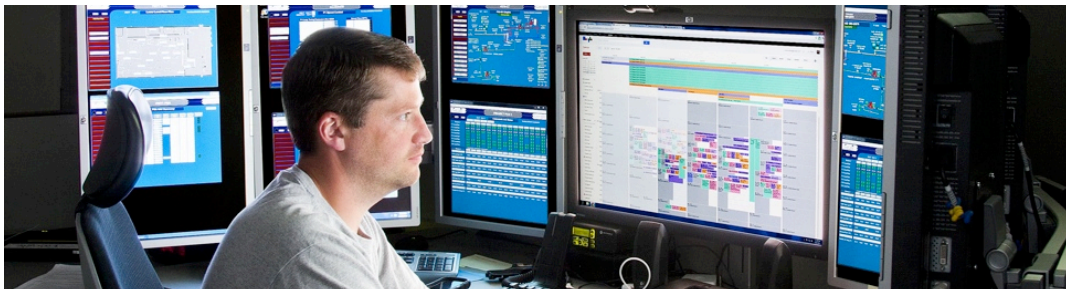
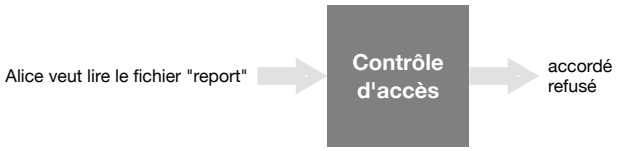
Administration Système — Le contrôle d'accès



Année académque 2014/15

Le contrôle d'accès
Introduction

- Un système d'exploitation offre
  - des *comptes* pour des utilisateurs individuels
  - un vaste choix *d'opérations* : éditer des fichiers de texte, se loguer sur un ordinateur distant, changer le nom du système, installer des nouveaux logiciels, etc.
- Un *système de contrôle d'accès* est une boîte noire qui considère des action potentielles (paire utilisateur/opération) et sur chacune rend une décisions si elle est admissible ou pas.



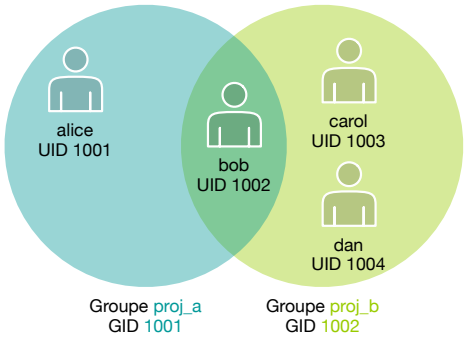
- Dans le cas de Unix/Linux, en réalité il n'y a pas une seule boîte noire, mais plusieurs, dispersés à travers le système.

Le contrôle d'accès
Introduction

- Le contrôle d'accès d'Unix a été conçu en suivant ces règles générales :
  - Des objets (fichiers et processus) ont des *propriétaires*. Les propriétaires ont un vaste contrôle sur ces objets.
  - On devient propriétaire des objets que l'on crée.
  - Le compte spécial appelé *root* peut agir comme propriétaire de tout objet.
  - Seulement *root* peut effectuer certaines opérations d'administration sensibles.

Le contrôle d'accès
Le modèle traditionnel Unix

- Avant de pouvoir utiliser un système Unix, un utilisateur doit obtenir un compte. À sa création
  - l'utilisateur reçoit un **UID** *(User Identification)*, un nombre entier unique qui identifie l'utilisateur
  - l'utilisateur est rattaché à un groupe au moins (groupe principal)
    - chaque groupe possède un identifiant unique **GID** (Group Identification)
- Chaque utilisateur et groupe reçoit aussi un nom qui est affiché dans la plupart des interfaces, mais en interne le système travaille uniquement avec les UID et GID.
  - Les noms UID sont stockés dans `/etc/passwd`.
  - Les noms GID sont stockés dans `/etc/group`.
- La commande `id` permet d'afficher ces informations.

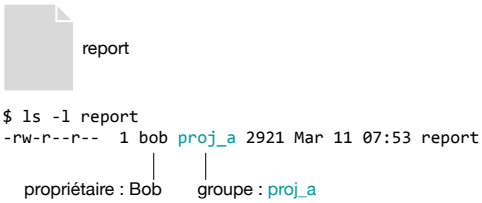



```
$ id bob
uid=1002(bob) gid=1001(proj_a)
groups=1001(bob),1002(proj_b), ...
$
```

Le contrôle d'accès

Le modèle traditionnel Unix

- Dans le modèle traditionnel Unix, chaque fichier est attribué
  - un **propriétaire *(owner)***, identifié par son UID,
  - un **groupe *(group)***, identifié par son GID.
- La commande `ls -l` permet d'afficher ces informations (`ls -ln` pour voir les UID et GID numériques).



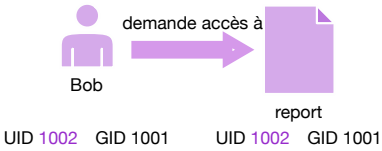
- Seul le propriétaire peut changer les droits d'accès du fichier.

- À travers le groupe, plusieurs utilisateurs peuvent accéder au fichier.
  - Le propriétaire décide sur l'accès groupe.
  - Ceci permet de partager des fichiers pour le travail en équipe.

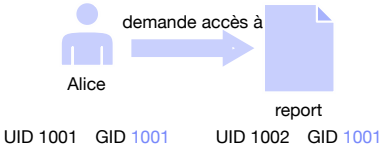
Le contrôle d'accès

Le modèle traditionnel Unix

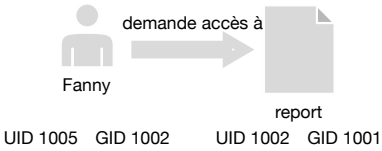
- Quand un utilisateur veut accéder à un fichier, le système distingue trois cas de figure :
  - L'UID de l'utilisateur est-il identique à l'UID du fichier ? Si oui l'utilisateur est le **propriétaire (owner)** du fichier.



  - Si les UID sont différents, un des GIDs de l'utilisateur est-il identique au GID du fichier ? Si oui, l'utilisateur **appartient au groupe (group)** associé au fichier.



  - Dans les autres cas (aucune correspondance) : l'utilisateur appartient au **reste du monde (others)**.



Le contrôle d'accès

Le modèle traditionnel Unix

- Pour chaque cas de figure existe un jeu de droits d'accès, attachés au fichier
  - Les droits *owner*
  - Les droits *group*
  - Les droits *others*
- Chaque jeu de droits comprend trois actions qui sont permises ou pas (flag binaire)
  - L'action de lécture *(read, r)*
  - L'action d'écriture *(write, w)*
  - L'action d'exécution *(execute, x)*

```
drwxr-xr-x  8 marcel.graf marcel.graf 4096 Mar 11 07:56 .
drwxr-xr-x 25 root        root        4096 Mar  5 12:37 ..
-rw-r--r--  1 marcel.graf marcel.graf 3593 Mar  6 16:31 .bashrc
-rw-r--r--  1 marcel.graf marcel.graf 3021 Mar 11 07:56 ch1
drwxrwxr-x  2 marcel.graf marcel.graf 4096 Mar 11 07:56 play
-rw-r--r--  1 marcel.graf proj_a      2921 Mar 11 07:53 report
```

Owner (user) | Group | Others | Owner | Group

Le contrôle d'accès

Le modèle traditionnel Unix — Les permissions des fichiers ordinaires

- La signification exacte des actions *read* / *write* / *execute* dépend du type de fichier.
- Pour les fichiers ordinaires
  - ***read*** : Le contenu du fichier peut être lu, chargé en mémoire, visualisé, recopié.
  - ***write*** : Le contenu du fichier peut être modifié, on peut écrire dedans. La suppression n'est pas forcément liée à ce droit (voir droits sur répertoire).
  - ***execute*** : Le fichier peut être exécuté depuis la ligne de commande, s'il s'agit soit d'un programme binaire (compilé), soit d'un script (shell, Python, ...).
- Exemple : `-rwxr-xr--  1 bob  proj_a  2921 Mar 11 07:53 create_backup`
  - Le fichier **`create_backup`** appartient à l'utilisateur **`bob`** et au groupe **`proj_a`**.
  - Le propriétaire (**`bob`**) a les droits de lecture, écriture et exécution.
  - Le groupe (les membres du groupe **`proj_a`**) a les droits de lecture et exécution.
  - Les autres ont le droit de lecture.

Exercice 04.01

- Pour les fichiers suivants, déterminez qui est le propriétaire, à quel groupe est rattaché le fichier et caractérisez le groupe de personnes qui peut lire, écrire et exécuter le fichier.
  - `/etc/passwd`
  - `/bin/ls`
  - `~/.bashrc`
  - `~/.bash_history`

Le contrôle d'accès

Le modèle traditionnel Unix — Les permissions des répertoires

- Pour les répertoires, les actions *read*, *write*, et *execute* ont la signification suivante
  - ***read*** : On peut lister les noms des fichiers dans le répertoire.
  - ***write*** : On peut créer, renommer et supprimer des fichiers dans le répertoire. Pour protéger les fichiers de la suppression, enlever cette action.
  - ***execute*** : On peut examiner les méta-données d'un fichier dans le répertoire. On peut faire le répertoire le répertoire courant (commande `cd`).
- Si on a seulement la permission **x** pour un répertoire, on ne peut pas utiliser **ls** pour afficher les fichiers du répertoire, mais on peut afficher un fichier si on connait son nom.

```
$ who am i
zach pts/7 Aug 21 10:02
$ ls -ld /home/max/info
drwx-----x. 2 max pubs 4096 08-21 09:31 /home/max/info
$ ls -l /home/max/info
ls: /home/max/info: Permission denied
$ ls -l /home/max/info/financial /home/max/info/notes
-rw-------. 1 max pubs 34 08-21 09:31 /home/max/info/financial
-rw-r--r--. 1 max pubs 30 08-21 09:32 /home/max/info/notes
$
```

Le contrôle d'accès

Le modèle traditionnel Unix — Les permissions des répertoires

- La possibilité de lire un fichier ou non ne dépend pas des droits du répertoire, elle dépend uniquement des droits du fichier.

```
$ ls -l /home/max/info/financial /home/max/info/notes
-rw-------. 1 max pubs 34 08-21 09:31 /home/max/info/financial
-rw-r--r--. 1 max pubs 30 08-21 09:32 /home/max/info/notes
$ cat /home/max/info/notes
This is the file named notes.
$ cat /home/max/info/financial
cat: /home/max/info/financial: Permission denied
$
```


Exercice 04.02

- a) Examinez les droits de votre répertoire personnel.
  - Qui est le propriétaire et à quel groupe est rattaché le répertoire ?
  - Quelle est la configuration des droits ?
    - Qui peut lister les fichiers ?
    - Qui peut créer des fichiers ?
- b) Grâce à quels droits pouvez-vous créer des fichiers dans le répertoire `/tmp` ?

Le contrôle d'accès

Le modèle traditionnel Unix — Les permissions des répertoires — Sticky bit

- Certains répertoires comme `/tmp` permettent à tout utilisateur de créer des fichiers. Ils sont donc configurés avec la permission **w** pour *others*.
  - Cette permission met en danger un fichier créé par un utilisateur, car tout autre utilisateur peut le supprimer (ou renommer).
- Pour pallier à ce problème on utilise le *sticky bit*, affiché par un **t** à la place du **x**. Il est aussi appelé *restricted deletion flag*.
  - Le *sticky bit* peut être seulement activé par *root*.
    - Commande : `chmod +t` *répertoire*
  - S'il est activé sur un répertoire, un fichier peut être renommé ou supprimé seulement par un utilisateur qui a la permission **w** sur le répertoire **et est le propriétaire du fichier**.

Le contrôle d'accès

Le modèle traditionnel Unix — Les outils pour modifier les permissions

- Un système Unix comprend trois outils pour modifier les permissions d'un fichier.
  - Le propriétaire peut utiliser **`chmod`** pour changer les flags de permission d'un fichier.
  - Le propriétaire peut utiliser **`chgrp`** pour changer le groupe auquel appartient un fichier.
  - L'utilisateur *root* peut utiliser **`chown`** pour changer le propriétaire d'un fichier.

Le contrôle d'accès

Le modèle traditionnel Unix — Les outils pour modifier les permissions — **chmod**

- La commande **`chmod`** *(change mode)* peut être utilisée par le propriétaire d'un fichier pour changer les flags de permissions.
- Elle connait deux modes de fonctionnement
  - Mode symbolique : `chmod a+rw letter`
  - Mode numérique : `chmod 600 letter`

Le contrôle d'accès

Le modèle traditionnel Unix — Les outils pour modifier les permissions — **chmod**

- Dans le **mode symbolique** on choisit pour quelle catégorie d'utilisateurs on veut changer les flags
  - le propriétaire, *owner*, abrégé par **u** (!)
  - le groupe, *group*, abrégé par **g**
  - les autres, *others*, abrégé par **o**
  - tous (u, g et o), *all*, abrégé par **a**
- Ensuite on spécifie si les permissions sont à
  - ajouter, abrégé par **+**
  - retirer, abrégé par **-**
  - affecter, abrégé par **=**
- Finalement on spécifie les permissions (**r**, **w**, **x**)

```
chmod  u  +   r w x   file
       g  -   □ □ □
       o  =
       a
```

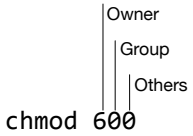
Exemple : Pour tous (**a**) ajouter (**+**) les droits de lecture (**r**) et écriture (**w**)

```
$ ls -l letter.0210
-rw-r-----. 1 sam pubs 6193 02-10 14:22 letter.0210
$ chmod a+rw letter.0210
$ ls -l letter.0210
-rw-rw-rw-. 1 sam pubs 6193 02-10 14:22 letter.0210
$
```

Le contrôle d'accès

Le modèle traditionnel Unix — Les outils pour modifier les permissions — **chmod**

- Dans le **mode numérique** on change tous les neuf flags à la fois.
  - Un groupe de permissions **r**, **w** et **x** est représenté par un nombre octal.
    - La permission **r** correspond au poids 4
    - La permission **w** correspond au poids 2
    - La permission **x** correspond au poids 1
  - Il y a trois groupes de permissions, pour *owner*, *group* et *others*, donc trois chiffres octaux dans cet ordre.



- Exemple 1 : Seulement le propriétaire du fichier peut lire (**4**) et écrire (**2**), le groupe et les autres n'ont aucune permission

```
$ chmod 600 letter.0210
$ ls -l letter.0210
-rw-------. 1 sam pubs 6193 02-10 14:22 letter.0210
$
```


Le contrôle d'accès

Le modèle traditionnel Unix — Les outils pour modifier les permissions — **chmod**

- Exemple 2 : Le propriétaire a le droit de lire (**4**), écrire (**2**) et exécuter (**1**), le groupe et les autres ont le droit de lire (**4**) et exécuter (**1**)

```
$ chmod 755 check_spell
$ ls -l check_spell
-rwxr-xr-x. 1 sam pubs 766 03-21 14:02 check_spell
$
```

Exercice 04.03

- a) Modification des droits d'accès
  - Créez un fichier et initialisez ses droits à `rw- --- ---` en utilisant la commande `touch file; chmod 600 file`
  - En utilisant chmod en mode symbolique, créez les configurations suivantes
    - `rw- r-- ---`
    - `rwx r-x ---`
    - `r-- r-- r--`
    - `rwx r-- r--`
    - `rwx --- ---`

- b) Droits d'accès contradictoires
  - Créez un fichier (vous en serez le propriétaire) avec une configuration de droits où
    - ni le propriétaire ni le groupe peut écrire
    - les autres si peuvent écrire.
  - Que fait le contrôle d'accès si vous essayez d'écrire dans ce fichier ?

Le contrôle d'accès

Le modèle traditionnel Unix — Les outils pour modifier les permissions — **chgrp** et **chown**

- Le propriétaire d'un fichier peut changer le groupe auquel appartient un fichier.
  - Il doit être membre du nouveau groupe.
- Exemple : Donner au groupe **`proj_a`** le fichier **`report`**
  - **`chgrp proj_a report`**

- Seulement un utilisateur avec des privilèges **root** peut changer le propriétaire d'un fichier.
- Exemple : Faire l'utilisateur **`alice`** propriétaire du fichier **`report`**
  - **`chown alice report`**

Le contrôle d'accès

Le modèle traditionnel Unix — Les permissions par défaut à la création — La commande **umask**

- Chaque fois qu'un utilisateur crée un nouveau fichier, des permissions par défaut lui sont attribuées.
- On peut les modifier avec la commande **`umask`** *(user mask)*
  - **`umask`** travaille avec les *droits à retirer* dans un masque
  - En standard, un nouveau fichier est créé avec les permissions **`rwx r-x r-x`**.
  - Pour ce cas, la commande est **`umask 0022`** :
    - Le premier 0 indique que les chiffres sont en octal.
    - Le second 0 indique qu'aucun droit pour l'utilisateur est retiré, donc **rwx**.
    - Le 2 indique que le droit **w** (**2**) est retiré pour le groupe et les autres, donc **`r-x`**.
  - Bien sûr, quand un programme crée un fichier, il peut restreindre encore les droits. Par exemple un éditeur va créer un fichier texte en enlevant le droit **x**.
  - Quand on modifie les permissions par défaut avec **`umask`** l'action est valable pour les fichiers crées ensuite, sans effet rétroactif sur les fichiers déjà créés.
  - Quand on appelle **`umask`** sans arguments la commande affiche le masque courant.