esiea

Programmation Système

LAB3 - Les fichiers

Table des matières

By Rezy

1 Objectifs

Ce *LAB* présente les notions de base de gestions de fichiers. Il est basé sur la mise en parallèle de notions relatives aux commandes Shell avec leur équivalent en langage **C**.

- Documents : *Linux – les commandes.pdf* et *Linux – Les systèmes de fichiers*.
- Notions abordées : l'inode, le lien physique, l'affectation des droits.
- Commandes et fichiers exploités : `ln`, `mkdir`, `rm`, `stat`,

- Travail à rendre : Vous devrez répondre directement à plusieurs questions au sein de ce document. Vous le copierez sur Moodle sous le nom : « LAB3_*NOM*.pdf » **ainsi que les trois .c**.

2 Travaux préparatoires

Pour vous familiariser à nouveau avec la manipulation de fichiers :

- lisez et effectuez les commandes décrites aux chapitres §3.8 à §3.10 du support *Linux – les commandes.pdf*;

3 Manipulation de fichiers

3.1 Introduction

Pour manipuler les fichiers, on distingue les fonctions système dites de « bas niveau » (propre à chaque système d'exploitation) et les fonctions de la bibliothèque C standard dite de « haut niveau » commune à tous les systèmes. Nous ne voyons ici que quelques fonctions de bas niveau. Les fonctions de « haut niveau » ont été abordées lors de l'étude du langage C.

- Lisez le document *Support Linux – Les systèmes de fichiers*.

Sur votre système, quelle est la partition montée sur /home ? Quelle est la taille d'un bloc pour cette partition ?

Avec la commande « df -h », on obtient ces informations :

Filesystem Size Used Avail Use% Mounted on

/dev/nvme0n1p2 451G 103G 326G 24% /

Donc la partition montée sur /home est celle de la racine / soit / dev/nvme0n1p2 (car le /home n'as pas de partition dédiée sur mon systeme)

Créez un fichier *new_file* contenant la date du jour (`echo 'date' > new_file`). Quels droits et quel numéro d'inode possède ce fichier ? Les droits attribués correspondent-ils à la valeur de l'`UMASK` ? **Justifiez**.

Pour obtenir ces informations, on lit cette ligne de la commande « ls -ialh » :

16947705 -rw-r--r-- 1 saml saml 5 Sep 12 10:55 new_file

Les droits sont en lecture pour tout le monde et en ecriture uniquement pour l'owner

Le numero d'inode est 16947705

Oui cela correspond bien a la valeur de umask car nos permissions par defaut sont 666 moins le 022 du umask soit des permissions de 644 correspondant aux permissions que l'ont observe sur notre fichier novuellement créé

Créez un nouveau répertoire *new_rep*. Quelle commande utilisez-vous pour créer une copie de *new_file* dans ce répertoire au moyen d'un **lien physique** (nom du fichier copié : *copy_new_file*) ? Visualisez les attributs de ces deux fichiers (`ls -lisa new_file new_rep/copy_new_file`). Quels sont leurs numéros d'inode ? Quel est le nombre de liens ? Quel est le nombre de blocs utilisés ?

Les commandes a executer sont :

mkdir new_rep

ln new_file new_rep/copy_new_file

Resultat :

16947705 4 -rw-r--r-- 2 saml saml 33 Sep 12 11:09 new_file

16947705 4 -rw-r--r-- 2 saml saml 33 Sep 12 11:09 new_rep/copy_new_file

On constate donc que les numero d'inode sont les memes, soit que le fichier pointe vers le meme endroit dans la memoire.

Supprimez le fichier d'origine *new_file*. Quels sont les changements pour *copy_new_file* ? À quel moment l'inode relatif à ce fichier sera rendu au système ? Dans ce cas, les données constitutives du fichier seront-elles supprimées du support ? Quelle commande permettrait de s'assurer que les données d'un fichier supprimé sont devenues illisibles sur le support (sous Linux ? Sous Windows ?).

Si on suprime l'inode d'origine new_file, rien de change pour notre copie sous forme de hardlink.

L'inode relatif a ce fichier sera donc rendu au systeme quand il ne sera plus referencé par aucun fichier.

Les données ne seront pas supprimées du support tant que l'inode est référencée. Quand tous les fichiers y faisant référence seront supprimés, les données ne seront pas automatiquement supprimées (sauf si c'est spécifié) non plus. Cependant, elles pourront etre supprimées si le systeme souhaite ecrire de nouvelles données a cet endroit.

Une commande pour s'assurer de la suppression des données est :

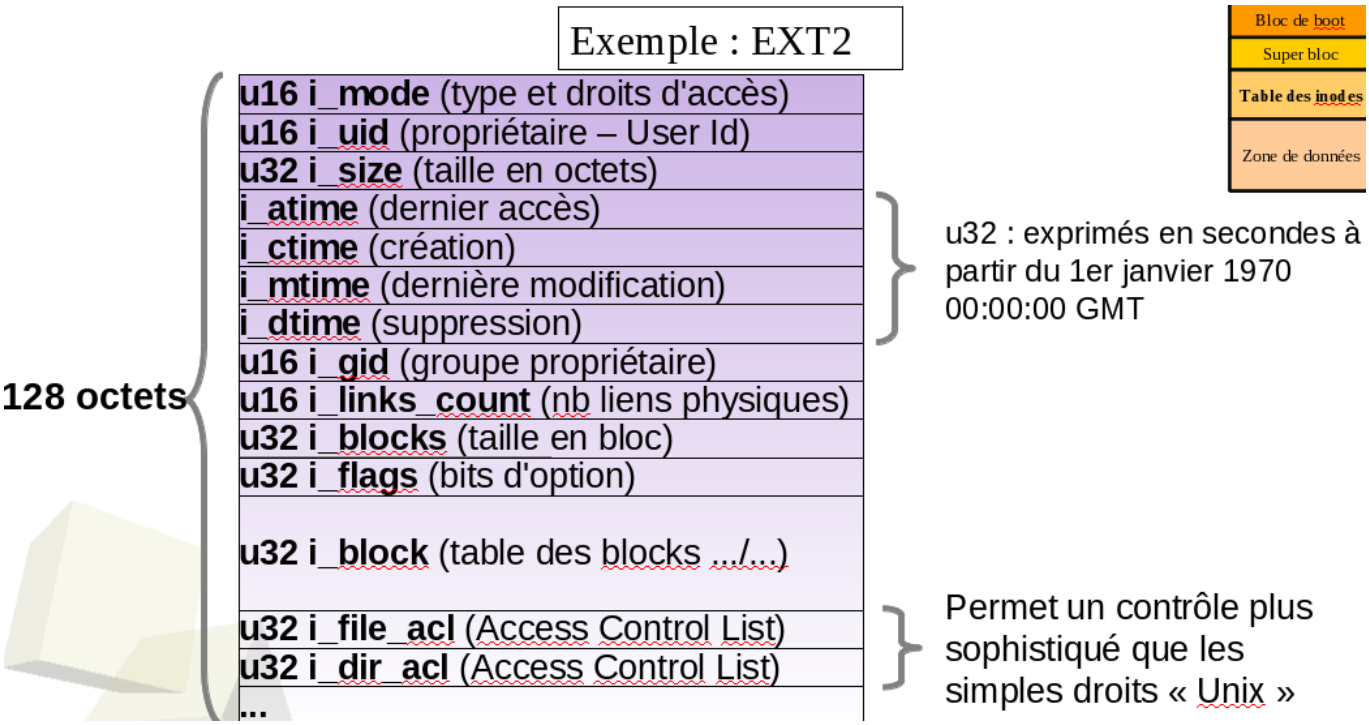
shred -u filename

Cette commande remplace des données du fichier avec des données aleatoires.

- Recréez *new_file* au moyen d'un lien physique

3.2 Accès aux informations de l'inode

Un inode est une entrée de la table des inodes. Cette entrée est constituée d'une structure nommée `stat` qui contient toutes les informations liées à l'inode (à l'exclusion des noms de fichier associés qui sont référencés dans les fichiers de type répertoire).



La table des blocs contient des pointeurs vers des zones du disque stockant le contenu du fichier.

En Shell :

La commande `stat` (`man stat`), du même nom que la structure, permet d'afficher les informations sur un inode (mise à par le contenu des blocs de données).

En C :

La fonction `stat` (`man 2 stat`) permet de récupérer les informations d'un inode dans une structure de type `stat`. La fonction `stat` est déclarée dans le fichier header `unistd.h`. La structure `stat` est déclarée dans le fichier header `sys/stat.h`.

```
#include <sys/stat.h>
#include <unistd.h>
int stat (char *nom, struct stat *buffer);
```

Explication des paramètres :

- `char *nom` : nom du fichier
- `struct stat *buffer` : pointeur vers une variable de type `stat` qui sera rempli lors de l'appel de la fonction.

Créez un programme en C qui récupère les informations de l'inode d'un fichier (ou d'un répertoire) que vous passez en paramètre. Pour voir ces informations, vous utiliserez l'outil de suivi des variables du débogueur DDD. Faites une copie d'écran montrant la variable `argv[1]` et les informations de l'inode correspondant au fichier passé en paramètre.

```
Breakpoint 1, main (argc=2, argv=0x7fffffffdb58) at part-3-2_DECASTRO_LUCAS.c:7
7       int main(int argc, char *argv[]) {
(gdb) print argv[1]
$1 = 0x7fffffffdfa3 "new_file"
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) cont
Continuing.
Num Inode : 16947705
Type de fichier : Fichier Normal
Taille : 33 octets
Nombre : 1
[Inferior 1 (process 11443) exited normally]
```

Intégrez à votre programme une fonction qui convertit le temps Unix en une date exploitable par « Mme Michu ». Faites une copie d'écran de votre programme qui affiche le numéro d'inode et la date de création (`ctime`) du fichier passé en paramètre **(rendez le code C sur Moodle dans un fichier appelé part-3-2_*NOM*.c)**.

```
Num Inode : 16947705
Type de fichier : Fichier Normal
Taille : 33 octets
Nombre : 1
Date de création : Tue Sep 12 11:32:46 2023
```

« `.` » et « `..` » sont des répertoires comme les autres.

Quels sont les numéros d'inode de « `/home` », « `/home/new_rep/..` », « `/` », « `/.` » et « `/..` » ?

Que constatez-vous ? Expliquez.

```
➜ lab3 ls -i /home/
14848789 new_rep  13107202 saml
➜ lab3 ls -i /home/new_rep/..
14848789 new_rep  13107202 saml
➜ lab3 ls -i /
     13 bin         1 dev  25690113 etc        14 lib         11 lost+found  27000833 mnt         1 proc        1
run   24117249 srv        1 tmp   7208961 var
28835841 boot   25952257 efi   13107201 home         15 lib64   20971521 media         21233665 opt
11010049 root       16 sbin        1 sys  11796481 usr
➜ lab3 ls -i /.
     13 bin         1 dev  25690113 etc        14 lib         11 lost+found  27000833 mnt         1 proc        1
run   24117249 srv        1 tmp   7208961 var
```

```
28835841 boot   25952257 efi    13107201 home        15 lib64   20971521 media       21233665 opt
11010049 root      16 sbin       1 sys  11796481 usr
➜ lab3 ls -i /..
     13 bin         1 dev  25690113 etc        14 lib        11 lost+found  27000833 mnt        1 proc       1
run   24117249 srv       1 tmp   7208961 var
28835841 boot   25952257 efi    13107201 home        15 lib64   20971521 media       21233665 opt
11010049 root      16 sbin       1 sys  11796481 usr
➜ lab3
```

Apres toutes ces commandes, nous constatons que les repertoires .. ont les memes numero d'inodes que leurs repertoires parents (s'ils en ont un, sinon il s'agit d'eux-memes). Il s'agit donc d'un lien physique vers leur repertoire parents et le repertoire « . » est un lien physique vers le repertoire lui-meme.

3.3 Fonctions de manipulation

La manipulation de fichiers consiste :

- soit à en modifier le contenu via un outil d'édition adapté au format du fichier (suite bureautique, éditeur de texte, logiciel de traitement d'images, etc.) ;
- soit à modifier les champs de leur inode (`chown`, `chmod`, `touch`, etc.) ;
- soit à modifier un fichier répertoire les référençant (`cp`, `ln`, `mv`, `rm`, etc.).

En C standard, les fonctions `fopen`, `fclose`, `fread`, `fwrite`, etc. permettent diverses manipulations indépendamment du système de fichiers (File System) sous-jacent.
En C système, les fonctions de manipulation permettent d'interagir directement sur les champs de l'inode ou dans les fichiers répertoire.
À titre d'exemple, nous allons voir comment modifier les droits d'un fichier.

En Shell :

Les commandes `chmod`, `chown` et `chgrp` permettent de modifier les attributs d'un fichier (droits et propriétaires).
Les utilisateurs et les groupes d'utilisateurs sur Linux sont en fait des nombres entiers (« User ID » et « Group ID ») que le système associe aux noms réels via les fichiers « `/etc/passwd` » pour l'UID et « `/etc/group` » pour le GID.

En Shell :

- Listez les groupes auxquels vous appartenez
- Créez deux fichiers « `new_file` » et « `new_file2` »
- Changez le groupe propriétaire du fichier « `new_file` » (un parmi ceux auxquels vous appartenez)

Affichez les droits de « `new_file` » et « `new_file2` »

```
➜ lab3 groups
sys rfkill wheel saml
➜ lab3 touch new_file new_file2
➜ lab3 chgrp sys new_file
```

```
➜ lab3 ls -lisa new_file new_file2
16947705 4 -rw-r--r-- 1 saml sys  33 Sep 12 11:52 new_file
16947710 0 -rw-r--r-- 1 saml saml  0 Sep 12 11:52 new_file2
```

En C :

1 Autorisation d'accès

Avant de tenter toute manipulation de fichier par un programme (ou processus), il est nécessaire de savoir si le processus en question a le droit de le faire. Rappel : sous Linux un processus hérite des droits de l'utilisateur qui le lance.
La fonction `access` (`man 2 access`) permet de vérifier quels droits un processus possède sur un fichier.

```
#include <unistd.h>
int access (char *nom, int droit);
```

Explication des paramètres :

- `char *nom` : nom du fichier ;
- `int droit` : prends les valeurs des 4 constantes suivantes (prédéfinies dans le fichier header) :
  - `F_OK` : le fichier existe
  - `R_OK` : je peux lire le fichier
  - `W_OK` : je peux écrire dans le fichier
  - `X_OK` : je peux exécuter le fichier
- la valeur de retour est nulle si OK, sinon elle vaut `-1` et la variable `errno` est renseignée.

2 Modification des attributs

Les fonctions `chmod()` et `fchmod()` permettent de modifier les droits d'un fichier :

- `chmod()` modifie les droits d'un fichier par son nom;
- `fchmod()` modifie les droits d'un fichier par son descripteur renvoyé par la fonction `open()` (`man 2 open`).

```
#include <sys/stat.h>
int chmod (char *nom, int droits);
int fchmod (int file, int droits);
```

Explication des paramètres :

- `char *nom` : nom du fichier ;
- `int file` : descripteur de fichier ouvert au préalable par la fonction `open()` ;
- `int droits` : droits attribués au fichier. Utilisez au choix une valeur octale ou une association de bits comme dans la structure `stat`.

Les fonctions `chown()` et `fchown()` permettent de modifier le « propriétaire » et le « groupe propriétaire » d'un fichier (`uid` et `gid`)

- `chown()` : modifie `uid` et `gid` d'un fichier par son nom;
- `fchown()` : modifie `uid` et `gid` d'un fichier par son descripteur renvoyé par la fonction `open()`.

```
#include <sys/unistd.h>
int chown (char *nom, int uid, int gid);
int fchown (int file, int uid, int gid);
```

Explication des paramètres :

- `char *nom` : nom du fichier;
- `int file` : descripteur de fichier ouvert au préalable par la fonction `open()`;
- `int uid` : numéro du futur propriétaire du fichier;
- `int gid` : numéro du futur groupe propriétaire du fichier.

Remarque : Si on ne désire modifier qu'un des deux paramètres (`uid` ou `gid`), il suffit de renseigner le

deuxième avec la valeur spéciale `-1`.

- Réalisez un programme en C qui affiche les droits que votre processus possède sur un fichier (ou un répertoire) que vous passez en paramètre **(rendez le code C sur Moodle dans un fichier appelé part-3-3_*NOM*.c)**.
- Copiez ici une capture d'écran de son exécution.

```
Permissions:
Owner read: Yes
Owner write: Yes
Owner execute: No

Group read: Yes
Group write: No
Group execute: No

Others read: Yes
Others write: No
Others execute: No
```

La suite de ce document fournit les fonctions système permettant toutes les manipulations sur les fichiers. La commande « man 3 nom_fonction » fournit l'aide nécessaire à l'exploitation de ces fonctions.

3 Création/suppression

- `int creat(char *nom, int droits)` : création d'un fichier « classique ». On préfère de plus en plus utiliser la fonction `open()` décrite dans la rubrique suivante.
- `int mkdir(char *nom, int droits)` : création d'un fichier « répertoire ».
- `int mkfifo(char *nom, int droits)` : création d'un fichier « pipe ».
- `int mknod(char *nom, int droits, int dev)` : création d'un fichier périphérique.
- `int link(char *source, char *cible)` : création d'un lien physique vers un fichier.
- `int symlink(char *source, char *cible)` : création d'un lien symbolique vers un fichier.
- `int unlink(char *nom)` : décrémente le nombre de liens physiques d'un fichier, si le compteur de lien atteint zéro, l'inode est rendu au système.
- `int rmdir(char *nom)` : supprime un répertoire à condition que ce dernier soit vide.

4 Accès au contenu

- `int open (char *nom, int mode[int droits])` : demande l'allocation d'une nouvelle entrée dans la table des fichiers ouverts du système. La fonction renvoie un entier « descripteur de fichier » pour le fichier dont le nom est passé en paramètre. Ce descripteur servira de référence ultérieure pour accéder au fichier et pour le fermer. Cette fonction permet l'ouverture d'un fichier existant ou la création d'un nouveau fichier (évite d'utiliser la fonction `creat()`). int mode peut prendre les valeurs des constantes prédéfinies suivantes :
  - `O_RDONLY` : ouverture en lecture seule;
  - `O_WRONLY` : ouverture en écriture seule;
  - `O_RDWR` : ouverture en lecture-écriture.

  Et facultativement des constantes prédéfinies suivantes :
  - `O_CREAT` : création du fichier s'il n'existe pas;
  - `O_TRUNC` : vidage préalable du fichier;
  - `O_APPEND` : écriture à la fin du fichier;

- `O_EXCL` : ne pas ouvrir le fichier s'il n'existe pas.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
void main(void) {
    int desc;
    if ((desc=open("test.txt", O_RDWR|O_CREAT|O_EXCL, 0777)) < 0)
        perror("erreur d'ouverture");
    else close(desc);
}
```

- `int close (int desc)` : ferme un fichier préalablement ouvert par open();
- `int read (int desc, char *buffer, int nb)` : lis un volume d'octets dans un fichier ouvert avec `open()`;
- `int write (int desc, char *buffer, int nb)` : écris un volume d'octets dans un fichier ouvert avec `open()`;
- `off_t lseek (int desc, off_t offset, int whence)` : déplace le pointeur interne de lecture/écriture.

- **Copiez sur Moodle sous le nom « part-4_*NOM*.c »**, un programme en C qui crée un nouveau fichier dont le nom est passé en paramètre. Ce fichier doit être en « lecture seule » et uniquement pour son propriétaire. Ce fichier contient au format XML (cf. exemple ci-dessous) la date et l'heure actuelle, le nom de l'utilisateur, son UID et un mot de passe aléatoire de 10 caractères que vous générez.

Le format XML est un format texte utilisant des balises que vous définissez, par exemple :

```
<nouvel_usager>
    <jour>yyyyyyyy</jour>
    <nom>xxxxxxx</nom>
    etc.
</nouvel_usager>
```

Pour info, le HTML est un format XML dont les balises sont normalisées.

Vous devez respecter les règles suivantes quant au choix de vos balises : Elles sont en anglais. Elles sont préfixées par **vos** initiales (exemple : `<AB_date>` … `</AB_date>`).

- Lancez votre programme en tant qu'utilisateur sans privilège (non root). Faites une copie d'écran montrant les droits du fichier généré et son contenu.

```
➜ lab3 gcc part-4_DECASTRO_LUCAS.c -o part-4_DECASTRO_LUCAS.o
➜ lab3 ./part-4_DECASTRO_LUCAS.o log.txt
➜ lab3 ls -lisa log.txt
16947889 4 -r-------- 1 saml saml 162 Sep 12 14:04 log.txt
➜ lab3 cat log.txt
<new_user>
  <LDC_date>2023-09-12 14:04:48</LDC_date>
  <LDC_name>saml</LDC_name>
  <LDC_uid>1000</LDC_uid>
  <LDC_password>8wBW5VB7Hf</LDC_password>
</new_user>
```