NOM Prénom :

CSC3102 – Contrôle Final 1
Année 2024 – 2025

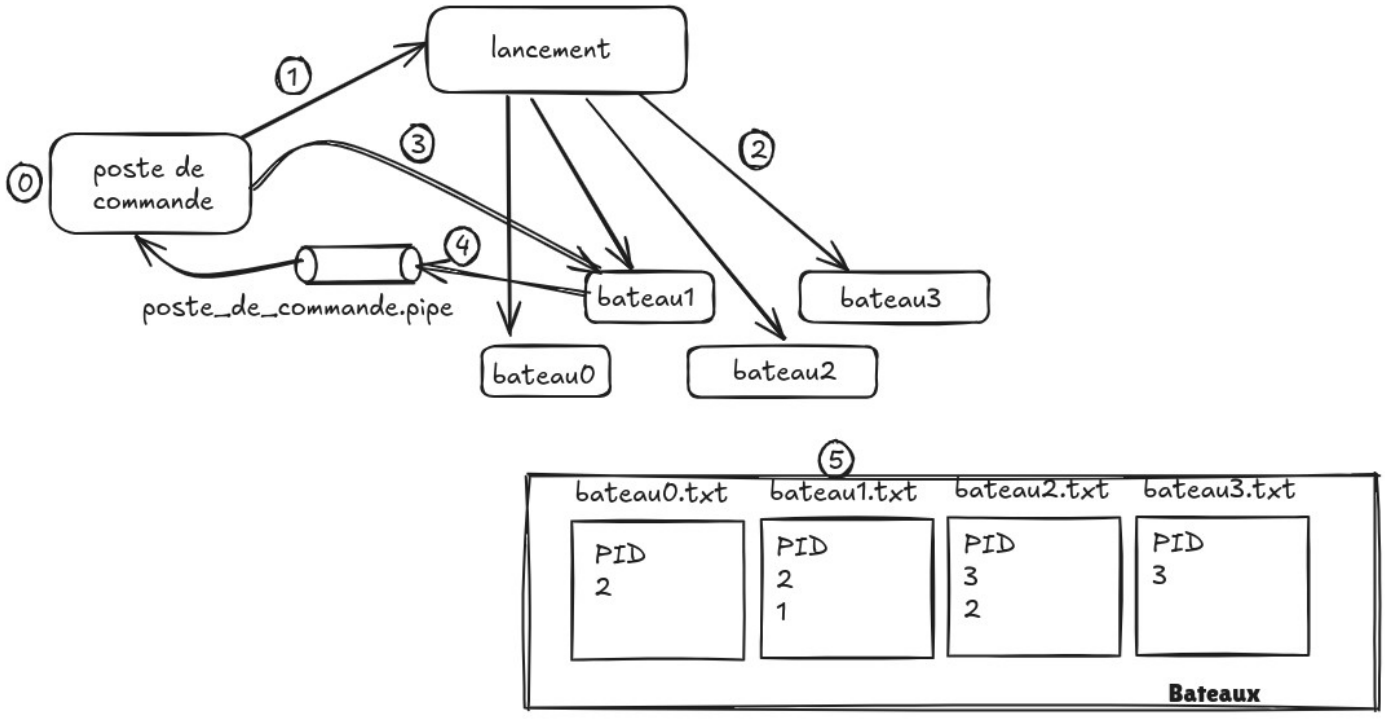
Consignes :

- 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 8 **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.

Bataille navale

Nous proposons de mettre en œuvre une version rudimentaire du jeu de la bataille navale.

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



Cette figure préambule décrit le mode de fonctionnement de la bataille navale attendue.
Le point de démarrage est (0) le script poste_de_commande.sh qui commence par (1) déployer la flotte navale grâce au script lancement.sh. Celui-ci (2) lance les bateaux simulés par le script bateau.sh.
Un bateau est modélisé par un fichier bateau<ID>.txt placé dans le répertoire Bateaux. Une fois déployé, celui-ci navigue jusqu'à (3) recevoir une invitation à tirer. À réception, le bateau (4) émet un message au poste de commande indiquant son ordre de tir sur un bateau. Puis le poste de commande (5) le répercute sur le bateau visé.
Dans un tour de jeu (*i.e.* un tour de boucle du poste de commande), les bateaux en navigation sont tour à tour invités à tirer. À son issue, l'état de la flotte est réévaluée et le poste de commande entame un nouveau tour de jeu. La fin de partie n'est ici pas traitée.

[2 pts] Préambule

1) [0,5 pt] Un répertoire Bataille_Navale a été préalablement créé dans l'arborescence de votre compte utilisateur. Donnez **la** commande qui permet de trouver l'emplacement de ce répertoire.

```
find ~ -type d -name « Bataille_Navale »
```

2) [0,5 pt] En admettant que le chemin identifié à la question précédente est stocké dans une variable nommée chemin_bataille_navale, donnez **la** commande qui permet de vous placer dans ce répertoire.

```
cd $chemin_bataille_navale
```

3) [0,5pt] Donnez **la** commande qui permet de créer un répertoire nommé bateaux dans le répertoire courant.

```
mkdir bateaux
```

4) [0,5pt] Donnez **la** commande qui permet de renommer le répertoire bateaux en Bateaux.

```
mv bateaux Bateaux
```

[1,5 pts] Bateau

Le script bateau.sh, dont vous trouverez le squelette en fin d'exercice, simule la création et la navigation d'un bateau. Le processus associé au bateau prendra fin uniquement lorsque le bateau sera coulé.

Afin de le rendre accessible à d'autres (en particulier au poste de commande), les informations relatives à l'état d'un bateau sont stockées dans un fichier lui étant propre nommé bateau<numero>.txt du répertoire Bateaux. La première ligne correspondra au PID du processus simulant le bateau ; les suivantes, au décompte des différentes touches subies par le bateau.

(5) [1pt] Afin de créer le fichier relatif au bateau à créer, il faut connaître combien de bateaux sont d'ores et déjà en train de naviguer. Pour cela, dans le script bateau.sh, ajoutez les lignes de commandes qui permettent de compter le nombre de fichiers bateau<numero>.txt présent dans répertoire Bateaux en utilisant le motif adéquat. Pour cela, vous pouvez lister les fichiers adéquats avant de les compter. Attention, s'il n'y a aucun fichier bateau<id>.txt, vous ne pourrez pas les lister. Pour éviter cela, commencez par tester l'existence du fichier Bateaux/bateau0.txt et le cas échéant, numérotez le nouveau bateau en création 0.

(6)[0,5pt] Créez le fichier avec le nom correctement forgé avec la bonne numérotation trouvée à la question précédente, écrivez sur la première ligne le PID du processus courant et sur la suivante, la longueur du bateau passée en premier argument du script.

Le bateau entre alors dans sa phase de navigation avec la boucle infinie fournie en fin de script.

[4,5 pts] Lancement

Le script lancement.sh, fourni en fin d'exercice, sera appelé par le poste de commande afin d'initialiser la partie de bataille navale. Après avoir procédé à du nettoyage, ce script est en charge de créer une flotte de bateaux décrite par une variable nommée longueurs_bateaux.

(7) [1pt] Commencez par détruire tous les fichiers du répertoire Bateaux correspondant aux bateaux potentiellement créés à une exécution précédente ; puis tuez tous les processus potentiellement résiduels d'une exécution précédente ; à savoir les processus s'appelant poste_de_commande.sh, lancement.sh, bateau.sh et traitement_tir.sh.

(8) [1pt] Écrire la boucle qui permet de lancer les bateaux dont les longueurs sont données dans la variable longueurs_bateaux. Pour cela itérer sur chaque mot de cette variable et faites appel au script bateau.sh en lui passant en argument la longueur couramment considérée.

(9) [0,5pt] Comme vu à la partie précédente, une fois passé la phase d'initialisation, un bateau simule sa navigation grâce à une boucle infinie. Le retour au poste de commande pour le lancement du bateau suivant est donc vain. Afin de constituer une flotte de bateaux navigant en même temps, il est donc nécessaire de lancer les bateaux en parallèle. Modifiez la boucle de création des bateaux en conséquence.

(10) [2pts] Les bateaux étant à présent lancés en parallèle, cela pose-t-il des problèmes de concurrence ? Votre réponse doit prendre en considération les lancement.sh et bateau.sh. Justifiez votre réponse (dans tous les cas), si nécessaire par un exemple de scénario problématique.
Si oui, 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 Bataille_Navale. Faites précéder vos ajouts de la balise **(10)**, précisant ainsi que vous traitez la question **(10)**.

Il y a de la concurrence sur le numéro à attribuer au bateau à lancer lors du décompte des fichiers nommés Bateaux/bateau*.txt. Si deux processus bateaux.sh sont lancés en parallèle, un premier peut compter 3 bateaux « en navigation », commuter au profit du second, qui compte lui aussi 3 bateaux en navigation. Les deux processus vont donc tous les deux s'attribuer le numéro de bateau 4 et écrire dans le même fichier Bateaux/bateau4.sh leur information.
La section critique s'étend de la consultation, et jusque qu'à la création du fichier relatif au nouveau bateau.

Le script de lancement s'achève lorsque tous les bateaux sont entrés en navigation. Le code de la détection de terminaison de la phase de création des bateaux est donné en fin de script. Il consiste à attendre qu'il y ait autant de fichiers Bateaux/bateau<id>.txt que de bateaux à créer. Ce code utilise une variable nb_bateaux qui sera ajoutée dans la code dans une question ultérieure.

[2,5 pts] Poste de commande

Dans le script poste_de_commande.sh, après avoir lancé la flotte, le poste de commande a pour rôle d'inviter les bateaux à donner leur ordre de tir à tour de rôle, de récupérer cet ordre et de le traiter en le répercutant sur le bateau visé. Dans cette version rudimentaire de la bataille navale, la détection de fin de partie n'est pas traitée. C'est pour cela qu'il n'est pas prévu en l'état de sortie de la boucle infinie qui constitue le corps du script.

(11) [0,5pt] Dans la boucle infinie du script poste_de_commande.sh, écrire une boucle qui itère sur les différents bateaux en navigation, *i.e.* les fichiers Bateaux/bateau<id>.txt existants. Remarque : cette boucle englobe les réponses des futures questions 12 et 13 et un morceau de la 21 (l'incrémentation de la variable cpt). Elle se termine au tag "# fin Q11".

(12) [0,5pt] Toujours dans la boucle de la question (11), récupérez dans une variable nommé pid le PID du processus correspondant au bateau<id> en lisant la première ligne du fichier Bateaux/bateau<id>.txt.

(13) [1pt] Et encore et toujours dans la boucle de la question (11), notifiez le bateau que le poste de commande attend son ordre de tir, grâce à un signal USR1.

À réception de ce signal, le bateau concerné doit choisir sur quel bateau tirer, puis informer le poste de commande via le tube nommé ouvert par ce dernier.

(14)[0,5pt] Dans poste_de_commande.sh mais avant la boucle de la question (11), créez un tube nommé poste_de_commande.pipe que vous maintenez ouvert en lecture/écriture grâce à une redirection avancée qui permettra de récupérer l'ordre de tir du bateau ultérieurement.

[3 pts] Ordre de tir

Dans cette version préliminaire de la bataille navale, l'ordre de tir d'un bateau consiste en un nombre tiré au sort dans un intervalle compris entre 0 et 2 fois la taille de la flotte initiale.

(15) [1pt] Dans lancement.sh, comme il y a autant de bateaux que de longueurs dans la chaîne longueurs_bateaux, stockez dans une variable nb_bateaux le nombre de mots qu'il y a dans la chaînes longueurs_bateaux.

(16)[1pt] Il s'agit à présent de transmettre la taille de la flotte lancée à chacun des bateaux la constituant.
Les bateaux étant lancés après le calcul de cette variable dans lancement.sh, faites-en sorte de l'ajouter à l'environnement du processus lancement.sh transmis aux processus enfant et ainsi la rendre disponible aux processus bateau.sh.

(17) [1pt] Dans le script bateau.sh, ajoutez le traitement de réception du signal envoyé par le poste de commande à la question **(13)**. Ce traitement consiste à transmettre au poste de commande via le tube nommé poste_de_commande.pipe le numéro du bateau sur lequel le bateau tire.
Le numéro du bateau ciblé est tiré au sort entre 0 et 2 fois la taille de la flotte initiale. Pour cela, vous pouvez utiliser la commande suivante :

expr $(rand) % $nb_espace_tir
avec espace_tir=$(expr 2 * $nb_bateaux) # déjà dans le script.

[1,5 pts] Traitement de tir

La dernière étape de la boucle infinie du poste de commande consiste à traiter le tir émis par le bateau courant. Ce traitement est délégué au script traitement_tir.sh.

Dans le script traitement_tir.sh,
(18) [0,5 pt] Commencez par lire le tir émis sur le tube poste_de_commande.pipe par le bateau dans une variable nommée no_bateau_touche.

(19) [0,5 pt] Si le bateau visé n'a pas déjà été coulé (ce qui est le cas s'il n'y a plus de fichier associé dans le répertoire Bateaux), comme chaque tir a pour effet de couler une portion du bateau, il s'agit de décrémenter de un la longueur de bateau restant à flot. Ainsi, récupérez le nombre de touches restant à faire pour couler le bateau, noté en dernière ligne du fichier Bateaux/bateau<no_bateau_touche>.txt dans une variable nb_parties_a_flot, décrémentez-la et écrivez-la en fin du fichier Bateaux/bateau<no_bateau_touche >.txt.

(20) [0,5 pt] Si la variable nb_parties_a_flot est nulle alors le bateau est coulé. Dans ce cas, récupérez le PID du processus correspondant au bateau coulé (en première ligne du fichier associé), supprimez le fichier et tuez le processus au PID récupéré.

[4pts] Multi-tirs

La boucle infinie du poste de commande consiste en une séquence d'invitation au tir/tir/répercutions du tir. Dans notre scénario, comme nous ne comptons pas les points (et en admettant que nous ne prévoyons pas de le faire), cette boucle doit assurer le fait que tous les bateaux tirent le même nombre de fois et que tous les tirs soient traités, sans que l'ordre de leur traitement soient impactant. Ainsi, nous proposons dans cette partie d'inviter tous les bateaux en navigation à tirer, puis de traiter en parallèle tous les tirs émis.

(21) [1 pt] Mettez fin à la boucle mise en place à la question **(11)** juste après avoir invité un bateau à tirer (question **(13))** afin de clôturer la phase d'invitations au tir, puis ajoutez une nouvelle boucle dans laquelle est déporté l'appel à traitement_tir.sh.
Cette nouvelle boucle devant faire autant de tours qu'il y a eu de tirs, utilisez la variable cpt, comptant les tirs, restées jusqu'à présent inutile, à cette fin. Prenez également soin de lancer le script traitement_tir.sh en arrière plan.

(22) [2 pts] Le fait de traiter les ordres de tirs en parallèle engendre-t-il des problèmes de concurrence ? Justifiez votre réponse(dans tous les cas), si nécessaire par un exemple de scénario problématique.
Si oui, 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 Bataille_Navale. Prenez soin de faire le verrouillage le plus fin possible. Faites précéder vos ajouts de la balise **(22)**, précisant ainsi que vous traitez la question **(22)**.

Il y a concurrence quand deux bateaux tirent sur le même et qu'il faut lui décrémenter sa longueur dans le fichier Bateaux/bateau<visé>.txt. L'un peut récupérer la longueur encore à flot du bateau visé, commuter ; puis le second récupère la même longueur, décrémente, écrit dans le fichier la nouvelle longueur ; le premier reprend son exécution et décrémente la valeur initiale avant d'être cette valeur erronée dans le fichier. Le premier tir traité est donc perdu.
La section critique commence au moment dès que le fichier est consulté pour récupération de la longueur à flot et jusqu'à l'écriture de la nouvelle longueur dans le fichier.

(23) [1 pt] Après la boucle de traitement des tirs, faites en sorte que la boucle "while true" qui englobe tout l'algorithme ne recommence pas un nouveau tour avant que tous les processus traitement_tir soient terminés.

[1 pt] Mise en production
Nous passons à présent à l'étape de mise en production.

(24) [0,5 pt] Donnez la commande qui permet d'ajouter à l'utilisateur propriétaire le droit d'exécution sur l'ensemble des scripts (tous les fichiers se terminant par .sh) de votre répertoire de travail.

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

(25) [0,5 pt] Donnez la commande qui lance le script poste_de_commande.sh.

```
./poste_de_commande.sh
```

bateau.sh

```
#! /bin/bash

# Q17
espace_tir=$(expr 2 \* $nb_bateaux)
trap 'expr $(rand) % $nb_espace_tir >
poste_de_commande.pipe' USR1
# fin Q17

### Phase de création
echo Lancement du bateau $$ de
longueur $1

# Q10 (1/2)
./P.sh Bateaux.lock

# Q5
if [ ! -f Bateaux/bateau0.txt ]; then
    num=0
else
    num=$(ls Bateaux/bateau*.txt | wc
-l)
fi
touch Bateaux/bateau${num}.txt
# fin Q5

# Q10 (2/2)
./V.sh Bateaux.lock

# Q6
echo $$ > Bateaux/bateau${num}.txt
echo $1 >> Bateaux/bateau${num}.txt
# fin Q6



### Phase de navigation
while true ; do
    sleep 1
done
```

lancement.sh

```
#! /bin/bash

### Nettoyage d'exécution précédentes
# Q7
rm Bateaux/bateau*.txt; rm *.pipe
killall poste_de_commande.sh
lancement.sh bateau.sh
traitement_tir.sh
# fin Q7

longueurs_bateaux="2 2 3 3"

# Q15
nb_bateaux=$(echo $longueurs_bateaux |
wc -w)
# fin Q15

# Q16
export nb_bateaux
# fin Q16

### Lancement de la flotte de bateaux
# Q8
for l in $longueurs_bateaux ; do
    # Q9 -> ajout du &
    ./bateau.sh $l &
done
#fin Q8


### attente de la création de tous les
bateaux
while [ ! -f bateau0.txt ]; do
    sleep 1
done

cpt=0
while [ $cpt -ne $nb_bateaux ]; do
    sleep 1
    cpt=$(ls bateau*.txt | wc -l)
done
```

poste_de_commande.sh

```
#! /bin/bash

# Q14
mkfifo poste_de_commande.pipe
exec 3<>poste_de_commande.pipe
# fin Q14

./lancement.sh

while true; do

    # Q11 - la boucle for
    cpt=0
    for b in Bateaux/bateau*.txt; do
     cpt=$(expr $cpt + 1)
     ### invite au tour de tir
     # Q12
     read pid < $b
     # Q13
     kill -USR1 $pid
    done

    # Q21
    while [ $cpt -gt 0 ]; do
     ./traitement_tir.sh &
     cpt=$(expr $cpt - 1)
    done
    # fin Q21

    # Q23
    wait

done

echo FIN
```

traitement_tir.sh

```
#! /bin/bash

# Q18
read no_bateau_touche <
poste_de_commande.pipe

# Q22 (1/2)
./P.sh bateau${no_bateau_touche}.lock

# A l'eau, car bateau déjà coulé
if [ ! -f Bateaux/bateau$
{no_bateau_touche}.txt ]; then
    exit 0
fi

# Q19
nb_parties_a_flot=$(tail -1 bateau$
{no_bateau_touche}.txt)
nb_parties_a_flot=$(expr
$nb_parties_a_flot - 1)
echo $nb_parties_a_flot >> bateau$
{no_bateau_touche}.txt
# fin Q19

if [ $nb_parties_a_flot -eq 0 ]; then
    # Bateau touché coulé
    # Q20
    read pid < bateau$
{no_bateau_touche}.txt
    rm Bateaux/bateau$
{no_bateau_touche}.txt
    kill $pid
    # fin Q20
fi

# Q22 (2/2)
./V.sh bateau${no_bateau_touche}.lock
```