# Architecture et Systèmes

Stefan Schwoon

Cours L3, 2018/19, ENS Cachan

# Utilisateurs dans Unix

Les systèmes d'exploitation modernes gèrent une multitude d'utilisateurs :

- "personnes naturelles"
- "utilisateurs virtuels" (admin, "daemons")
- voir `/etc/passwd` dans Unix

Objectif : partager des ressources en respectant les contraintes de sécurité, les données privées etc.

Aspects pratiques : mots de passe, dossier 'home', ...

On s'intéressera aux relations avec d'autres aspects (processus, fichiers, ...)

# Identifiants utilisateur et groupe

Chaque utilisateur est associé à un identifiant numérique (user id ou uid).

Chaque utilisateur appartient à un groupe *primaire*, et potentiellement à quelques groupes `supplémentaires`.

- chaque groupe possède son identifiant numérique (gid)
- voir `/etc/passwd` pour les groupes primaires
- voir `/etc/group` pour les groupes supplémentaires

Les groupes sont déterminés par l'administrateur.

La commande id sert à afficher ses identifiants utilisateur et groupe.

# Utilisateurs/groupes et processus

Les processus possèdent quelques attributs concernant les utilisateurs :

- l'identifiant utilisateur réel (getuid)
- l'identifiant utilisateur effectif (geteuid)
- l'identifiants groupe réel et effectif (getgid, getegid)
- une liste d'identifiants groupes supplémentaires (getgroups)

Ceux-ci déterminent la plupart des privilèges que possède un processus.

Normalement, ces identifiants sont hérités du processus père lors d'un `fork` (on verra quelques exceptions plus tard).

# Utilisateur réel et effectif

Utilisateur réel : l'utilisateur qui a lancé le processus

Utilisateur effectif : l'utilisateur qui détermine les privilèges du processus

Normalement, les deux identifiants sont identiques !

Exceptions : p.ex., `passwd` (pour changer de mot de passe)

> L'utilisateur réel est celui qui change son mot de passe.
>
> Or, il n'a pas le droit de modifier le fichier qui contient les mots de passe, seulement root les possède.
>
> L'utilisateur effectif d'un processus exécutant `passwd` est donc root.

Le mécanisme pour changer d'utilisateur effectif s'appelle `setuid` ; on verra quand on discute les systèmes de fichiers.

# Entrées/sorties

Les opérations entrées/sorties transmettent des données entre la mémoire et d'autres processus ou des périphériques.

Dans Unix/Posix, les entrées/sorties se font par des fichiers.

Un fichier dans Unix va au-délà d'une collection de données; c'est une structure de données abstraite qui possède au moins une opération de lecture et d'écriture.

# Exemples

Un fichier peut représenter des réalisations différentes, p.ex.:

- fichiers sur un disque dur
- une zone de mémoire temporaire - buffer (console, pipe);
- une structure dans le noyau (`/proc`);
- les connections réseau

Les accès se font uniformement par les mêmes opérations, mais selon le type de fichier le noyau fait appel à un pilote (*driver*) qui y correspond.

# Aspects de E/S

## Stockage des données

- système de fichiers
- droits d'accès
- organisation physique d'un disque dur

## Gestion au niveau des processus

- fonctions pour accéder aux fichiers/créer etc
- fonctions pour manipuler les données d'un fichier

# Système de fichiers

Unix gère un système de fichiers pour stocker des données au-délà de la vie d'un processus.

Ce système de fichiers est une arborescence :

- Les nœuds internes sont les dossiers (ou répertoires).
- Les feuilles sont des fichiers (ordinaires ou spéciaux).
- Sur certains nœuds on peut greffer un arborescence supplémentaire (p.ex. une partition, une clé USB) (mount point en anglais).

Voir `mount` pour une liste des systèmes greffés.

# Organisation d'un système de fichiers

Les nœuds sont référencés par des chemins:

chemin absolu: en commençant par la racine `/` et suivant les dossiers, p.ex. `/home/schwoon/toto.txt`

chemin relatif on l'interprète en commençant dans un *dossier actuel*, p.ex. `schwoon/toto.txt` si on se trouve dans `/home`.

Dans un chemin relatif, `..` veut dire le dossier en-dessus, `.` le dossier actuel.

Ce dossier actuel est un attribut du processus (modifier avec `chdir`).

Le dossier actuel est hérité par les processus fils.

Les chemins absolus et relatifs sont interprétés par toutes les fonctions qui gère les fichiers.

# Inœuds

Structure de données pour stocker les méta-données d'un fichier permanent, p.ex. sur disque dur

Une partie d'un disque dur est reservé pour stocker ces inœuds.

Les méta-données stockés dans un inœuds contiennent le suivant :
type, propriétaire, groupe, droits d'accès, nombre de pointeurs vers cet inœud, les blocks où les données du fichiers sont stockés, . . . , mais pas le nom du fichier.

# Relation entre fichiers et inœuds

Un inœud représente une unité de données sur disque; un fichier est une reférence vers un inœud avec un nom.

Pour la plupart des fichiers, cette relation est un-à-un (mais pas, p.ex., pour les dossiers).

`ls -i` donne les identifiants du inœud associé avec un fichier ;
`stat` affiche les méta-données d'un inœud.

# Organisation d'un dossier

Un dossier est un fichier spécial.

Ses données consistent d'une liste de son contenu, avec pour chaque item :

- son nom
- son inœud

Du coup plusieurs entrées peuvent référencer le même inœud avec des noms différents.

Un inœud (et les données stockés avec lui) est libéré lorsqu'on supprime le dernier lien vers lui (fonction `unlink` dans C).

# Utilisateurs/groupes et fichiers

Chaque fichier (plus précisement : chaque inœud) appartient à un utilisateur et à un groupe.

Les fichiers créés par un processus portent ses identifiants effectifs.

Droits d'accès sont déterminés par les uid et gid associés avec un processus :

- 9 "ugo" bits: (user,group,others) $\times$ (read,write,execute)
- 3 autres bits: setuid, setgid, sticky

Commandes: `chmod`, `chown`, `chgrp`, `umask`

# Droits d'accès : exemple

Supposons qu'un processus souhaite lire dans un fichier.

Si son id effectif utilisateur égale le propriétaire du fichier, on vérifie le bit *(user,read)*.

Si le groupe du fichier est soit égale au groupe effectif du processus, soit dans ses groupes supplémentaires, on vérifie le bit *(group,read)*.

Sinon, on vérifie le bit *(others,read)*.

Pour écrire : c'est l'analogue avec *write*.

Le bit *execute* est utilisé lorsqu'on utilise une fonction de la famille `exec`.

# Droits d'accès sur les dossiers

Les bits rwx ont une significance légèrement differente sur les dossiers :

read: on peut obtenir la liste des fichiers du dossier

write: on peut modifier la liste (créer, rénommer, supprimer des fichiers)

execute: on peut obtenir (avec `stat`) les méta-données des fichiers

# Setuid et setgid

Les bits setuid/setgid sur les *fichiers*:

Quand un processus exécute le fichier, ses identifiants effectifs deviennent ceux du fichier.

Les idenfients réels restent inchangés.

Le bit setgid sur les *dossiers*:

Les fichiers crées dans ce dossier portent le gid du dossier (et pas du processus qui les a créé).