NOM Prénom :

CSC3102 – Contrôle Final 1
Année 2023 – 2024

Consignes :

- répondez aux questions en écrivant soit dans les cadres fournis au fil du sujet, soit dans les listings des scripts placés en fin d'exercice, aux endroits des marques correspondant aux questions ;
- si vous manquez de place, vous pouvez écrire sur la page 5 **en l'indiquant dans l'encart initial ;**
- les réponses qui ne requièrent pas de code doivent être justifiées ;
- polycopié, notes personnelles manuscrites et exemples de script sont autorisés ;
- la durée est fixée à 1h30 ;
- le barème est donné à titre indicatif.

Mise en service d'une ligne de tramway

L'ouverture d'une ligne de tramway se fait après différentes phases de tests de mise en service.

Conseil : dans cet exercice fil rouge, les questions ne se suivent pas forcément. **N'hésitez pas à passer.**

[3,5 pts] Préambule

1) [1 pt] Donnez **la** ligne de commande qui permet de créer un répertoire nommé Tramway dans votre répertoire de connexion **quel que soit** votre positionnement dans le système de fichiers.

```
mkdir ~/Tramway
```

2) [0,5 pt] Donnez **la** ligne de commande qui vous place dans le répertoire Tramway **quel que soit** votre positionnement dans le système de fichiers.

```
cd ~/Tramway
```

3) [1 pt] Un répertoire Contrôle_Final se trouve dans le répertoire parent du répertoire Tramway. Sachant que vous êtes placés dans le répertoire Tramway, donnez **la** ligne de commande qui déplace tout le **contenu** du répertoire Contrôle_Final vers le répertoire de travail, en utilisant uniquement des chemins relatifs.

```
mv ../Contrôle_Final/* .
```

4) [1 pt] Donnez **la** ligne de commande qui permet de consulter l'ensemble des fichiers ainsi obtenus. L'affichage attendu est celui du cadre qui suit celui de votre réponse. Pour vous aider à trouver la (les) bonne(s) option(s), vous trouverez en annexe page 8 du sujet le manuel (censuré) de la commande en question.

```
ls -R
```

Affichage attendu :

```
.:
Journal  P.sh  recap_activite.sh  service.sh  tramway.sh  V.sh

./Journal:
T91.run
```

[1,5 pts] Tramway

Le script tramway.sh, dont le squelette est fourni en fin d'exercice, simule l'activité d'un tramway.
Ce tramway est identifié par son nom, passé en premier paramètre et stocké dans la variable nommée *tram*.
Le script reçoit aussi le chemin absolu vers le répertoire Journal, passé en deuxième paramètre et stocké dans la variable *tram_dir* (elle deviendra utile à la question **(g)**).
Le tramway roule (c'est-à-dire qu'un processus exécute tramway.sh) pendant une période fixe, ici 3 secondes, matérialisée par une attente grâce à la commande *sleep*.

(a) [1,5 pts] Dans le script tramway.sh, implémentez la procédure de gestion de défaillance : écrivez la portion de code qui permet d'afficher un message et d'arrêter le tramway (quitter le programme) lorsque celui-ci reçoit un signal d'alarme de type USR1.

[5,5 pts] Mise en service préliminaire

L'ouverture de la ligne T91 est imminente, il est donc temps de faire les premiers tests de mise en service. Vous verrez qu'ils sont très préliminaires !

Dans le script service.sh qui orchestre les tests de mise en service,

(b) [0,5pt] Pour commencer, initialisez la variable *tram* avec la chaîne de caractères T91 (le nom de la ligne de tramway devant être mise en service) ; et la variable *tram_dir* avec le chemin absolu du répertoire Journal tel qu'il a été listé précédemment à la question **4)**.

Le Test 1, déjà écrit dans service.sh, consiste à faire rouler le tramway T91 sans voyageur et sans incident. Ensuite, le Test 1 bis doit faire une mise en situation du Test 1 où le tramway T91 roule toujours sans voyageur, mais subit une défaillance au bout d'une seconde de roulage.

(c) [1,5 pts] Écrivez le Test 1 bis : démarrez un nouveau tramway T91, puis attendez 1 seconde, et enfin simulez une défaillance en lançant un signal d'alarme de type USR1 à ce tramway roulant.

La procédure de gestion de défaillance implémentée à la question **(a)** ayant correctement fonctionné, viennent ensuite des tests de montée en charge. Dans le Test 2, déjà écrit dans service.sh, deux tramways sont successivement lancés sur la ligne.

(d) [1,5 pts] En l'état, le Test 2 engendre-t-il des problèmes de concurrence ? Justifiez votre réponse, si nécessaire par un exemple de scénario problématique du point de vue de la concurrence.
En cas de problèmes de concurrence, protégez les sections critiques identifiées dans les scripts à l'aide des scripts P.sh et V.sh, vus en TP et disponibles dans le répertoire Tramway dans lequel vous travaillez (cf. question **4)**). Faites précéder vos ajouts de la balise (d), précisant ainsi que vous traitez la question **(d)**.

Pas de problème de concurrence : pas de processus concurrents.

À présent, le Test 3, déjà écrit dans service.sh, est un test à pleine charge : l'ensemble des 6 tramways disponibles sur la ligne T91 sont mis en roulage.

(e) [0,5 pt] Avant de passer à la phase de tests suivante, il faut s'assurer que l'ensemble des tramways du Test 3 ont fini de circuler. Mettez en place une barrière d'attente des processus tramways juste avant de finir le Test 3.

(f) [1,5 pts] En l'état, le Test 3 engendre-t-il des problèmes de concurrence ? Justifiez votre réponse, si nécessaire par un exemple de scénario problématique du point de vue de la concurrence.
En cas de problèmes de concurrence, protégez les sections critiques identifiées dans les scripts à l'aide des scripts P.sh et V.sh, vus en TP et disponibles dans le répertoire Tramway dans lequel vous travaillez (cf. question **4)**). Faites précéder vos ajouts de la balise (f), précisant ainsi que vous traitez la question **(f)**. Si la solution mise en place à la question **(d)** gère d'ores-et-déjà les éventuels problèmes de concurrence, précisez-le ici.

Pas de problème de concurrence : il y a des processus concurrents, mais ils ne partagent aucune ressource.

[6,5 pts] Décompte des tramways en activité sur la ligne

À présent, on souhaite tenir un journal de bord de la ligne T91 afin de répertorier l'heure de départ et d'arrivée de chaque tramway de la ligne. Ce journal de bord se trouve dans le sous-répertoire Journal et est nommé T91.run. Il est créé, vide, par le script service.sh avant tout départ de tramway.
Chaque entrée du journal est normalisée selon la forme suivante : *numéro;nom de la ligne;évènement;date* (c'est-à-dire quatre champs séparés par des points-virgules « ; »). La description de chaque champ est la suivante :

- *numéro* : numéro d'entrée dans le journal (commence à 1, est croissant et unique pour chaque entrée)
- *nom de la ligne* : nom de la ligne de tramway sur laquelle survient l'évènement
- *évènement* : vaut exactement « départ » pour l'évènement de départ du tramway, et « arrivée » pour l'évènement d'arrivée du tramway
- *date* : date de l'évènement, vous pouvez utiliser la commande *date* pour l'obtenir

(g) [3 pts] Dans le script tramway.sh, faites ajouter au journal de bord de la ligne de tramway concernée, des entrées indiquant le départ et l'arrivée d'un tramway, telle que décrites ci-dessus. Les ajouts doivent se faire autour de la commande *sleep 3* qui simule le voyage du tramway : une entrée avec « départ » juste avant, et une entrée avec « arrivée » juste après.
Attention à bien respecter la forme des entrées du journal. En particulier, les numéros des entrées doivent se suivre, de manière croissante et monotonique : le numéro d'une entrée est celui de la précédente incrémenté de 1. Vous devez donc au préalable lire depuis le journal le numéro de l'entrée précédente. Pensez à gérer le cas particulier de la première entrée, lorsque le fichier du journal est vide.
De plus, prêtez attention aux éventuels caractères spéciaux présents dans vos messages et au besoin, décomposez votre réponse en plusieurs commandes (plus simples).

(h) [1,5 pts] Les modifications apportées au script tramway.sh remettent-elles en cause votre analyse de problèmes de concurrence après les Test 2 et Test 3 ? Engendrent-elles à présent des problèmes de concurrence ? Justifiez votre réponse, si nécessaire par un exemple de scénario problématique du point de vue de la concurrence.
En cas de problèmes de concurrence, protégez les sections critiques identifiées dans les scripts à l'aide des scripts P.sh et V.sh, vus en TP et disponibles dans le répertoire Tramway dans lequel vous travaillez (cf. question **4)**). Faites précéder vos ajouts de la balise (h), précisant ainsi que vous traitez la question **(h)**. Si la solution mise en place aux questions précédentes qui portaient sur la concurrence gère d'ores et déjà les problèmes de concurrence possibles, précisez-le ici.

Problème de concurrence : le fichier journal est partagé par tous les tramways de la ligne T91.

Exemple de problème de concurrence :
1. tram 1 lit le numéro du dernier évènement ;
2. tram 2 lit le numéro du dernier évènement, puis écrit son propre évènement en se basant sur ce numéro
3. tram 1 veut écrire son évènement, mais il va se baser sur un numéro de dernier évènement erroné.

Il faut établir une section critique (pour implémenter une exclusion mutuelle) autour de la lecture et de l'écriture dans le journal.

Le script recap_activite.sh est en charge d'exploiter le journal de bord d'une ligne. Il reçoit les mêmes paramètres que tramway.sh : le nom de la ligne, et le chemin absolu vers le dossier Journal.
Ce script a deux effets. Le premier est d'afficher sur sa sortie standard, l'un des trois messages suivants en fonction de l'état d'activité des tramways de la ligne (avec « <tram> » remplacé par le nom de la ligne de tramway concerné) :

- Tramway <tram> - aucune rame roulante
- Tramway <tram> - une rame roulante
- Tramway <tram> - XX rames roulantes

Le second effet, est de terminer avec un code de retour reflétant l'état d'activité des tramways :

- s'il n'y a plus aucune rame roulante : 0 ;
- s'il y a exactement une rame roulante : 1 ;
- s'il y a deux rames roulantes ou plus : 2.

Dans le script recap_activite.sh,

(i) [1 pt] En premier lieu, déterminez le nombre de départs ayant eu lieu sur la ligne. Pour cela, vous pouvez compter le nombre de fois où apparaît le motif « *départ* » dans le fichier journal de bord de la ligne concernée. Faites de même pour le nombre d'arrivées.

(j) [0,5 pt] Conservez dans une variable nommée *nb_roulants* le nombre de tramways encore en train de rouler, en faisant la différence entre le nombre de départs et de le nombre d'arrivées sur la ligne concernée.

(k) [0,5 pt] En fonction de la valeur de la variable *nb_roulants*, produisez l'affichage de sortie attendu, ainsi que le code de retour attendu.

[1,5 pts] Retour à la mise en service consolidée

Il faut maintenant vérifier, grâce au script recap_activite.sh, que le nombre de tramways circulants est cohérent avec les tests menés. Au moment des tests dits consolidés, seul le tramway défaillant du Test 1 bis est considéré comme encore roulant, car il n'a pas atteint la fin de sa période de roulage, son arrivée n'a donc pas été enregistrée dans le journal de bord de la ligne T91.

Dans le script service.sh,

(l) [1 pt] Exécutez le script recap_activite.sh et conservez le code de retour de l'exécution dans une variable. Si le résultat est cohérent, c'est-à-dire qu'il traduit bien le fait qu'il reste exactement un tramway roulant, alors produisez l'affichage « Nombre de rames roulantes OK », sinon produisez l'affichage « Nombre de rames roulantes NOK. »

(m) [0,5 pt] Achevez le script en supprimant le journal de bord de la ligne T91.

[1,5 pts] Mise en production

Nous passons à présent à l'étape de mise en production.

(n) [1 pt] Donnez la commande qui permet d'ajouter à l'utilisateur propriétaire le droit d'exécuter sur l'ensemble des scripts (fichiers .sh) de votre répertoire de travail.

```
chmod u+x *.sh
```

(o) [0,5 pt] Donnez la commande qui lance le script service.sh.

```
./service.sh
```

service.sh

```
#! /bin/bash

#(b)
tram="T91"
tram_dir="$HOME/Tramway/Journal"

touch "$tram_dir/$tram.run"


echo "*** Tests préliminaires ***"

echo "Test 1 - une rame"

./tramway.sh.cor "$tram" "$tram_dir"

echo "FIN Test 1"

echo "Test 1 bis - une rame
défaillante"

#(c)
./tramway.sh.cor "$tram" "$tram_dir" &
sleep 1
kill -USR1 $!

echo "FIN Test 1 bis"

echo "Test 2 - deux tramways"

nb_trams=2
while [ $nb_trams -ne 0 ]; do
    ./tramway.sh.cor "$tram"
"$tram_dir"
    nb_trams=$(expr $nb_trams - 1)
done

echo "FIN Test 2"
```

service.sh (suite)

```
echo "Test 3 - tous les tramways"

nb_trams=6
while [ $nb_trams -ne 0 ]; do
    ./tramway.sh.cor "$tram"
"$tram_dir" &
    nb_trams=$(expr $nb_trams - 1)
done

#(e)
wait

echo "FIN Test 3"

echo "*** Tests consolidés ***"

echo "Test bilan"

#(l)
./recap_activite.sh.cor "$tram"
"$tram_dir"
recap=$?

if [ "$recap" -eq 1 ]; then
    echo "Nombre de rames roulantes
OK"
else
    echo "Nombre de rames roulantes
NOK"
fi

echo "Réinitialisation"

#(m)
rm "$tram_dir/$tram.run"

echo "FIN Test bilan"

echo "FIN Tests"
```

tramway.sh

```
#! /bin/bash

if [ $# -ne 2 ] ; then
    echo "tramway - appel incorrect"
    exit 2
fi >&2

tram="$1"
tram_dir="$2"

#(a)
trap "echo \"Alarme déclenchée\"; exit
1" USR1

#(g.1)
./P.sh "$tram".run.lock     #(h)

prec="$(tail -n1 "$tram_dir/$tram.run"
| cut -d";" -f1)"
if [ -z "$prec" ]; then
  prec=0
fi
echo "$(expr $prec + 1);$tram;départ;$
(date)" >> "$tram_dir/$tram.run"

./V.sh "$tram".run.lock     #(h)

sleep 3

#(g.2)
./P.sh "$tram".run.lock     #(h)

prec="$(tail -n1 "$tram_dir/$tram.run"
| cut -d";" -f1)"
echo "$(expr $prec + 1);$tram;arrivée;$
(date)" >> "$tram_dir/$tram.run"

./V.sh "$tram".run.lock     #(h)
```

recap_activite.sh

```
#! /bin/bash

if [ $# -ne 2 ]; then
    echo "recap - appel incorrect"
    exit 254
fi >&2

tram="$1"
tram_dir="$2"

#(i)
nb_departs=$(grep "départ"
"$tram_dir/$tram.run" | wc -l)
nb_arrivees=$(grep "arrivée"
"$tram_dir/$tram.run" | wc -l)

#(j)
nb_roulants=$(expr $nb_departs -
$nb_arrivees)

#(k)
case $nb_roulants in
    0)
        echo "Tramway $1 - aucune
rame roulante"
        exit 0;;
    1)
        echo "Tramway $1 - une rame
roulante"
        exit 1;;
    *)
        echo "Tramway $1 -
$nb_roulants rames roulantes"
        exit 2
esac
```

Annexe : page de manuel de la commande ??? (censurée, extraits choisis)

NOM

??? - Afficher le contenu de répertoires

SYNOPSIS

??? [*OPTION*]... [*FICHIER*]...

DESCRIPTION

Afficher les informations des *FICHIER*s (du répertoire courant par défaut). Les entrées sont triées alphabétiquement si aucune des options -tSUX n'est indiquée.

-a, --all
inclure les entrées débutant par « . »
-A, --almost-all
omettre les fichiers « . » et « .. »
-d, --directory
afficher le nom des répertoires eux-mêmes, pas leur contenu
-h, --human-readable
avec -l ou -s, afficher les tailles en format lisible (par exemple 1K, 234M, 2G, etc.)
-i, --inode
afficher le numéro d'index de chaque fichier
-l
utiliser un format d'affichage long
-r, --reverse
inverser l'ordre du tri
-R, --recursive
afficher récursivement les sous-répertoires
-s, --size
afficher la taille allouée de chaque fichier en nombre de blocs
-S
trier selon la taille des fichiers en commençant par le plus gros
-t
trier selon la date de modification, de la plus récente à la plus ancienne
-U
ne pas trier, afficher selon l'ordre original des entrées du répertoire
-X
trier alphabétiquement selon l'extension des entrées
--help
afficher l'aide-mémoire et quitter.
--version
afficher les informations de version et quitter.