ISS - Initiation aux Systèmes d'exploitation et au Shell
LU2IN020
Partiel du 14 novembre 2019

Nom : **Prénom :** **Numéro de groupe :**

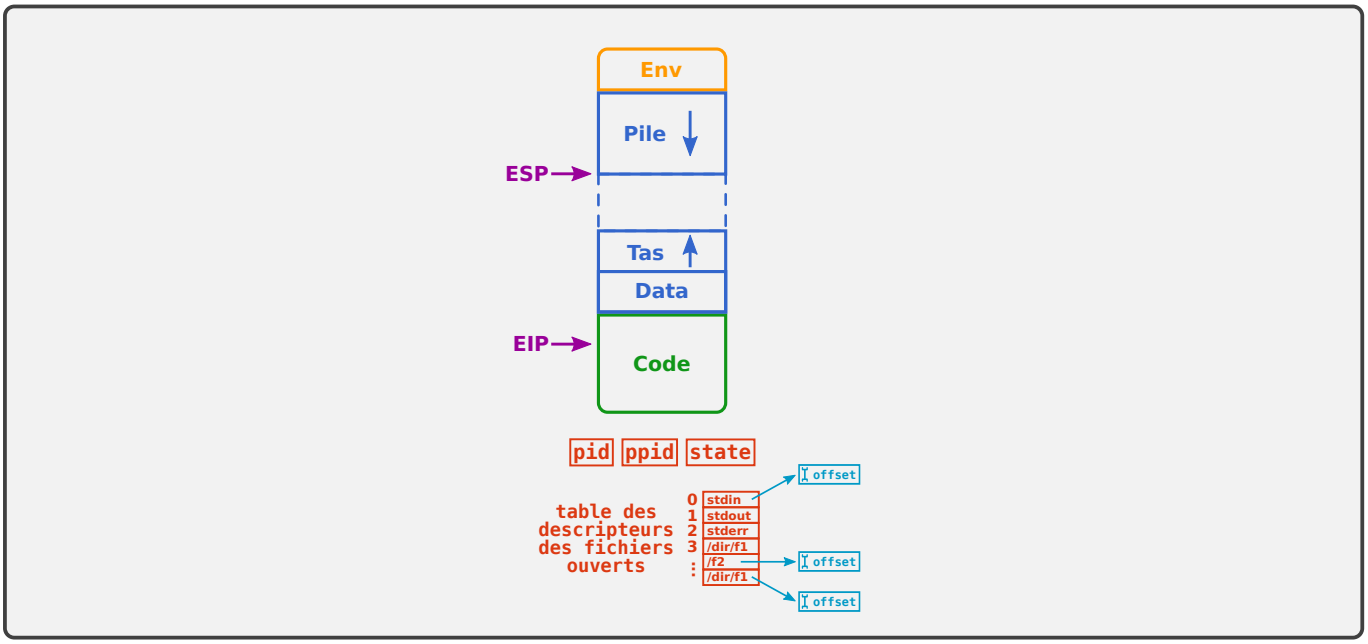
Aucun document autorisé pendant l'épreuve
Les téléphones portables, les montres connectées et autres appareils doivent être rangés dans votre sac.
Le barème n'est donné qu'à titre indicatif, pour vous permettre de juger de la difficulté des questions.
Attention : l'énoncé est imprimé recto-verso sur **7 pages**.

Hypothèse pour l'ensemble de l'examen : Pour simplifier, si les questions n'indiquent pas le contraire, on supposera que tous les exécutables sont bien présents dans le répertoire de l'exercice de et que les droits nécessaires à leurs exécutions sont attribués à l'ensemble des utilisateurs.

Exercice 1 : Questions de cours (6 points)

Question 1 – 1,5 point

Dessinez, tel que présenté au début de chaque cours, l'ensemble de la mémoire d'un processus en indiquant ses différentes zones, les registres spéciaux, ainsi que l'ensemble des données de l'OS permettant de gérer le fonctionnement de ce processus.



Question 2 – 0,5 point

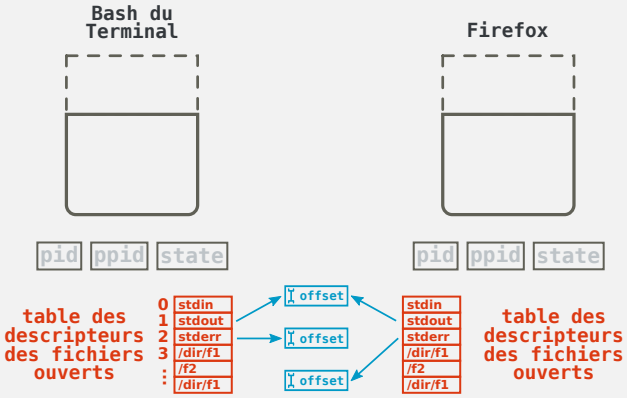
Quelle propriété, étudiée en cours, offre l'espace mémoire dédié aux variables d'environnement ?

C'est le seul espace mémoire utilisateur à ne pas être réinitialisé lors d'un `exec`.

Question 3 – 1 point

En vous concentrant uniquement sur la gestion des fichiers, faites un schéma simplifié de la mémoire correspondant au début de l'exécution de la commande suivante :

```
moi@pc /home/moi $ firefox 2> /tmp/firefox.log
```



Question 4 – 1 point

La deuxième commande de la séquence suivante n'est pas toujours fonctionnelle. Quelle supposition fait celui qui l'a lancée ? Donnez une ligne de commande qu'il aurait pu utiliser précédemment pour assurer cette supposition.

```
moi@pc /home/moi/bin $ gcc unCode.c -o monProg
moi@pc /home/moi/bin $ monProg
```

Comme il n'y a pas de chemin pour `monProg`, l'interpréteur *Bash* va utiliser la variable `$PATH` pour chercher l'exécutable. Cette commande suppose donc que le répertoire courant (`/home/moi/bin`) est bien dans le `$PATH`). C'est le cas si l'utilisateur a exécuté auparavant la commande suivante :

```
moi@pc /home/moi/bin $ PATH="$PATH:/home/moi/bin"
moi@pc /home/moi/bin $ monProg
```

Question 5 – 1 point

Le même programmeur relance maintenant son programme de la façon suivante. Quel problème pose cette nouvelle commande ? Quelle commande aurait-il du lancer ?

```
moi@pc /home/moi $ monProg > /tmp/toto 2> /tmp/toto
```

Avec cette commande, les deux sorties utilisent le même fichier avec des offsets différents (deux redirections impliquent deux ouvertures de fichiers). Les écritures vont donc s'écraser mutuellement. La bonne commande est :

```
moi@pc /home/moi $ monProg > /tmp/toto 2>&1
```

Question 6 – 1 point

Quels sont les trois grands services présentés en cours qu'offre généralement un système d'exploitation ?

Un système d'exploitation offre généralement :
- un système de partage du CPU / système de gestion des processus / ordonnanceur
- un système de gestion de la mémoire
- un système de gestion de fichiers

Exercice 2 : ABC (4 points)

Dans cet exercice, on considère les 3 scripts suivants, tous dans un même répertoire.

A.sh

```
#! /bin/bash

chaine="A"
echo "A-1 : $chaine"
./B.sh
/bin/echo "A-2 : $chaine"
```

B.sh

```
#! /bin/bash

chaine="${chaine}B"
echo "B-1 : $chaine"
./C.sh
echo "B-2 : $chaine"
source ./C.sh
echo "B-3 : $chaine"
export chaine="${chaine}X"
./C.sh
echo "B-4 : $chaine"
```

C.sh

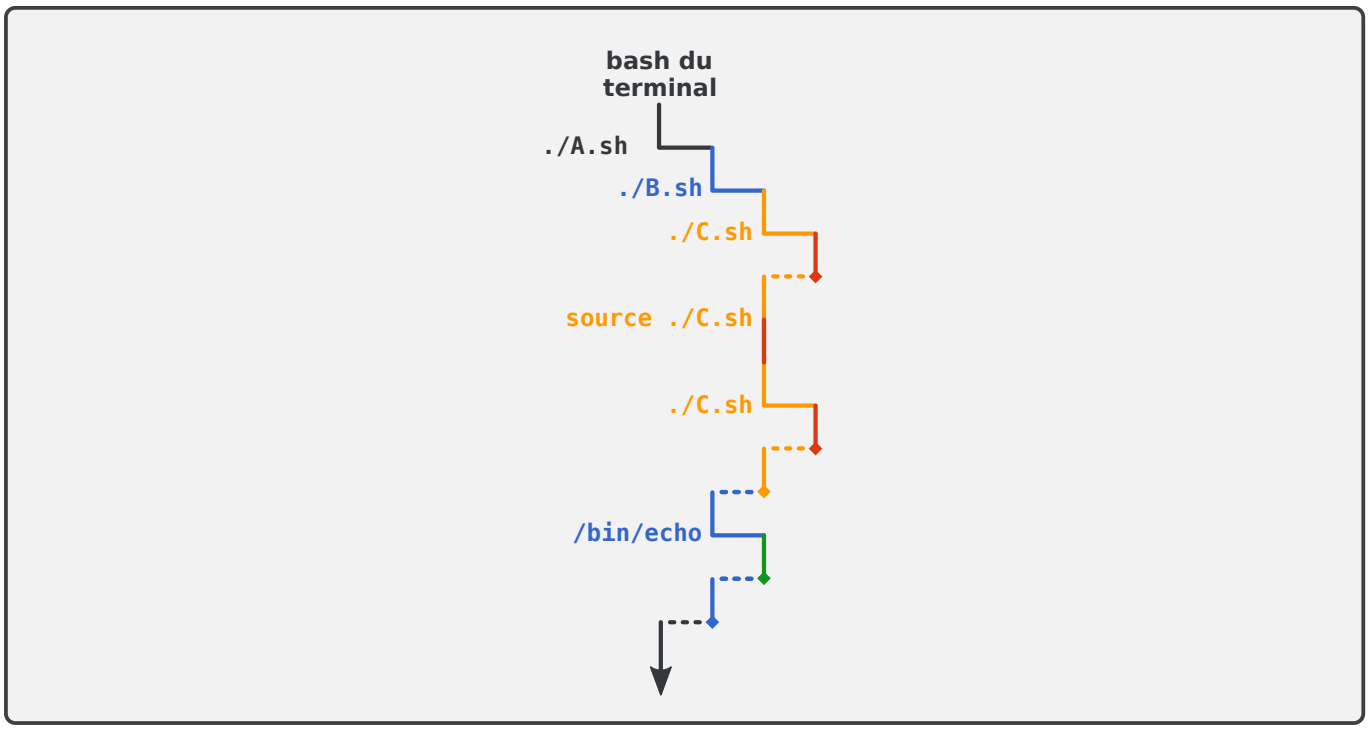
```
#! /bin/bash

chaine="${chaine}C"
echo "C-1 : $chaine"
```

Question 1 – 2 points

Dessinez le chronogramme correspondant à l'exécution de la commande suivante dans un terminal :

```
moi@pc /home/moi $ ./A.sh
```



Question 2 – 2 points

Quel est l'affichage produit par l'exécution du script `A.sh`

```
moi@pc /home/moi $ ./A.sh
A-1 : A
B-1 : B
C-1 : C
B-2 : B
C-1 : BC
B-3 : BC
C-1 : BCXC
B-4 : BCX
A-2 : A
```

Exercice 3 : Chasse au trésor (4 points)

Dans cet exercice on veut implémenter une chasse au trésor. Comme tout trésor qui se respecte, le nôtre sera identifié par un `'X'` sur une carte constituée de 10 lignes, composée de 10 lettres séparées par des virgules. L'absence de trésor est matérialisée par la lettre `'O'`.

Pour simplifier, on supposera ici que tous les paramètres des programmes, ainsi que les valeurs saisies sont corrects. Vous n'avez donc pas besoin de les tester avant de les utiliser.

Question 1 – 2 points

Pour commencer, implémentez un script `pasDeTresorIci.sh` qui teste s'il y a un `'X'` en j^e position sur la i^e ligne (où les deux numérotations commencent à 1) dans un fichier au format *csv*. Les valeurs de i et de j, ainsi que le nom du fichier sont passés en paramètre au script dans cet ordre. Votre script devra donc renvoyer le code de retour 0 s'il n'y a pas de `'X'` à la position indiquée et 1 dans le cas contraire.

pasDeTresorIci.sh

```
#! /bin/bash

line=$1
col=$2
map=$3

test $( head -n $line $map | tail -n 1 | cut -d ',' -f $col ) != 'X'
exit $?
```

Question 2 – 2 points

Utilisez votre script `pasDeTresorIci.sh` (vous pouvez supposer qu'il existe) pour implémenter un script `chasseAuTresor.sh` qui, tant que le trésor n'aura pas été trouvé, demandera à l'utilisateur deux coordonnées : un numéro de ligne et un numéro de colonne. Ce script s'arrête lorsque l'utilisateur aura saisi les coordonnées correspondant au trésor d'une carte, dont le nom de fichier aura été passé en paramètre. Il affichera alors le message suivant : `"Vous avez trouvé le trésor !!!"`,

chasseAuTresor.sh

```
#! /bin/bash

read line col
while ./pasDeTresorIci.sh $line $col $1 ; do
  read line col
done
echo "Vous avez trouve le tresor !!!"
```

Exercice 4 : ADN (6 points + 2 points bonus)

Dans cet exercice, on s'intéresse à une base de données de gènes sous la forme d'un fichier de caractères, où chaque ligne correspond à un gène. Un gène est une succession de bases nucléiques identifiés par une des 4 lettres suivantes `A`, `C`, `G` et `T`. Notons que les gènes n'ont pas tous la même longueur.

Pour l'ensemble de cet exercice, vous pouvez utiliser toutes les commandes étudiées en TD et en TP, à l'exception de la commande `sed`.

Question 1 – 1 point

La base originale `monADN.txt` contient un certain nombre de doublons. Pour accélérer les traitements à venir, on souhaite s'en débarrasser. Donnez la commande permettant de créer une nouvelle base `monADN-v2.txt` conservant un exemplaire de chaque gène de la base originale.

```
moi@pc /home/moi $ sort monADN.txt | uniq > monADN-v2.txt
```

Question 2 – 1,5 point

Donnez une commande qui permet de compter tous les gènes qui ont exactement 2 bases nucléiques `A` et une seule fois la base `T`.

```
moi@pc /home/moi $ grep -E '^[^A]*A[^A]*A[^A]*$' monADN.txt | grep -c -E '^[^T]*T[^T]*$'
```

Question 3 – 1,5 point

Donnez maintenant une commande qui permet d'afficher tous les gènes dans lesquels une séquence exclusivement composée de `G` et de `T` est répétée au moins une fois.

```
moi@pc /home/moi $ grep -E '([TG]+).*\1' monADN.txt
```

Question 4 – 1 point

Dans l'une des étapes de la synthèse des protéines, l'ADN sert de modèle pour fabriquer de l'ARN suivant le schéma suivant : les C deviennent des G et inversement ; les T deviennent des A tandis que les A deviennent des U. Donnez une commande qui fabrique la base `monARN.txt` dont chaque ligne contient l'ARN correspondant aux gènes de la base `monADN.txt`.

```
moi@pc /home/moi $ tr CGTA GCAU < monADN.txt > monARN.txt
```

Question 5 – 1 point

Contrairement aux questions précédentes de cet exercice, dans cette question vous devez utiliser la commande `sed` et uniquement la commande `sed`.

L'ADN est sujet à de nombreuses mutations. Dans cette dernière question, on veut réaliser une nouvelle base de données de gènes mutés suivant la règle suivante : toutes les occurrences des séquences CAT sont permutées avec les trois bases suivantes si elles existent. Ainsi, la séquence $....CATGGC...$ sera mutée en $....GGCCAT.....$

Proposez une commande qui produise un fichier `monADN-mute.txt` contenant une version mutée des gènes contenus dans la base `monADN.txt`. Des points supplémentaires seront attribués si votre fichier `monADN-mute.txt` ne contient que les gènes mutants (*i.e.*, ne contiennent pas les gènes restés identiques).

```
moi@pc /home/moi $ sed -n -E 's/CAT(...)/\1CAT/gp' monADN.txt > monADN-mute.txt
```

Question 6 – 2 points (BONUS)

Dans cette dernière question, vous allez devoir écrire un script. Vous êtes libre d'utiliser tout ce qui a été vu en TD et en TME. Notez toutefois qu'une attention sera portée à la simplicité du résultat lors de la correction. Enfin, sachez que des points pourront être accordés sur des solutions partielles.

Pour finir, on aimerait étudier les extrémités des gènes et plus particulièrement les séquences de trois bases que l'on retrouve à l'identique au début et à la fin d'un gène, *e.g.* dans le gène $GCT....GCT$. Vous allez donc implémenter un script qui affiche la séquence de 3 bases qui vérifie cette propriété sur le plus de gènes.

```bash
#! /bin/bash

baseADN=$1

nb_max=0
seq_max=""

for seq in {A,T,G,C}{A,T,G,C}{A,T,G,C} ; do
  nb=$(grep -c -E "^$seq.*$seq\$" $baseADN)
  if [ $nb -gt $nb_max ] ; then
    nb_max=$nb
    seq_max=$seq
  fi
done

echo $seq_max
```

```bash
#! /bin/bash

baseADN=$1

sed -n -E 's/^(...).*\1$/\1/p' $baseADN | sort | uniq -c | sort -n | tail -1 | \
     tr -s ' ' ,  | cut -d , -f 3
```