

# Architecture et Systèmes

Stefan Schwoon

Cours L3, 2022/2023, ENS Paris-Saclay

# Utilisateurs dans Unix

---

Utilisateurs :

personnes naturelles

utilisateurs virtuels (admin, “daemons”)

Gestion :

locale (`/etc/passwd`) dans Unix

à distance (serveur LDAP), `getent passwd`

⇒ On s’intéressera aux relations avec d’autres concepts  
(processus, fichiers, ...)

# Identifiants utilisateur et groupe

---

Un utilisateur ...

possède un identifiant numérique (**uid**) ;

appartient à un groupe *primaire* ;

appartient à quelques groupes *supplémentaires*.

commande utile : **id**

Groupes :

chaque groupe possède son identifiant numérique (**gid**)

gestion locale : voir `/etc/passwd` et `/etc/group`

# Utilisateurs et processus

---

Attributs d'un processus relatifs aux utilisateurs :

utilisateur réel (`getuid`)

utilisateur effectif (`geteuid`)

utilisateur sauvegardé (`getresuid`)

groupe réel et effectif (`getgid`, `getegid`)

groupes supplémentaires (`getgroups`)

Ces attributs déterminent des privilèges que possède un processus.

Ces identifiants sont hérités du processus père lors d'un `fork`.

# Utilisateur réel et effectif

---

**Utilisateur réel** : l'utilisateur pour qui le processus travaille

**Utilisateur effectif** : pris en compte pour les privilèges

Normalement, ces deux sont identiques !

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

utilisateur réel : celui qui souhaite changer son mot de passe

utilisateur effectif : root

# Gérer l'utilisateur effectif

---

`setuid` (voir `showids.c`)

modifier de façon permanente les privilèges d'un processus

l'utilisateur effective doit être 0

change les id réel, effectif et sauvegardé

`seteuid`

modifier de façon temporaire les privilèges d'un processus

on change l'utilisateur effectif seulement

si l'utilisateur effectif est 0, aucune restriction

sinon, on peut y mettre les valeurs réelles ou sauvegardées

# Utilisateurs/groupes et fichiers

---

Tout fichier 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 bits “ugo” : (user,group,others) × (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'analogie 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 signification légèrement différente sur les dossiers :

read: on peut obtenir la liste des fichiers du dossier

write: on peut modifier la liste (créer, renommer, 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.

Le bit setgid sur les *dossiers*:

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

# Entrées/sorties

---

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

Les entrées/sorties se font par des **fichiers**.

plus qu'une collection de données;

structure de données abstraite avec au moins deux opérations (**lecture**,  
d'**écriture**)

# Exemples

---

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

fichiers sur un disque dur ;

une zone de mémoire temporaire, p.ex. terminal, tube ;

représentation des données dans le noyau (`/proc`) ;

les connections réseau.

Les accès se font uniformément par les mêmes appels système, mais selon le type de fichier le noyau renvoie l'appel à un **pilote** pour réaliser l'opération.

# 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

# Systeme de fichiers

---

Unix gère un **systeme de fichiers** pour le stockage pérenne des données.

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 acceptés par toutes les appels système qui gèrent les fichiers.

# Inœuds

---

Un fichier consiste des données, et on y associe certains meta-données (nom du fichier, propriétaire, droits d'accès etc).

Les **inœuds** sont une structure de données pour stocker ces méta-données. Une partie d'un disque dur leur est réservée.

Parmi les données stockés dans un inœud, il y a le type de fichier, propriétaire, groupe, droits d'accès, nombre de pointeurs vers cet inœud, les blocs 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 référence vers un inœud avec un nom.

Pour la plupart des fichiers normaux, cette relation est un-à-un.  
Par contre, les dossiers typiquement possèdent plusieurs références.

`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 ses meta-données) est libéré lorsqu'on supprime son dernier lien (fonction `unlink` dans C).

# Liens durs et faibles

---

Unix connaît deux types de **liens**, tous les deux gérés par `ln`.

**Lien dur** : `ln foo bar` crée un nouveau fichier `bar` avec le même inœud que `foo`.

**Lien faible** : `ln -s foo bar` crée un fichier spécial `bar` qui ne contient qu'un pointeur vers un autre chemin (dans ce cas `foo`). Tout accès à `bar` est renvoyé vers ce chemin.