

# TD 5 - ASR7 Programmation concurrente

## Administration système

Sylvain Brandel, Guillaume Damiand, Laurent Lefèvre, Matthieu Moy, Grégoire Pichon

Printemps 2022

### I Des droits et des fichiers

Sur le système considéré dans cet exercice, on trouve notamment les trois utilisateurs suivants :

- `a1400` qui fait partie des groupes `etudiant` et `user` ;
- `frico` qui fait partie des groupes `prof` et `user` ;
- `ycaiou` qui fait partie des groupes `prof`, `user` et `ycaiou`

Dans un répertoire ayant tous les droits d'accès, la commande `ls -l` retourne ce qui suit :

```
srw----- 1 ycaiou ycaiou 0   déc.  3 06:24 kdeinit4__0
lrwxrwxrwx 1 ycaiou ycaiou 33   mai  2 2014 rap.tex -> rapport.tex
drwxr-xr-x 27 frico  user  27   janv.  2 11:06 public/
drwx--x--- 70 frico  prof  4096  avril 22 17:21 prive/
-rw-rw-r-- 1 a1400  etu   5349  déc. 10 2008 rapport.tex
-rw-rw---- 1 frico  prof  336   avril 19 12:45 public/sujet.tex
-rw-rw-r-- 1 frico  prof  336   avril 19 12:45 public/sujet.pdf
-rw-r--r-- 1 ycaiou prof  336   avril 19 18:32 public/correction.pdf
-rw-r--r-- 1 frico  prof  336   avril 19 18:32 prive/notes.ods
```

**Q.I.1)** - Qu'est-ce que le fichier `kdeinit4__0`? Et `rap.tex`?

`kdeinit4__0` est une socket (on devine d'après le nom qu'il permet de parler à un démon lié à l'environnement de bureau KDE); `rap.tex` est un lien symbolique. On le voit sur la première colonne : `-` pour les fichiers « normaux », `s` pour les sockets, `d` pour les répertoires (Directory), `l` pour les liens symboliques. On ne le voit pas sur cet exemple, mais il y a aussi `c` pour les « block device », qui correspondent à un périphérique (on en trouve dans `/dev/`).

**Q.I.2)** - Les droits sur `rap.tex` sont-ils normaux?

Oui : le lien symbolique a toujours des droits `rw-rwxrwx` : l'accès au fichier cible du lien est conditionné par les droits du fichier et non par ceux du lien.

**Q.I.3)** - Représentez dans une matrice les possibilités d'accès des fichiers et répertoires pour chaque utilisateur.

Fichier	ycaiou	frico	a1400
<code>kdeinit4__0</code>	RW	pas d'accès	pas d'accès
<code>rap.tex</code>	RWX	RWX	RWX
<code>public/</code>	R X	RWX	R X
<code>prive/</code>	X	RWX	pas d'accès
<code>rapport.tex</code>	R	R	RW
<code>public/sujet.tex</code>	RW	RW	pas d'accès
<code>public/sujet.pdf</code>	RW	RW	R
<code>public/correction.pdf</code>	RW	R	R
<code>prive/notes.ods</code>	R	RW	Pas d'accès

Attention : le fichier `prive/notes.ods` est accessible en lecture au groupe (et aux autres, mais à cause des droits de `prive/`, le groupe ne peut le lire), mais il n'est pas possible de lister le contenu du répertoire `prive/` (`ls -l prive/` renverra une erreur), donc il faut connaître le nom du fichier pour pouvoir y accéder.

**Q.I.4)** - Qu'est-ce qu'une telle « configuration » des droits sur `public/` et `prive/` permet de faire ?

Avoir une zone de partage en lecture et écriture entre prof pour `prive/`. `x` permet au groupe d'accéder au contenu de `prive/`, mais ne peut lister son contenu car le bit `r` n'est pas présent. Attention, le `x` est trompeur : sur un répertoire il ne s'agit pas de la permission d'exécuter (ce qui n'aurait pas de sens pour un répertoire), mais de la permission d'accéder au contenu du répertoire. Le bit `r` sur un répertoire est le droit de lister son contenu. Un utilisateur du groupe `prof` qui connaît les noms des fichiers dans `prive/` peut donc y accéder, mais quelqu'un qui ne connaît pas les noms des fichiers ne peut pas les découvrir.

On peut imaginer par exemple un binaire avec des droits `--x` pour le groupe. Seul quelqu'un du groupe peut exécuter, mais il ne peut pas lire ni modifier le code !

Si droit en écriture, un fichier déposé peut être modifié par quiconque connaît son existence : pour partager des notes s'il n'y avait pas TOMUS par exemple. Attention aux accès concurrents par contre !

Dans notre cas, l'accès en lecture à `notes.ods` est autorisé pour le groupe seulement, mais il faut quand même savoir que le fichier existe au préalable : c'est une forme de protection du fichier (pas aussi forte qu'un vrai mot de passe).

Avoir une zone d'accès pour tous, donc aussi les étudiants dans `public`.

Si temps restant....

Rqs sur fichiers : `file` (et numéro magique), `iconv` (et pbs d'encodage).

Retour sur fonction `mount` et `/etc/passwd` (et NFS et LDAP).

`lsuf`, etc.

Rappels sur `p` pour les pipes (comme `s` sockets, `d` directory) Rappels sur le bit `setUID`, noté `S` si pas `x`.

Parler du sticky bit :

```
drwxrwxrwt 12 root root 20480 12 déc. 22:21 /tmp
```

obtenu avec `chmod +t`, il interdit aux non-propriétaires de fichiers de renommer ou d'effacer les fichiers de ce répertoire.

Dans notre exemple, s'il y avait le sticky bit au répertoire parent, seul `ycaniou` pourrait effacer `rap.tex`.

Si le répertoire n'a pas les droits en exécution, il est noté `'T'`.

## II Le bit « setuid »

On considère maintenant cette situation (le `$` est l'invite de commande, ou « prompt ») :

```
$ sudo ls /root
$ cat test-setuid.c
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    int f = open("/root/i-can-haz-write-access-to-r00t.txt",
```

```

        O_WRONLY | O_CREAT);
    if (f < 0) {
        perror("Can't open file");
        exit(1);
    }
    write(f, "pwned", strlen("pwned"));
    close(f);
}
$ gcc test-setuid.c -o test-setuid
$ ls -l test-setuid
-rwxr-xr-x 1 moy moy 8808 nov. 30 17:27 test-setuid*
$ ./test-setuid
Can't open file: Permission denied
$ sudo chown root test-setuid
$ ./test-setuid
Can't open file: Permission denied
$ sudo chmod +s test-setuid
$ ls -l test-setuid
-rwsr-sr-x 1 root moy 8808 nov. 30 17:27 test-setuid*
$ ./test-setuid
$ sudo ls /root
i-can-haz-write-access-to-r00t.txt
$

```

**Q.II.1)** - Pourquoi les premières exécutions de `test-setuid` échouent-elles ? Pourquoi la dernière réussit-elle ?

L'exécutable essaie d'écrire dans le répertoire `/root`, qui est le répertoire `$HOME` de l'utilisateur `root`. C'est un utilisateur non-privilegié qui le fait, il n'est pas autorisé.

On modifie ensuite les droits sur le fichier pour que le propriétaire du fichier soit `root` et que le bit `setuid` soit positionné (le `s` qui remplace le `x` dans la sortie de `ls -l`, positionné par `chmod +s`). Le bit `setuid` signifie « quand le binaire est exécuté, il l'est en utilisant l'utilisateur propriétaire ». En d'autres termes, les utilisateurs qui lancent ce binaire sont `root` le temps de son exécution. C'est un moyen d'accorder des droits à un utilisateur, mais c'est aussi un gros danger potentiel : ne pas le positionner sur n'importe quel exécutable !

Deux exécutables bien connus l'utilisent : `su` et `sudo`.

### III Admin Système

Lorsque la ou les commandes sont propres à une distribution, nous supposons qu'il s'agit d'une distribution Debian ou dérivée (Ubuntu, comme votre VM Openstack).

**Q.III.1)** - Rappelez quelques variables d'environnement importantes. Comment toutes les lister avec leur affectation ?

`PATH, PS1, LD_LIBRARY_PATH, HOME, PWD, SHELL, USER, HOSTNAME...`

Avec la commande `env`.

**Q.III.2)** - Quelle est la différence entre sourcer un script (`source mon-script.sh` ou `. mon-script.sh`) et l'exécuter (`./mon-script.sh`) ?

Rappel d'ASR5 : exécuter provoque un `fork()` qui se terminera à la terminaison du programme appelé.

Donc Si le programme est un script qui met en place un environnement d'exécution (à l'aide de variables d'environnement) pour peut-être ensuite appeler un binaire, l'environnement d'exécution se termine avec le `fork()`.

Les commandes exécutées ensuite dans le terminal ne sont donc pas affectées par d'éventuelles variables qui auraient été initialisées après le `fork()`.

Sourcer un script permet justement de pouvoir mettre en place des variables d'environnement pour l'environnement actuel.

**Q.III.3)** - Comment mettre en place la complétion *intelligente*? À quoi sert-elle?

Il faut sourcer le fichier `/etc/bash_completion` (c'est souvent fait par défaut). Elle sert à avoir des propositions pertinentes en fonction du contexte quand on appuie sur TAB (exemples : `git [TAB]` liste les commandes `git`, `kill [TAB]` liste les PID, `ls --[TAB]` liste les options de `ls`, ...)

**Q.III.4)** - Quel est le pseudo système de fichiers permettant de configurer dynamiquement le fonctionnement du noyau ?

`/proc/` et `/sys/`

Les commandes suivantes peuvent se faire avec `apt`, un front-end qui se veut convivial pour gérer les paquets Debian. Mais `apt` ne fournit pas encore toutes les choses des autres outils, comme la recherche du package auquel appartient un fichier donné.

**Q.III.5)** - Comment lister tous les packages installés ?

`dpkg --list` ou `dpkg --get-selections`

**Q.III.6)** - Comment savoir si le package `apache2` est installé ?

`dpkg -l apache2` et on regarde si le *flag* `ii` est mis (premier « i » = « Desired », deuxième « i » = Status) ou `dpkg --get-selection | grep apache2`, ou encore `apt policy apache2` et on regarde si on trouve `***`.

Si la base de paquets n'a pas été mise à jour récemment, il faut le faire avec `apt update`.

**Q.III.7)** - Où modifier les *repositories* d'où la liste des packages et leur version, ainsi que les packages eux-mêmes seront téléchargés ?

`/etc/apt/sources.list`, et les fichiers `/etc/apt/sources.list.d/*.list` (qui permettent par exemple aux paquets d'ajouter un fichier de liste sans toucher au fichier principal `/etc/apt/sources.list`).

**Q.III.8)** - Comment savoir quelle version d'un package sera installée si on décide de l'installer ?

Méthode éléphant : `apt-get -s install lePackage` pour obtenir la version à partir de l'affichage de la commande, qui va simuler l'installation (le `-s`).

`apt-cache showpkg lePackage` permet de savoir de quel *repository* et dans quelle version le package sera installé (avec d'autres informations comme les sommes de contrôle permettant de savoir que le package n'a pas été altéré (*backdoor*)).

`apt policy lePackage` liste les versions disponibles.

**Q.III.9)** - Comment mettre à jour l'ensemble du système ?

`apt update`, puis `apt upgrade` ou `apt full-upgrade`. Attention, `update` ne met pas les packages à jour : la commande télécharge juste la liste des packages. `upgrade` fait une mise à jour de tous les packages, sauf ceux nécessitant une suppression de package. A utiliser pour les mises à jour mineures. `full-upgrade` met tous les packages à jour, éventuellement en supprimant des packages. A utiliser pour le passage d'une version à l'autre de distribution.