pause d'une seconde entre l'affichage du message et le renvoi du signal pour une meilleure lisibilité. Le processus `ping` sera le père du processus `pong`.

Question 1

Écrivez les scripts `ping.sh` et `pong.sh`. Une fois le mécanisme déployé, pour ne pas que les deux processus se terminent, vous ferez une boucle infini contenant une instruction sleep.

Exercice 20 – Processus zombis

Notions travaillées :

- **terminaison d'un processus**
- **état zombi**

Question 1

Écrivez un script creer_zombi.sh qui créé un nombre donné de processus zombis. Assurez-vous du bon fonctionnement de votre script avec la commande `ps u`.

Exercice 21 – Constatation des commutations

Notions travaillées :

- **partage du processeur et de l'affichage**
- **concurrence de processus**

Soit le script `affiche.sh` suivant :

exercices/commutations_TP/Scripts/affiche.sh

```
#! /bin/bash
# affiche.sh
if [ $# -ne 1 ] ; then
  echo Il faut un parametre : un entier
  exit 1
fi
i=0
while [ $i -lt $1 ] ; do
  echo $$ : $i
  i=$(($i + 1))
done
```

Question 1

Quel est l'affichage produit par l'exécution de la commande suivante ?

`Prompt% ./affiche.sh 5`

Question 2

Quel est l'affichage produit par l'exécution de la commande suivante ?

`Prompt% ./affiche.sh 5 & ./affiche.sh 5`

Exécutez la commande plusieurs fois (jusqu'à ce que vous obteniez un affiche ne correspondant pas à une alternance d'affichage entre les deux processus) et expliquez ce que vous obtenez. Qu'en concluez-vous ?

Question 3

Nous considérons maintenant le script `affiche2.sh` suivant :

exercices/commutations_TP/Scripts/affiche2.sh

```
#! /bin/bash
# affiche2.sh
if [ $# -ne 1 ] ; then
  echo Il faut un parametre : un entier
  exit 1
fi
i=0
while [ $i -lt $1 ] ; do
  echo -n $$ :
  echo $i
  i=$(($i + 1))
done
```

Exécutez plusieurs fois la commande suivante (jusqu'à ce que les modifications faites sur le script aient une conséquence sur l'affichage). Qu'en concluez-vous ?

```
Prompt% ./affiche2.sh 5 & ./affiche2.sh 5
```

Exercice 22 – Crible d'Eratosthène en pipeline

Notions travaillées :

- **parallélisation**
- **pipe**
- **récursivité**

Le crible d'Eratosthène est une méthode permettant de calculer les nombres premiers allant de 2 à un entier donné N. Dans sa version séquentielle, le principe est de remplir un tableau composé des entiers naturels trié allant de 2 à N et ensuite de parcourir ce tableau à partir du début. Pour chaque nombre rencontré à la case d'indice i (i variant de 0 à $N-1$), parcourir toutes les cases d'indice j (j allant de $i+1$ à $N-1$) et éliminer toutes les cases indicées par j qui sont divisibles par le nombre indicé par i. Les cases non éliminées à la fin du traitement seront les nombres premiers recherchés.

Il existe une version de ce crible (Crible d'Hoare) mettant en jeu plusieurs processus qui communiquent en pipeline. Chaque processus x ($x \geq 1$) du pipeline est associé au x'ième nombre premier noté $p(x)$. Chaque processus x recevra sur son entrée standard des nombres croissants envoyés depuis le processus $x-1$. Chaque nombre lu sur l'entrée standard qui n'est pas divisible par $p(x)$ (ceci veut dire qu'il est potentiellement premier du point de vue de x) sera transmis au processus successeur $x+1$ via un pipe. Chaque processus du pipeline fait donc office de filtre. Puisque les nombres sont reçus dans un ordre croissant, le premier entier reçu n qui n'est pas divisible par $p(x)$ est donc premier. Dans ce cas, si la borne N n'est pas atteinte, il faut créer le processus successeur $x+1$ avec $n = p(x+1)$. On notera l'existence d'un processus d'indice 0 qui aura le rôle d'amorcer le pipeline en envoyant au processus d'indice 1 ($p(1) = 2$) tous les entiers allant de 2 à N.

Question 1

Écrivez le script `crible.sh` du processus d'indice 0. Ce script prendra en paramètre l'entier N, créera le processus d'indice 1 (script crible_core.sh qui sera demandé dans la question suivante) et lui enverra via un pipe la séquence des entiers allant de 2 à N.

Question 2

Écrivez le script crible_core.sh pour les processus du pipeline.

NB : Pour lancer un script récursivement, vous pouvez utiliser la variable `$0` qui désigne la commande du script courant.

Synchronisation

Avant de commencer cette série d'exercice, vous devez avoir compris les notions de section critique, de verrou et de sémaphore (cf. annexe).

Exercice 23 – Incohérences liées aux commutations et section critique

Notions travaillées :

- **section critique**
- **verrouillage**

Soit le script `ecriture.sh` suivant :

exercices/incoherence_et_section_critique_TP/Scripts/ecriture.sh

```
#! /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
```

Question 1

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

```
Prompt% ./ecriture.sh a b c
```

Question 2

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

exercices/incoherence_et_section_critique_TP/Scripts/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 3

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 4

En utilisant la commande `synchro` qui vous a été fournie, modifiez les script `ecriture.sh` et `lancement_ecriture.sh` 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 parallélisme possible et ne pas bloquer les processus s'ils n'écrivent pas dans le même fichier.

exercices/incoherence_et_section_critique_TP/Scripts/ecriture.sh

```
#! /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
```

Question 5

Ajouter la commande suivante afin qu'elle soit exécutée entre deux sections critiques successives :

```
sleep $(( $(( $(date +%N) % 5)) + 1 ))
```

Que constatez-vous ?

Question 6

Donnez une solution pour assurer que le contenu de chaque fichier soit identique tout en assurant un maximum de parallélisme.

Exercice 24 – Accès concurrents à plusieurs fichiers

Notions travaillées :

- **sections critiques multiples**
- **concurrence et parallélisme**

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.

exercices/concurrence_fic_mult_TP/Scripts/creation_utilisateur.sh

```
#! /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 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 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 25 – Problème de la piscine

Notions travaillées :

- **concurrence**
- **interblocage**
- **sémaphore**

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 libérer la cabine.

Pour suivre la disponibilité des ressources, nous disposons d'un fichier par type de ressource (`nb_cabines` pour les cabines et `nb_paniers`). Chacun des fichiers contient un entier qui correspond au nombre de ressources disponibles.

- Pour prendre une ressource il faut :
  1. `lire le contenu du fichier` (pour savoir combien de ressources sont disponibles)
  2. `attendre éventuellement que la valeur lue soit au moins égale à 1`
  3. `mettre à jour le contenu du fichier` (pour signaler qu'une ressource de moins est disponible)
- Pour libérer une ressource il faut :
  1. `mettre à jour le contenu du fichier` (pour signaler qu'une ressource de plus est disponible)

Question 1

Écrivez le script `prendre_ressource.sh` qui prend en paramètre le nom d'un fichier (celui qui contient le nombre de ressources disponibles) et qui prend une ressource. Vous identifierez la ou les section(s) critique(s) et les protégerez en conséquence.

Question 2

Écrivez le script `rendre_ressource.sh` qui prend en paramètre le nom d'un fichier (celui qui contient le nombre de ressources disponibles) et qui rend une ressource. Vous identifierez la ou les section(s) critique(s) et les protégerez en conséquence.

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 un nombre donnée de 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 $$"
```

Question 3

Écrivez le script `usager.sh` correspondant à l'algorithme précédent (la prise et la libération d'une ressource ne doivent pas créer de nouveaux processus). Ce script prendra en paramètre le nombre de secondes qui correspondra au temps de baignade.

Question 4

Écrivez un script `lancement.sh` qui prend 4 arguments :

1. un nombre initial de paniers (i.e. le contenu initial du fichier `nb_paniers`)
2. un nombre initial de cabines (i.e. le contenu initial du fichier `nb_cabines`)
3. un nombre d'usagers à lancer en parallèle
4. le temps de baignade en seconde d'un usager

et qui :

- créé et initialise les deux fichiers `nb_paniers` et `nb_cabines`
- lance les usagers en parallèle en les paramétrant par le temps de baignade renseigné
- attend la fin des usagers avant de se terminer.

Question 5

Lancer le script avec 5 paniers, 3 cabines, 7 usagers et un temps de baignade de 3 secondes

Question 6

Quelles seraient les conséquences si les sections critiques des scripts `prendre_ressource.sh` et `rendre_ressource.sh` n'étaient pas protégées ? Vous pouvez tester ceci en supprimant les verrous et en exécutant le script `lancement.sh` plusieurs fois , jusqu'à remarquer une incohérence dans le contenu des fichiers `nb_paniers` et `nb_cabines` .

Question 7

Testez à nouveau pour 10 usagers, 5 paniers et 3 cabines. Expliquez pourquoi l'exécution se bloque à nouveau (regardez le contenu des fichiers `nb_cabines` et `nb_paniers`).

Question 8

Proposez une solution pour régler le problème identifié à la question précédente.

Question 9

La solution actuelle ne permet pas d'assurer l'absence de famine, c'est à dire le cas où un usager ne puisse jamais accéder à la piscine. Expliquez pourquoi.

Question 10

Pour résoudre le problème de la question précédente, modifiez les scripts précédents en utilisant la commande synchro en tant que sémaphore.

Exercice 26 – Ping pong avec sémaphore

Notions travaillées :

- **synchronisation avec sémaphores**
- **concurrence**

Le but de cet exercice est de coder deux scripts `ping.sh` et `pong.sh` qui afficheront alternativement les messages `ping` et `pong` respectivement. Pour ceci, nous utiliserons les sémaphores. Ainsi chaque processus devra se synchroniser à l'autre pour pouvoir afficher alternativement son message. On pourra faire une pause d'une seconde entre l'affichage des messages pour une meilleure lisibilité. Le processus `ping` sera le père du processus `pong`.

Question 1

Écrivez les scripts `ping.sh` et `pong.sh`.

Exercice 27 – Lecture alternée

Notions travaillées :

- **lecture de fichier**
- **synchronisation avec sémaphore**

Dans cet exercice, nous souhaitons avoir un ensemble de N processus qui lisent à tour de rôle dans un fichier. Les lectures doivent se faire par ligne et chaque ligne doit être lue exactement une seule fois. L'ensemble des processus s'arrête lorsqu'il n'y a plus de ligne à lire dans le fichier. La synchronisation entre processus se fera à l'aide de sémaphore.

Question 1

Écrivez un script `lecteur.sh` qui lit sur son entrée standard (commande `read`) une fois que son prédécesseur ait lu. Pour rappel, la commande `read` renvoie une valeur différente de 0 si on est arrivé à la fin du fichier.

Question 2

Écrivez un script `deploie_lecteurs.sh` qui prend en paramètre un nombre N de lecteurs. Il créera dans un premier temps le fichier de lecture dans le dossier `/tmp` et dans lequel le contenu d'une ligne i ($i \leq 0$) sera l'entier i. Le premier lecteur commencera sa lecture lorsque tous les processus seront déployés. Le processus déployeur attendra que l'ensemble du fichier soit lu et que l'ensemble des lecteurs se terminent correctement pour ensuite détruire le fichier et les éventuels sémaphores.