TP

Manipulation de fichiers et droits d'accès

Résumé

Dans ce TP, nous allons étudier la manipulation de fichiers sous Unix du point de vue de l'utilisateur et du programmeur. Nous nous intéresserons plus particulièrement aux permissions du système de fichier.

1 Commandes shell

Donner brièvement une description des commandes citées ci-dessous (ainsi que des options correspondantes lorsqu'elles sont indiquées). Pour cela, vous pourrez consulter les pages de manuel correspondantes et/ou essayer les commandes vous même.

`mkdir`

`rmdir`

`rm`

- `-r`
- `-i`

`ls`

- `-l`
- `-h`
- `-a`

pwd

cd (si vous ne connaissez pas, on trouve sa description dans la page de manuel de `bash` (pourquoi ici ?))

id

groups

chown

chgrp

chmod

Avec la dernière commande (`chmod`), modifiez les permissions de vos fichiers : enlevez vos droits de lecture, d'écriture, d'exécution (pour un programme) et à chaque fois testez l'effet. Vous pouvez vérifier l'état des permissions de vos fichiers avec la commande `ls` (et les bonnes options).

2 Appels systèmes et manipulation de fichiers

Ecrivez un programme en C qui copie le contenu d'un fichier `file.txt` qui vous aurez précédemment créé dans un nouveau fichier `file_copy.txt`.

Attention : Pour créer le fichier `file_copy.txt` avec les bon droits d'accès vous devez utiliser l'appel système `open()` avec les paramètres suivants :

- Les *flags* : `O_CREAT | O_RDWR`
- Les *mode* – définition des droits d'accès : `S_IRUSR | S_IWUSR`

3 Identité

Quel est votre nom utilisateur ? Quel est votre groupe principal ? Appartenez-vous à d'autres groupes ? Qu'en est-il pour votre enseignant ? (login `tropars` par exemple)

4 Permissions et utilisateurs

Utilisez deux comptes différents en même temps.

Note:
Vous pouvez utiliser la commande `ssh -X -l` *`login`* `localhost` pour obtenir un shell appartenant à un autre utilisateur sur la même machine.

Quels droits devez-vous positionner pour que votre binome puisse modifier un de vos fichiers ? Et pour qu'il puisse l'effacer ?

Note:
Travaillez dans un sous répertoire de `/tmp` pour éviter de faire des bêtises avec les fichiers de votre compte.

5 Droits des fichiers

Lancez le programme `extract_data` [1] depuis un répertoire où se trouve le fichier `trouvez_fichiers.data`. Cela crée un répertoire nommé `trouvez_fichiers`. Explorez ce répertoire [2] **en vous servant des commandes shell citées précédemment** pour accéder aux fichiers. Trouvez les 4 fichiers C et exécutez le Makefile dans le répertoire qui en comporte un. [3].

6 Projets en binômes

Vous aurez des projets à faire où vous serez quelques personnes qui devront lire/créer/modifier/effacer les mêmes fichiers. Mais les personnes extérieures à votre projet ne devront pas avoir accès à ces fichiers.

Proposez une solution basée sur le système de permissions Unix pour régler ce problème.

7 Les permissions Unix spéciales

En plus des 3 séries de permissions `rwx`, les systèmes unix ont 3 bits supplémentaires. Lors de l'affichage (`ls -l`), ce bit apparaît en affichant une lettre différente à la place du `x`.

7.1 SetUID et SetGID

1. Observez les permissions du programme `/bin/su` et/ou `/usr/bin/sudo`. Que font ces programmes ?
2. Comment pouvez-vous expliquer le rôle de la permission `s` observée sur ces deux programmes ?

7.2 Restricted deletion flag ou Sticky bit

Les manipulations ici doivent être faites sous plusieurs identités. Le plus simple est d'utiliser la machine mandelbrot et travailler en groupe dessus.

Créez un répertoire `toto` avec des droits `rwx` pour tout le monde.

1. C'est un programme compilé pour linux amd64, donc à exécuter sur la machine `mandelbrot` par exemple
2. On trouve sous ce répertoire 4 fichiers C
3. Ce sont des programmes vainqueurs du concours IOCCC, donc des exemples de ce qu'il **ne** faut **pas** faire quand on écrit du code C.

1. Demandez à un de vos camarades de venir créer un fichier dans ce répertoire [4]. Quelles sont les permissions du fichier créé et à qui appartient-il ? Pouvez-vous effacer ce fichier ? Qu'est-ce qui permet ce comportement ?
2. Demandez à nouveau à votre camarade de recréer un fichier dans votre dossier aux permissions laxistes. Puis, demandez à un autre camarade d'effacer ce fichier. À nouveau, quelles sont les permissions du fichier créé et à qui appartient-il ? Est-il possible à cet autre camarade d'effacer le fichier ? Qu'est-ce qui permet ce comportement ?
3. Refaites toutes ces manipulations et répondez aux mêmes questions après avoir ajouté la permission `t` au dossier initialement créé [5]. Que constatez-vous ? Voyez-vous un intérêt à cette permission ?

8 Liste de contrôle d'accès (ACL)

De nos jour, Linux possède un système complémentaire de gestion des droits d'accès : les listes de contrôle d'accès (*Access Control Lists* ou ACL). Avec cette extension au système classique, en plus des permissions `rwx` pour le propriétaire du fichier, le groupe du fichier et les autres, on peut donner (ou pas) des permissions `rwx` pour n'importe quel autre utilisateur et/ou groupe.

Les commandes pour manipuler les ACL sont `getfacl` et `setfacl`. Lisez la documentation.

1. À quoi sert le `mask` ?
2. À quoi servent les entrées commençant par `default:` ?
3. Comment utiliser les ACL pour résoudre le problème précédent (projets en binômes) ?

4. Pour créer un fichier texte d'une ligne, il suffit d'une commande telle que
`echo "mon texte" > fichier`
5. `chmod +t dossier`