NOM Prénom :

CSC3102 – Contrôle Final 2
Année 2024 – 2025

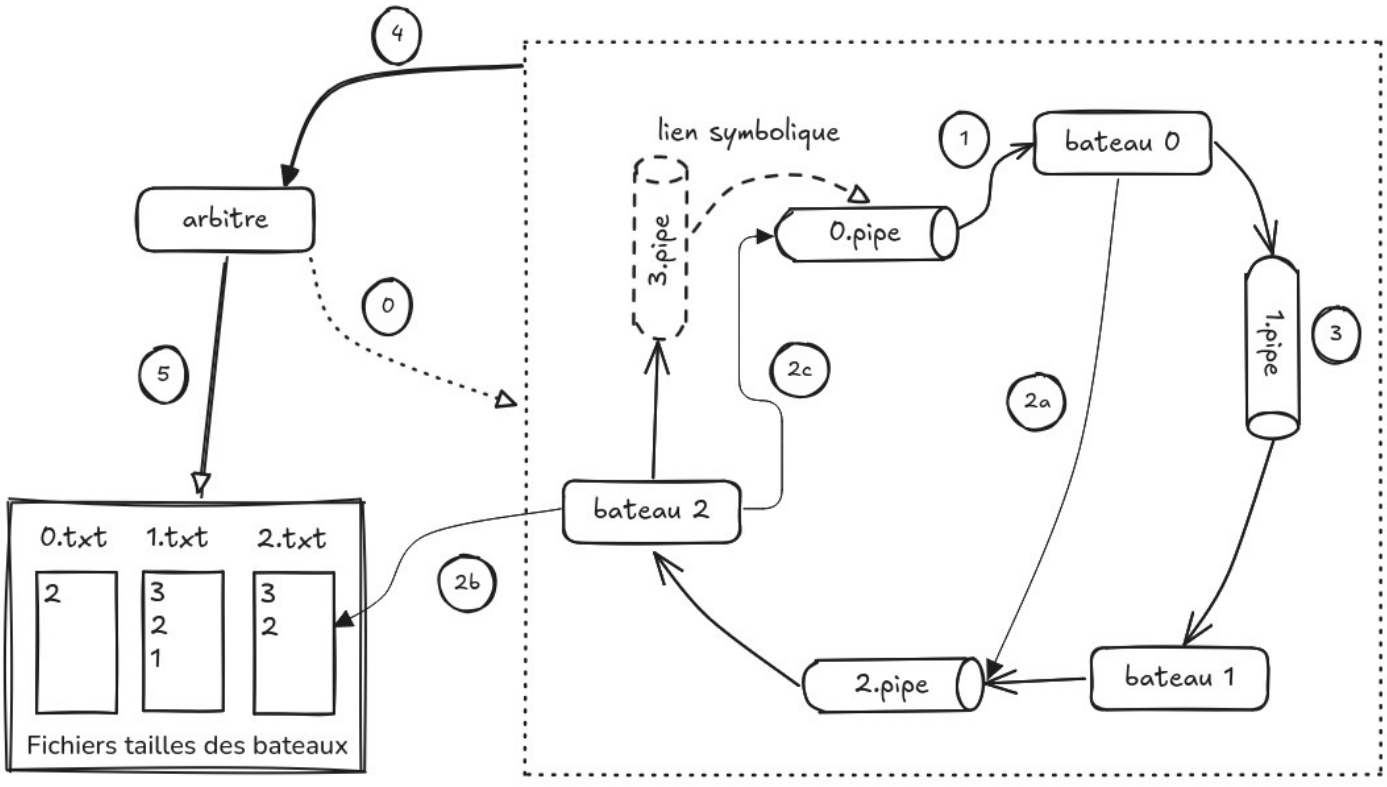
Consignes :

- 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 à l'emplacement initial ;**
- les réponses qui ne requièrent pas de code doivent être justifiées ;
- polycopié, notes personnelles manuscrites et exemples de scripts sont autorisés ;
- la durée est fixée à 1h30 ;
- le barème est donné à titre indicatif.

Bataille navale circulaire

Nous proposons de mettre en œuvre une version rudimentaire du jeu de la bataille navale au tour par tour. Chaque bateau tire, puis transmet l'ordre de tirer au bateau suivant, et ainsi de suite jusqu'à ce qu'il ne reste plus qu'un seul bateau.

Note : dans les énoncés, la syntaxe <XXX> indique un nom variable. Par exemple, le nom de fichier <ID>.txt désigne les fichiers 0.txt, 1.txt, etc.
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.
Au démarrage (0), le script arbitre.sh déploie la flotte navale en lançant plusieurs fois le script bateau.sh qui simule un bateau ; un bateau est modélisé par son processus qui exécute bateau.sh, et par son fichier <ID>.txt contenant sa taille. L'arbitre crée aussi les tubes de communication entre les bateaux. Remarquez que pour maintenir des noms de tubes facilement prévisibles, il établit un lien symbolique portant le nom du tube successeur attendu par le dernier bateau, vers le tube prédécesseur du premier bateau.

Une fois lancés, les bateaux naviguent en attendant un ordre de tirer ou une touche.
Ainsi, un tour de tir commence par (1) un bateau ID qui reçoit l'ordre de tirer de son prédécesseur (ou par l'arbitre pour le premier tour) dans son tube <ID>.pipe.
Pour tirer, un bateau décide d'une cible, puis (2a) indique à sa cible qu'elle est touchée, en envoyant un message de touche dans son tube. Celle-ci (2b) met à jour sa taille dans son fichier <ID>.txt, et (2c) confirme la touche au tireur via le tube de ce dernier. Enfin, un tour de tir se termine en (3) transmettant l'ordre de tirer à son successeur.
Si la taille d'un bateau touché tombe à 0, alors celui-ci se retire du jeu en supprimant son fichier taille <ID>.txt, son tube <ID>.pipe, et en remplaçant ce dernier par un lien symbolique vers le tube du successeur.
La partie se termine lorsqu'un bateau se rend compte qu'il est le dernier en jeu, auquel cas (4) il signale la fin de partie au processus arbitre.
En parallèle des tours de tir, l'arbitre (5) observe régulièrement les tailles des bateaux.

[2,5 pts] Préambule

(1) [0,5 pt] Un répertoire appelé mer, a été créé dans votre répertoire de connexion. Donner **la** commande pour vous déplacer dans ce dossier.

(2) [0,5pt] Donnez **la** commande pour lister l'ensemble des fichiers dans le dossier courant.

(3) [0,5 pt] Vous constatez que le script arbitre.sh se trouve déjà dans ce dossier. Mais il reste aussi des épaves d'une précédente partie. Ce sont des fichiers dont les noms sont des numéros de bateaux, avec les suffixes .txt ou .pipe. Donnez **la** commande pour supprimer toutes les épaves (et uniquement les épaves).

(4) [1pt] Il manque cependant le script bateau.sh. Donnez **la** commande pour établir un lien symbolique nommé bateau.sh dans le dossier courant, vers le script bateau.sh présent dans le dossier parent ; afin que cette installation soit utilisable chez un autre utilisateur, **utilisez uniquement des chemins relatifs**.

[2,5 pts] Bateau : Préparation

Le script bateau.sh, dont vous trouverez le squelette en fin d'exercice, simule la création et la navigation d'un bateau. Il s'agit principalement d'une boucle de lecture du tube d'entrée <ID>.pipe, qui traite les ordres de tirer et les touches.

(5) [1pt] Lors de son lancement par l'arbitre, le bateau reçoit son numéro et sa taille, respectivement dans son premier et son deuxième argument. Conservez son numéro dans une variable id, et sa taille dans une variable taille. Puis écrivez dans le fichier taille <ID>.txt la taille du bateau.

(6) [1pt] Le bateau met ensuite en place son tube d'entrée : le nom est donné par la variable tube_precedent. Créez ce tube nommé, et maintenez-le ouvert en lecture / écriture grâce à une redirection avancée.

Le bateau attend à l'infini de recevoir un ordre de tir ou une touche. Dans les deux cas, il s'agit d'un message envoyé sur le tube nommé créé précédemment. Le message est constitué de deux mots : l'ordre (une chaîne de caractère valant « tirer » ou « touché »), et l'origine, c'est-à-dire le bateau qui le lui a envoyé.

(7) [0,5pt] Au tout début de la boucle infinie, lisez depuis le tube nommé créé précédemment, l'ordre et l'origine de l'ordre, dans des variables nommées respectivement ordre et origine.

[2 pts] Arbitre : début de partie
Laissons un instant le script bateau.sh, pour écrire le script arbitre.sh, dont vous trouverez le squelette en fin d'exercice. Il gère la préparation et le lancement de la partie, ainsi que le nettoyage. La donnée initiale est la liste des tailles des bateaux, dans la variable bateaux.

(8) [1,5pts] L'arbitre commence par lancer tous les bateaux en parallèle. Écrivez la boucle qui itère sur chaque mot de la variable bateaux, et qui exécute à chaque fois le script bateau.sh **en arrière-plan**, en lui donnant en premier argument le numéro du bateau, et en deuxième argument sa taille. Vous devez donc tenir vous-mêmes le compte des numéros des bateaux, dans la variable id_bateau.

Remarque : Chaque bateau de numéro ID s'attend à lire l'ordre de tirer ou la touche dans son tube nommé <ID>.pipe, et à transmettre l'ordre de tir au bateau suivant dans le tube <ID+1>.pipe. Cette chaîne de transmission est cependant cassée entre le dernier bateau et le bateau 0 (voir schéma d'introduction). Notez alors qu'elle est rétablie grâce à un lien symbolique, qui porte le nom de tube attendu par le dernier bateau et qui pointe vers le tube d'entrée du premier bateau.

(9) [0,5pt] Lancez la partie, en écrivant le premier ordre de tir dans le tube du premier bateau. Il s'agit d'un message comportant le mot « tirer », suivi de l'émetteur de l'ordre (ici, vous pouvez indiquer « arbitre »).

[1,5 pts] Bateau : Touché
Revenons maintenant au script bateau.sh, en écrivant d'abord le traitement d'une touche reçue. Le bateau doit mettre à jour sa taille, et répondre au tireur. Mais avant de répondre, s'il est touché-coulé, il doit se retirer du jeu en supprimant son fichier taille <ID>.txt et en faisant en sorte que son bateau prédécesseur envoie désormais ses ordres de tir à son bateau successeur.

(10) [0,5pt] Décrémentez la taille du bateau (variable taille), et mettez à jour celle-ci en ajoutant la ligne au fichier taille <ID>.txt.

(11) [0,5pt] Si la taille du bateau est positive, le bateau n'est que touché. Mais sinon, le bateau est touché-coulé : supprimez son fichier taille ainsi que son tube nommé.

La suppression du tube d'un bateau casse la chaîne de transmission de l'ordre de tir : en effet, le bateau prédécesseur doit désormais envoyer l'ordre au successeur du bateau coulé. Cependant, le bateau prédécesseur n'a pas connaissance de ce changement, il faut donc re-router ses messages.

(12) [0,5pt] En vous inspirant de la remarque de la partie précédente, rétablissez la chaîne de transmission à l'aide d'un lien symbolique : il faut que le bateau prédécesseur du bateau coulé, envoie ses messages au bateau successeur du bateau coulé.

[6,5 pts] Bateau : Ordre de tirer
Écrivons ensuite le traitement d'un ordre de tir. Le bateau doit établir une liste de cibles, choisir une cible, indiquer la touche à sa cible, attendre sa réponse, et enfin transmettre l'ordre de tir au bateau suivant.

Les cibles valides sont les bateaux toujours en lice, ce qui est matérialisé par le fait que leur fichier taille <ID>.txt existe toujours (puisque comme écrit précédemment, un bateau touché-coulé supprime son fichier taille). Il faut cependant retirer de la liste des cibles le bateau tireur (il ne doit pas se tirer dessus) !

(13) [1pt] Conservez dans la variable cibles la liste des cibles valides, c'est-à-dire la liste des fichiers tailles (<ID>.txt) dont les bateaux sont toujours en jeu, à l'exclusion du bateau tireur. Pour ce faire, vous pouvez écrire une chaîne de commandes pour lister puis filtrer les fichiers.
Le bateau cible sera sélectionné dans cette liste, et son numéro extrait plus loin, dans la variable cible. Ne vous souciez donc pas que l'extension .txt apparaisse dans la liste construite.

(14) [1pt] Il est possible que le bateau tireur soit le dernier en lice. Cette situation est identifiable par la variable cibles construite précédemment, qui est vide. Dans ce cas, il doit notifier l'arbitre que la partie est terminée. Envoyez le signal USR1 au processus arbitre, parent du processus bateau tireur.

S'il y a des cibles, il faut compter leur nombre afin de pouvoir ensuite en sélectionner une (ici, au hasard) avant d'extraire son fichier taille de la liste cibles, pour pouvoir la cibler.

(15) [0,5pt] Comptez dans la variable nb_cibles, le nombre de fichiers tailles listés dans la variable cibles.

Ensuite, la variable num_cibles déjà donnée, contient le numéro de la cible dans la liste de la variable cibles.

(16) [1pt] À l'aide des commandes *head* et *tail*, sélectionnez dans la variable fic_cible, le fichier ciblé.

Juste après, le numéro de bateau ciblé a été extrait de son nom de fichier dans la variable cible.

(17) [0,5pt] Écrivez le message de touche (mot « touché » suivi du numéro du bateau tireur) dans le tube de la cible, puis lisez sa réponse depuis le tube du tireur dans la variable reponse.

(18) [0,5pt] Enfin, transmettez l'ordre de tir au bateau suivant, en écrivant dans le tube suivant du tireur (dont le nom est donné par la variable tube_suivant).

(19) [2pts] Maintenant que le script bateau.sh est écrit, existe-t-il des problèmes de concurrence ?
Justifiez votre réponse dans tous les cas, le cas échéant 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 courant. Prenez soin de faire le verrouillage le plus fin possible. Faites précéder vos ajouts de la balise **(19)**, précisant ainsi que vous traitez la question **(19)**.

Le script bateau.sh est à présent terminé, et nous revenons au script arbitre.sh pour traiter la fin de partie.

[2pts] Arbitre : Fin de partie
La partie se termine lorsqu'un bateau observe qu'il est le dernier : il envoie alors le signal USR1 à l'arbitre.

(20) [1pt] À l'endroit approprié dans le script arbitre.sh, écrivez le gestionnaire du signal de fin de partie. Celui-ci doit positionner la variable bataille à « finie » afin de terminer la boucle de suivi de la partie.

(21) [1pt] L'arbitre s'occupe aussi de nettoyer après la partie : avant de quitter, 1. tuez tous les processus bateau.sh, 2. attendez qu'ils soient tous morts, et 3. supprimez enfin les fichiers taille <ID>.txt qu'il reste, ainsi que les tubes nommés <ID>.pipe.

[3pts] Arbitre : Surveillance de la partie

L'arbitre souhaite surveiller la partie, en affichant périodiquement l'état des bateaux encore vivants.

(22) [1pt] Complétez la boucle de suivi de la partie : pour chaque fichier taille restant, affichez son nom, suivi de sa dernière ligne, qui contient donc la taille du bateau correspondant.

(23) [2pts] Pensez-vous que l'ajout de cette surveillance entraîne des problèmes de concurrence ?
Justifiez votre réponse dans tous les cas, le cas échéant 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 courant. Prenez soin de faire le verrouillage le plus fin possible. Faites précéder vos ajouts de la balise **(23)**, précisant ainsi que vous traitez la question **(23)**.

bateau.sh

```
#! /bin/bash

if [ $# -lt 2 ]; then
    echo "Manque arguments"; exit 2
fi >&2

#Q5




tube_precedent=$id.pipe
tube_suivant="$(expr $id + 1)".pipe

#Q6



while true; do

    #Q7


    if [ "$ordre" = "touché" ]; then

        #Q10



        if [ $taille -gt 0 ]; then

            echo "touché !"
                >"$origine".pipe

        else

            #Q11


            #Q12



            echo "touché coulé !"
                >"$origine".pipe

        fi
```

bateau.sh (suite)

```
elif [ "$ordre" = "tirer" ]; then
        echo "$id : Reçu ordre de tirer
par $origine"

        #Q13



        if [ -z "$cibles" ]; then
            echo "$id : Gagné !"

            #Q14


        else

            #Q15



            # +1 pour numéro de ligne
            num_cible=$(expr $RANDOM %
                $nb_cibles + 1)

            #Q16



            cible="$(basename
                "$fic_cible" .txt)"

            echo "$id : Tir sur $cible"

            #Q17



            echo "$id : $cible $reponse"

            #Q18



        fi
    fi

    sleep 1

done
```

arbitre.sh	

```
#! /bin/bash



bateaux="2 3 3 4"




#Q8









while ! [ "$(ls *.txt | wc -l)" -eq
$id_bateau ]; do
    sleep 0.1
done


ln -s 0.pipe $id_bateau.pipe


#Q9



bataille="en cours"
while [ "$bataille" != "finie" ]; do


    sleep 2 &
    wait $!



    #Q22









done


#Q21
```