# Système et programmation système

Examen 1 – 1h30

Mardi 4 avril 2023

**Tout document est interdit. Tout appareil électronique, incluant téléphone portable, est interdit.**

## Exercice 1 : Questions diverses

**Question 1.1** Pourquoi utilise-t-on `dash` plutôt que `bash` pour interpréter les scripts shell ?

**Question 1.2** En langage C, que trouve-t-on dans un fichier d'entête d'une unité de compilation ?

**Question 1.3** Citer 3 commandes pour lesquelles les shells `dash` et `bash` possèdent une version interne, et pour lesquelles il existe également une version externe. Par défaut, quelle version de la commande est exécutée ? Comment exécuter l'autre version ?

**Question 1.4** `killall COMMANDE_CIBLE`

- Que fait cette commande, et comment le fait-elle ? Que peut-on tenter si cette commande échoue ? Justifier précisément votre réponse.
- Ecrire une commande qui fait la même chose en utilisant les commandes `kill` et `pidof`.

**Question 1.5** La commande `file` permet de déterminer le type d'un fichier. Exemples :

```
$ file more_less.sh
more_less.sh: a /bin/dash script, ASCII text executable
$ file prog.c
prog.c: C source, ASCII text
$ file prog
prog: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), ...
```

Quand elle détecte qu'un fichier est un fichier texte, la sortie produite par la commande `file` contient le mot "`text`". Ecrire un script, utilisant la commande `file`, qui calcule et affiche la somme des nombres de lignes des fichiers texte contenus dans le répertoire courant, sans explorer ses éventuels sous-répertoires, et sans compter le nombre de lignes contenues dans le script lui-même.

**Question 1.6** Liens physiques et liens symboliques :

- Peut-on créer un lien physique nommé `rep2` pointant sur un inode de répertoire `rep` déjà existant ? Si oui, écrire la commande permettant de le faire ?
- Peut-on créer un lien symbolique nommé `rep3` pointant sur un inode de répertoire `rep` déjà existant ? Si oui, écrire la commande permettant de le faire ?

**Question 1.7** Variable `PATH` :

- Que contient cette variable ?
- Est-ce une variable d'environnement ou une variable interne du shell ?
- Affecter dans une variable `VAR` la première valeur mémorisée dans la variable `PATH`, en n'utilisant pas, si possible, de commande externe.

# Exercice 2 : Programmation en C

Synopsis du programme à écrire :

```
./lines_between N1 N2 [FILE]
```

Le programme `lines_between` écrit sur sa sortie standard les lignes du fichier, qu'il traite, dont le numéro est compris dans l'intervalle [`N1`, `N2`], bornes incluses. Si l'argument `FILE` est absent, le programme lit ses données sur son entrée standard. On considère que la première ligne du fichier traité porte le numéro 1 : pour être valides, `N1` doit donc être strictement positif, et `N2` doit être supérieur ou égal à `N1`.

Remarque : il n'est pas demandé de détecter une erreur si l'argument correspondant à un nombre contient des caractères invalides après des chiffres. Par exemple, les valeurs `17abc` ou `17.23` seront considérées comme étant valides et égales à `17`.

Exemples d'exécution :

```
$ cat fic
Question du jour :
Quelle est le contraire de microsoft office ?
Reponse :
macrohard onfire !
$ ./lines_between 1 2 fic
Question du jour :
Quelle est le contraire de microsoft office ?
$ ./lines_between 5 8 fic
$ ./lines_between 4 15 fic
macrohard onfire !
$ ./lines_between 3 2
Usage : ./lines_between N1 N2 [FILE]
N1 must be strictly greater than 0, and N2 greater than or equal to N1
```

**Question 2.1** Ecrire le programme `lines_between` en C en contrôlant la validité des arguments transmis, et en traitant les erreurs lors des ouvertures et fermetures de fichiers.

# Exercice 3 : Commandes shell et programmation shell

Le fichier `/etc/passwd` contient une ligne par utilisateur. Le format du fichier est le suivant :

```
login:password:UID:GID:name:home:shell
```

où :
- le champ `login` correspond au nom de connexion de l'utilisateur, c'est-à-dire au `username` de l'utilisateur
- le champ `password` n'est pas utilisé et est égal à `x`
- le champ `GID` est l'identifiant du groupe principal de l'utilisateur
- le champ `name` correspond au nom complet de l'utilisateur ou à un champ de commentaires, et **ne doit pas être utilisé** dans cet exercice

Le fichier `/etc/group` contient une ligne par groupe d'utilisateur. Le format du fichier est le suivant :

```
group_name:password:GID:user_list
```

où :
- le champ `group_name` correspond au nom du groupe d'utilisateurs
- le champ `password` n'est pas utilisé et est égal à `x`
- le champ `GID` est l'identifiant du groupe d'utilisateurs
- le champ `user_list` est une liste de `username` qui sont membres du groupe, séparés par des virgules (",")

Contenu partiel du fichier `/etc/passwd` :

```
root:x:0:0:root:/root:/bin/bash
sys:x:3:3:sys:/dev:/usr/sbin/nologin
eric:x:1000:1000:eric,,,:/home/eric:/bin/bash
eleve:x:1001:1001::/home/eleve:/bin/bash
eleve2:x:1002:1001::/home/eleve2:/bin/bash
```

Contenu du fichier `/etc/group` concernant les utilisateurs `eric`, `eleve` et `eleve2` :

```
adm:x:4:eric,syslog
cdrom:x:24:eric
sudo:x:27:eric
prof:x:1000:
student:x:1001:eleve2
```

**Question 3.1** Dans quel fichier les mots de passe des utilisateurs sont-ils stockés ? Sous quelle forme sont-ils stockés ? Quelle commande permet à un utilisateur de modifier son mot de passe ? Cette commande possède-t-elle une permission spéciale ? Justifier votre réponse.

**Question 3.2** À partir **uniquement** du fichier `/etc/passwd` et des commandes vues en cours, on veut connaître :

1. exclusivement le nombre d'utilisateurs du système (sans que le nom du fichier utilisé, ici `/etc/passwd`, soit affiché)
2. Les logins (ou usernames) des utilisateurs référencés dans les 3 dernières lignes du fichier `/etc/passwd`
3. Les UID des utilisateurs dont le login :
   — commence par la lettre 'e' écrite en minuscule ou en majuscule,
   — et contient en tout entre 4 et 5 lettres.
4. Les logins (ou usernames) des utilisateurs dont l'UID est compris entre 1000 et 9999, bornes incluses.
5. Les identifiants des groupes principaux des utilisateurs du système (les doublons doivent être supprimés)
6. Les identifiants des 3 groupes qui ont le plus grand nombre d'utilisateurs ayant, pour groupe principal, le groupe ciblé. Les identifiants de groupe doivent être triés dans l'ordre décroissant suivant le nombre d'utilisateurs.

La suite de cet exercice a pour objectif d'écrire un script nommé `Groupes.sh`.

Synopsis du programme à écrire :

```
./Groupes.sh USERNAME
```

Ce script prend en argument le login (ou `USERNAME`) d'un utilisateur, et affiche le login reçu en argument, suivi d'un caractère deux points (" :"), suivi de la liste des noms des groupes auquels l'utilisateur appartient.

Exemples d'exécution du script correspondants aux contenus des fichiers donnés ci-dessus :

```
$ ./Groupes.sh eric
eric : prof adm cdrom sudo
$ ./Groupes.sh eleve
eleve : student
$ ./Groupes.sh eleve2
eleve2 : student
```

Le traitement demandé doit être effectué à partir des informations contenues dans les fichiers `/etc/passwd` et `/etc/group`. Dit autrement, vous n'avez pas le droit d'utiliser des commandes comme `id` ou `groups`.

Le script doit commencer par vérifier que :
— les fichiers `/etc/passwd` et `/etc/group` sont des fichiers réguliers avec la permission de lecture ;
— le script est appelé avec le bon nombre d'argument ;
— et que l'utilisateur de login égal à `USERNAME` existe bien.
En cas d'erreur, il faut afficher un message destiné à l'utilisateur et terminer le script.

Le script doit définir au moins une fonction qui renvoie la liste des noms des groupes, trouvés dans le fichier `/etc/group`, auquels l'utilisateur de login égal à `USERNAME` appartient. Le `USERNAME` de l'utilisateur doit être passé en paramètre à cette fonction. Par exemple, pour les contenus des fichiers donnés ci-dessus, cette fonction doit renvoyer :

- pour `eric`, la liste `adm cdrom sudo`
- pour `eleve`, une liste vide
- et pour `eleve2`, une liste contenant uniquement le groupe `student`

Indication : il ne faut pas oublier d'afficher le nom du groupe principal de l'utilisateur, mais il faut faire attention à ne pas l'afficher 2 fois car il peut déjà être présent dans la liste renvoyée par la fonction précédente.

**Question 3.3** Ecrire le script `Groupes.sh` en respectant les indications données ci-dessus, et en veillant à bien **factoriser** votre code.

# Annexe

## Prototypes

```
FILE *fopen(const char *path, const char *mode);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *fp);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *fp);
int fprintf(FILE *stream, const char *format, ...);
int fclose(FILE *fp);
int feof(FILE *fp);
int ferror(FILE *fp);
int atoi(const char *nptr);
int strcmp(const char *s1, const char *s2);
void perror(const char *s);
```

## Commandes

```
awk [OPTION]... [ -- ] program-text [input-file]...
basename NAME [SUFFIX]
chmod [OPTION]... [MODE|SYMBOL] FILE...
cp [OPTION]... [-T] SOURCE DEST
cut [-dSEP] [-fLIST] [FILE]
dirname [OPTION] NAME...
file [OPTION]... FILE...
grep [-E] [-v] PATTERN [FILE]
head [-n K] [FILE]...
kill [-signal|-s signal] pid...
killall [-s,--signal signal] command...
ln [-s] target link
mv [OPTION]... [-T] SOURCE DEST
pidof program...
mkdir [-p] dir
sed [-E] [-e] 's/PATTERN/REPLACEMENT/[INDICATOR]' [FILE]
seq [OPTION]... FIRST LAST
sort [-n] [-r] [FILE]
tail [-n K] [FILE]...
test [-e FILE] [-d FILE] [-f FILE] [-r FILE] [-s FILE]
test [-z STR] [-n STR] [STR1 = STR2] [STR1 != STR2]
test [arg1 OP arg2]     # avec OP = -eq, -ne, -lt, -le, -gt, ou -ge
rm [OPTION]... [FILE]...
uniq [-c] [-u] [FILE]
wc [-c] [-l] [-w] [FILE]
```

## Extraction de sous-chaîne

```
${VAR%motif} renvoie une chaîne après suppression du plus petit suffixe correspondant avec motif
${VAR%%motif} renvoie une chaîne après suppression du plus grand suffixe correspondant avec motif
${VAR#motif} renvoie une chaîne après suppression du plus petit préfixe correspondant avec motif
${VAR##motif} renvoie une chaîne après suppression du plus grand préfixe correspondant avec motif
```