# 

LAB3 - Les fichiers

**Programmation système**

**KHAMPHOUSONE Thierry – TC-44**

Table des matières

[1 - Objectifs 2](#__RefHeading__317_62609081)

[2 - Rappels et mise en place de l’environnement 2](#__RefHeading__742_1331557722)

[3 - Travaux préparatoires 2](#__RefHeading___Toc2787_1718237428)

[4 - Manipulation de fichiers 2](#__RefHeading__9_62609081)

[4.1 - Introduction 2](#__RefHeading__35634_1331557722)

[4.2 - Accès aux informations de l’inode 3](#__RefHeading__35636_1331557722)

[4.3 - fonctions de manipulation 5](#__RefHeading__35642_1331557722)

By Rexy



# 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 (et Scripts Shell) avec leur équivalent en langage C.

* Documents : Support « 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**.

# 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 » ;
* Lisez la rubrique « topic4 » du cours LPIC1V3 (Linux Professional Institute Certification) disponible sur Moodle (dans « accueil » + « espace commun » + « Certification Linux »). Effectuez plusieurs autotests des quatre premiers « topics ».

# Manipulation de fichiers

## 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 ?

A screenshot of text

Description automatically generated

- La partition montée sur /home est /dev/sda8/

- La taille d’un bloc pour cette partition est 4096

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 ?

A screenshot of a cell phone

Description automatically generated

Le fichier est un fichier ordinaire et possède les droits en lecture écriture pour User (owner), lecture pour group (group user) et lecture pour all (all user).

Le numéro d’inode est 1573031 pour le fichier new\_file.

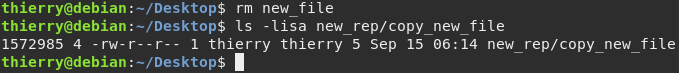
Créez un nouveau répertoire « new\_rep ». Quelle commande utilisez-vous pour créer une copie de « new\_file » dans ce répertoire au moyen d’un lien physique (nom du fichier copié : « copy\_new\_file ») ?  
Visualisez les attributs de ces deux fichiers (ls -lisa new\_file new\_rep/copy\_new\_file). Quels sont leurs numéros d’inode ? Quel est le nombre de liens ? Quel est le nombre de blocs utilisés ?

A close up of a logo

Description automatically generated

Création d’un lien physique, le fichier new\_file (ici inode : 1572985) en dehors du dossier new\_rep et le fichier au sein du dossier new\_rep possèdent le même numéro d’inode, le nombre de lien est de 2 et le nombre de bloc utilisé est de 4.

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 ?).



Le seul changement observable est que le nombre de lien est maintenant de 1. L’inode relatif à ce fichier sera rendu au système le moment où le nombre de lien est de 0, c’est-à-dire qu’il n’a plus de nom qui puisse se référer au numéro d’inode. Les données constitutives du fichier seront donc bien supprimées du support.

A picture containing drawing

Description automatically generated

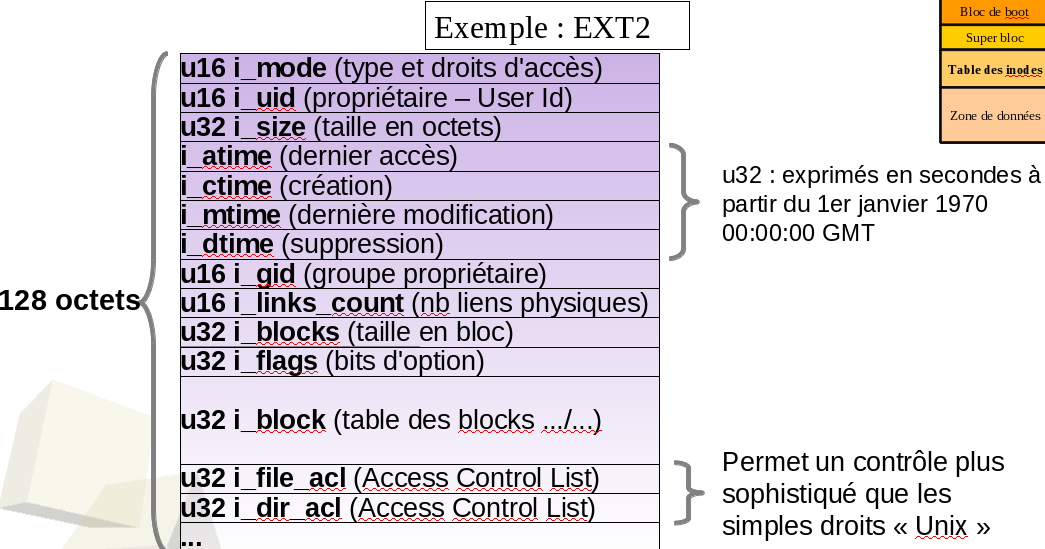
Cette commande me permet de voir si le contenu du fichier est encore accessible ou pas par un nom de fichier.

**Lien physique : Les fichiers peuvent avoir plusieurs noms différents et chemin différents mais accèdent à un même fichier car ils pointent sur le même numéro d’inode. Donc le fichier existera tant qu’il existera au moins un nom.**

* Recréez « new\_file » au moyen d’un lien physique

## 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 à l’appel de la fonction.

Créez un programme en C qui récupère les informations de l’inode d’un fichier (ou d’un fichier 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.



Pour utiliser ddd sur notre programme avec un argument en entrée.

A screenshot of a cell phone

Description automatically generated

On a bien les informations de l’inode 1572985 et le contenu de argv[1]

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.c)**.

**Le fichier a bien été déposé sur Moodle au nom de part-3-2.c**

A screenshot of a cell phone

Description automatically generated

« . » et « .. » sont des fichiers répertoire comme les autres. Quels sont les numéros d’inode des fichiers répertoire « /home/student », « /home/student/new\_rep/.. », « / », « /. » et « /.. ». Que constatez-vous. Expliquez.

A screenshot of a cell phone

Description automatically generated

On remarque que l’inode de la racine / est 2 et qu’on ne peut pas aller se placer avant la racine car ça sera encore la racine /.

On remarque également que les autres inodes débutent tous pas 1572 ou 1573.

## 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 (FS) sous-jacent.

En C-system, 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 »

A picture containing bird

Description automatically generated

Avant changement du groupe propriétaire du fichier new\_file

A screenshot of a cell phone

Description automatically generated

Après changement du groupe propriétaire du fichier new\_file

A close up of text on a black background

Description automatically generated

**en C :**

### 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.

### 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 les « uid/gid » d’un fichier par son nom ;
* fchown() : modifie les « uid/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 fichier répertoire) que vous passez en paramètre **(rendez le code C sur Moodle dans un fichier appelé part-3-3.c)**.
* Copiez ici une capture d’écran de son exécution.

A close up of text on a black background

Description automatically generated

**Application de mon programme sur un fichier créé en tant que super user en tant qu’user :**

**- sudoFile existe, il peut être lu, et non modifiable et non executable**

**Application de mon programme sur un fichier créé en tant qu’user en tant qu’user :**

- new\_file existe, il peut être lu et modifiable mais non executable.

**Le code C a été déposé sur le Moodle en tant que part-3-3.c**

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.

### 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.

### 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 « Prog\_LAB3\_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 l’utilisateur propriétaire. Ce fichier contient au format XML (cf. exemple ci-dessous) la date et l’heure actuelle, le nom de votre 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. 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.

J’impose 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.

A screenshot of a cell phone

Description automatically generated

Le fichier PigeonFile.txt possède bien le droit en lecture seule uniquement pour le propriétaire.

L’affichage de son contenu est bien affiché sur la sortie standard.

**Le fichier a été déposé sur Moodle au nom de : Prog\_LAB3\_Thierry\_Khamphousone.c**