LAB3 - Les fichiers

Table des matières

By R

2 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**.

3 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 » ;

4 Manipulation de fichiers

4.1 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 ?

```
Disk /dev/sda: 58,39 GiB, 62684954624 bytes, 122431552 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x080f141c

Device     Boot   Start       End   Sectors  Size Id Type
/dev/sda1  *       2048   1050623   1048576  512M  b W95 FAT32
/dev/sda2       1052670 122429439 121376770 57,9G  5 Extended
/dev/sda5       1052672 122429439 121376768 57,9G 83 Linux
srikanth@srikanth-VirtualBox:~$
```

La taille de ce bloc pour cette partition est de 57,9 BG

Sudo fdisk -l

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 ?

```
srikanth@srikanth-VirtualBox:~/Documents$ touch new_file
srikanth@srikanth-VirtualBox:~/Documents$ nano new_file
srikanth@srikanth-VirtualBox:~/Documents$ echo `date` > new_file
srikanth@srikanth-VirtualBox:~/Documents$ cat new_file
sam. 11 sept. 2021 17:57:02 CEST
srikanth@srikanth-VirtualBox:~/Documents$
```

```
srikanth@srikanth-VirtualBox:~/Documents$ ls -l
total 100
-rw-rw-r-- 1 srikanth srikanth  4688 sept. 10 21:09 erreur.txt
-rw-rw-r-- 1 srikanth srikanth     0 sept.  9 21:31 file
-rw-rw-r-- 1 srikanth srikanth     0 sept. 10 15:55 file1
-rw-r--r-- 1 srikanth srikanth    33 sept. 11 17:57 new_file
```

Les droits que le fichier possède.

Voici le numero d'inode du fichier new_file, le numero d'inode est 6296099

Le fichier possède des droit pour la lecture et lecture pour l'utilisateur, et l'ecriture pour pour le groupe et les autres, comme on peut le voir dans la capture au-dessus, rw-r—r--.

```
srikanth@srikanth-VirtualBox:~/Documents$ ls -i new_file
6296099 new_file
srikanth@srikanth-VirtualBox:~/Documents$
```

Le chmod et l'umask sont semblable mais l'umask fonctionne dans le sens contraire du chmod, par defaut l'umask est régler 002 ce qui correspond bien au droit sur le fichier new_file.

Comme on peut le voir dans la capture au-dessus, la valeur umask est 0022 soit rw-r-r qui correspondent au droit attribués lors de la création du 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 ?

on utilise la commande ln pour copier le fichier dans un autre répertoire

```
srikanth@srikanth-VirtualBox:~/Documents$ mkdir new_rep
srikanth@srikanth-VirtualBox:~/Documents$ ln new_file /home/srikanth/Documents/new_rep/copy_new_file
srikanth@srikanth-VirtualBox:~/Documents$ cd new_rep/
srikanth@srikanth-VirtualBox:~/Documents/new_rep$ ls -l
total 4
-rw-r--r-- 2 srikanth srikanth 33 sept. 11 17:57 copy_new_file
srikanth@srikanth-VirtualBox:~/Documents/new_rep$ ls -i copy_new_file
6296099 copy_new_file
srikanth@srikanth-VirtualBox:~/Documents/new_rep$
```

```
srikanth@srikanth-VirtualBox:~/Documents/new_rep$ cd -
/home/srikanth/Documents
srikanth@srikanth-VirtualBox:~/Documents$ ls -lisa new_file /home/srikanth/Documents/new_rep/copy_new_file
6296099 4 -rw-r--r-- 2 srikanth srikanth 33 sept. 11 17:57 /home/srikanth/Documents/new_rep/copy_new_file
6296099 4 -rw-r--r-- 2 srikanth srikanth 33 sept. 11 17:57 new_file
srikanth@srikanth-VirtualBox:~/Documents$
```

Leur numéros d'inodes sont les même et le nombre de lien et de 4

```
srikanth@srikanth-VirtualBox:~/Documents/new_rep$ stat new_file
  File: new_file
  Size: 33              Blocks: 8          IO Block: 4096   regular file
Device: 803h/2051d      Inode: 6296591     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/srikanth)   Gid: ( 1000/srikanth)
Access: 2021-09-12 10:09:27.739589846 +0200
Modify: 2021-09-12 10:10:06.321105221 +0200
Change: 2021-09-12 10:10:06.321105221 +0200
 Birth: 2021-09-12 10:09:27.739589846 +0200
srikanth@srikanth-VirtualBox:~/Documents/new_rep$
```

Le nombre de blocs et de 8 d'apres la capture

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 ?).

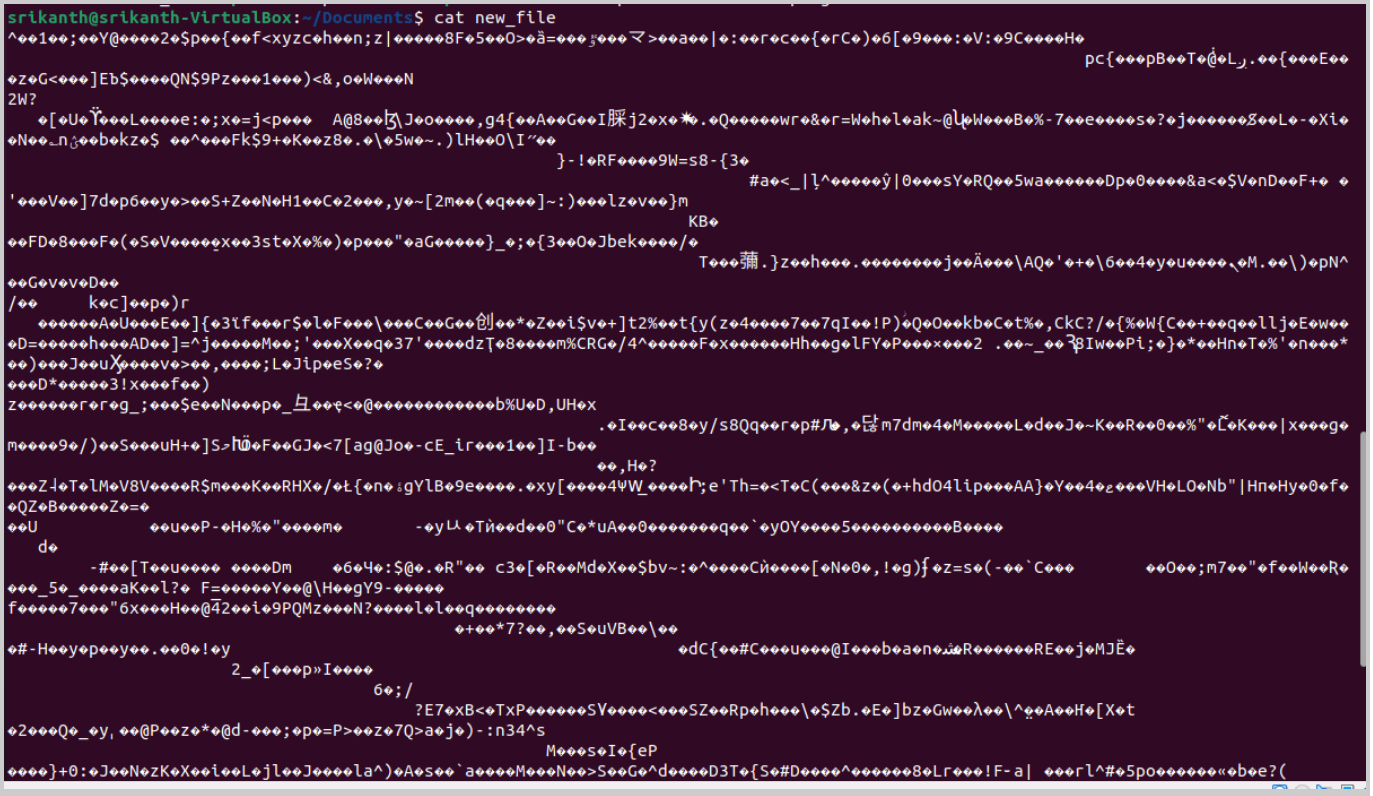
```
T/B◆dE◆hsrikanth@srikanth-VirtualBox:~/Documents$ ls
erreur.txt  file1     new_rep  part-3-2    part-3-2.c.save  part-3-3.c       partq   test1
file        new_file  part     part-3-2.c  part-3-3         part-3-3.c.save  prog.c
srikanth@srikanth-VirtualBox:~/Documents$ rm new_file
srikanth@srikanth-VirtualBox:~/Documents$
```

Quand on supprime le fichier new_file , il y a aucun changement pour le fichier copy_new_file, les données à l'intérieur de ce fichier sont toujours visibles.

```
srikanth@srikanth-VirtualBox:~/Documents$ ls
erreur.txt  file1     new_rep  part-3-2    part-3-2.c.save  part-3-3.c       partq
file        new_file  part     part-3-2.c  part-3-3         part-3-3.c.save  prog.c
srikanth@srikanth-VirtualBox:~/Documents$ shred -f new_file
srikanth@srikanth-VirtualBox:~/Documents$ ls
erreur.txt  file1     new_rep  part-3-2    part-3-2.c.save  part-3-3.c       partq
file        new_file  part     part-3-2.c  part-3-3         part-3-3.c.save  prog.c
srikanth@srikanth-VirtualBox:~/Documents$ sudo shred -f new_file
[sudo] password for srikanth:
srikanth@srikanth-VirtualBox:~/Documents$ ls
erreur.txt  file1     new_rep  part-3-2    part-3-2.c.save  part-3-3.c       partq
file        new_file  part     part-3-2.c  part-3-3         part-3-3.c.save  prog.c
```

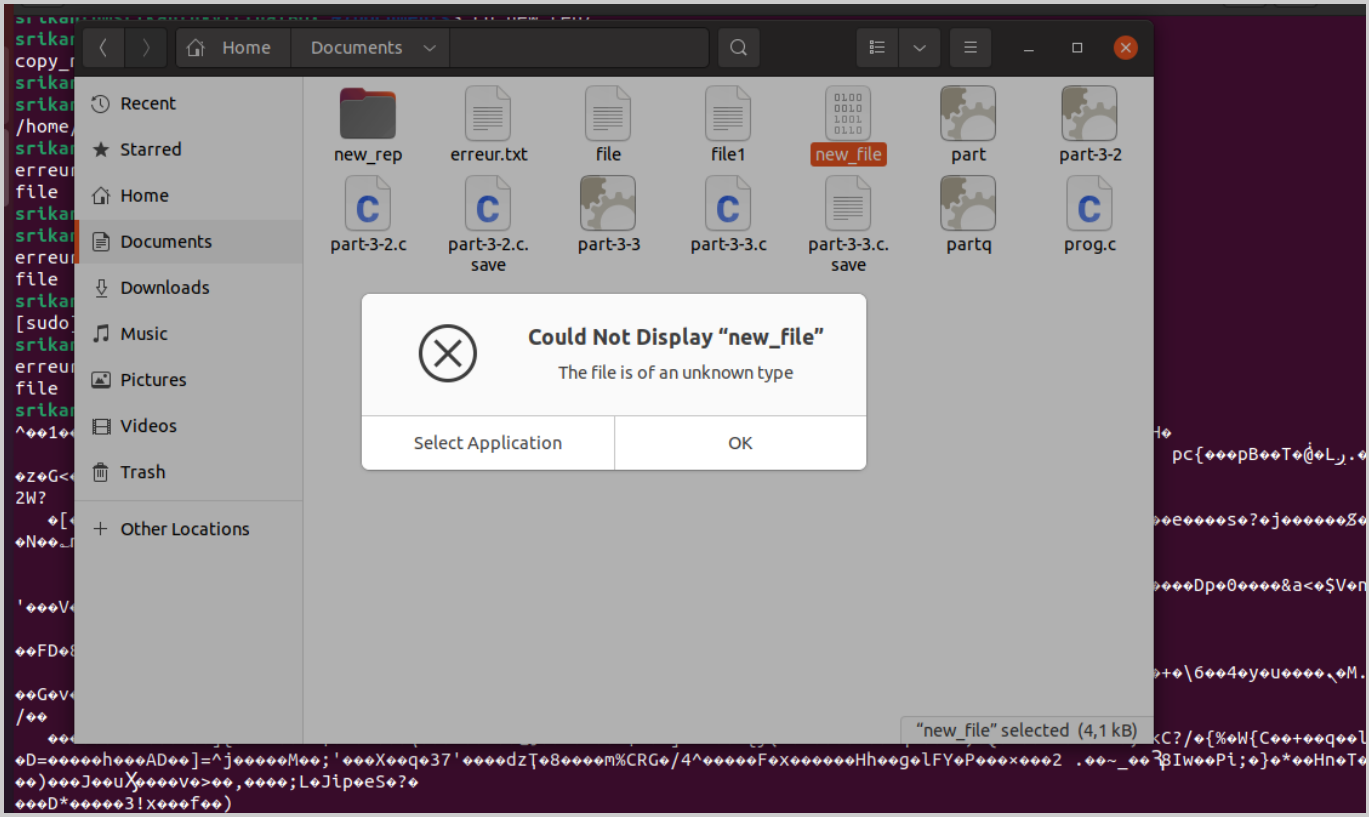
La commande qui permet de s'assurer que les donnes sont supprimées et de rendre illisibles les donnees est la commande shred

Pour supprimer et la rendre illisible un fichier sur windows il faut faire la commande sdelete. D'apres le document de microsoft : https://docs.microsoft.com/en-us/sysinternals/downloads/sdelete



Quand on affiche le fichier on voit bien que les donnes sont illisibles.

Encore une preuve on voit que c'est impossible à ouvrir le fichier

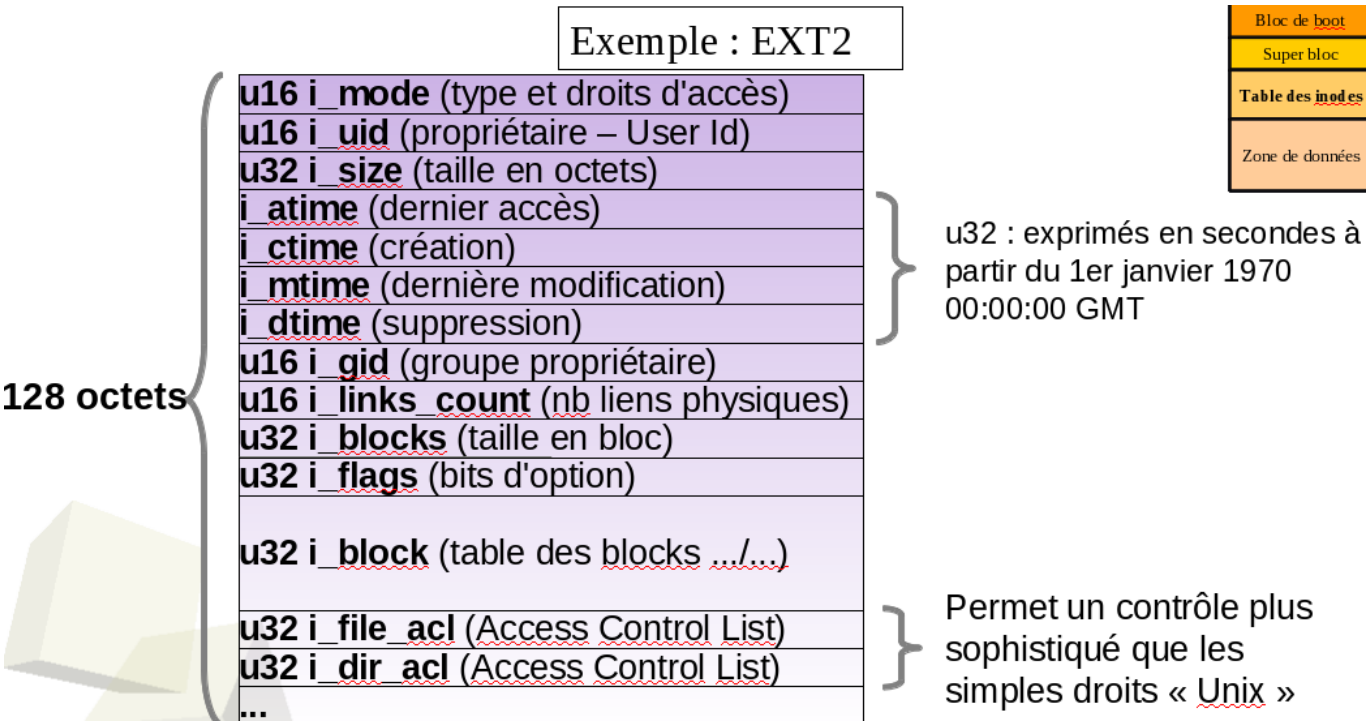


Même le fichier copy_new_file est aussi illisible.

- Recréez « new_file » au moyen d'un lien physique

4.2 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 :

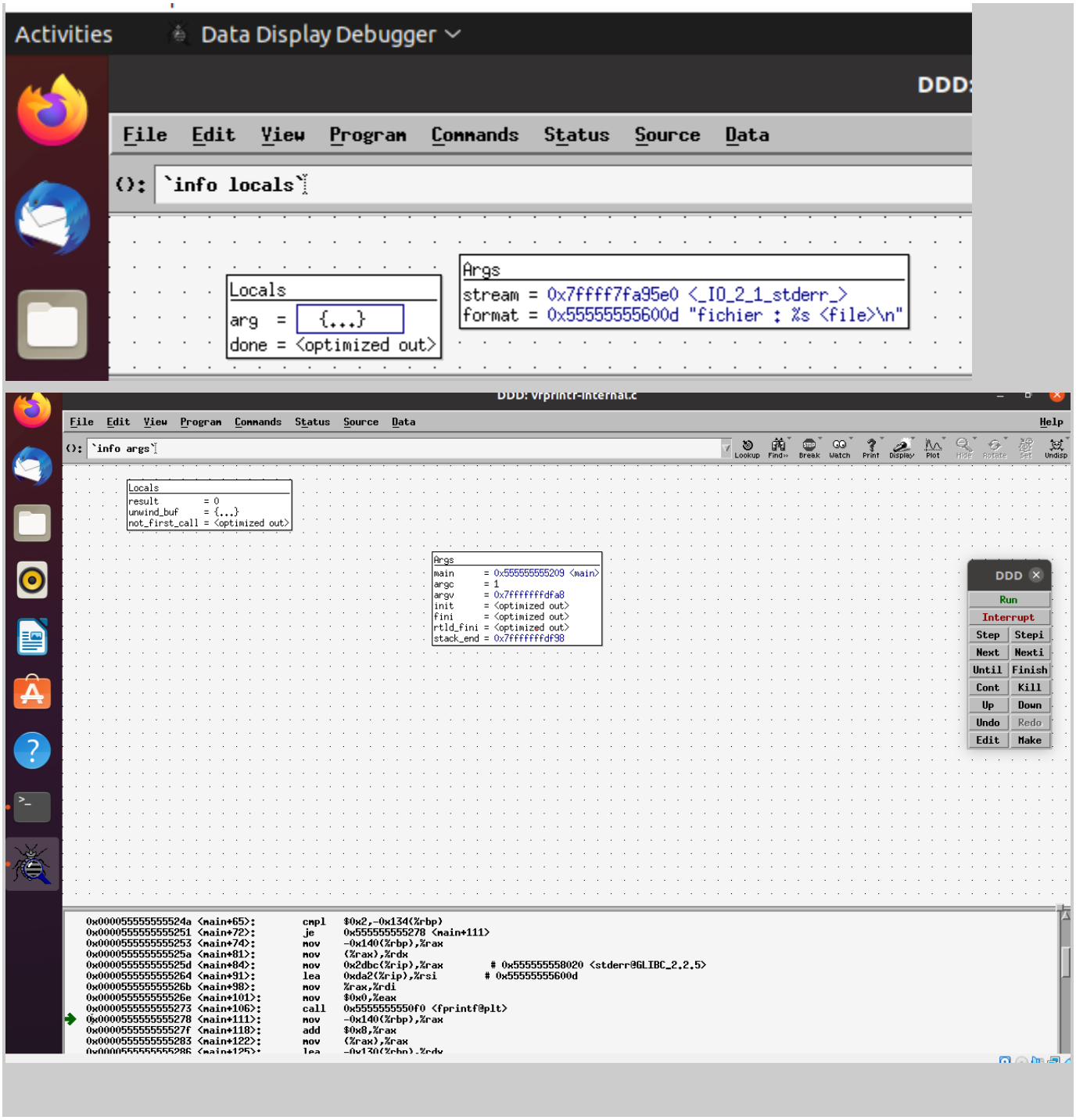
La fonction **stat** *(man 2 stat)* permet de récupérer les informations d'un inode dans une structure de type stat. La fonction stat est déclarée dans le fichier header « unistd.h ». La structure stat est déclarée dans le fichier header « sys/stat.h ».

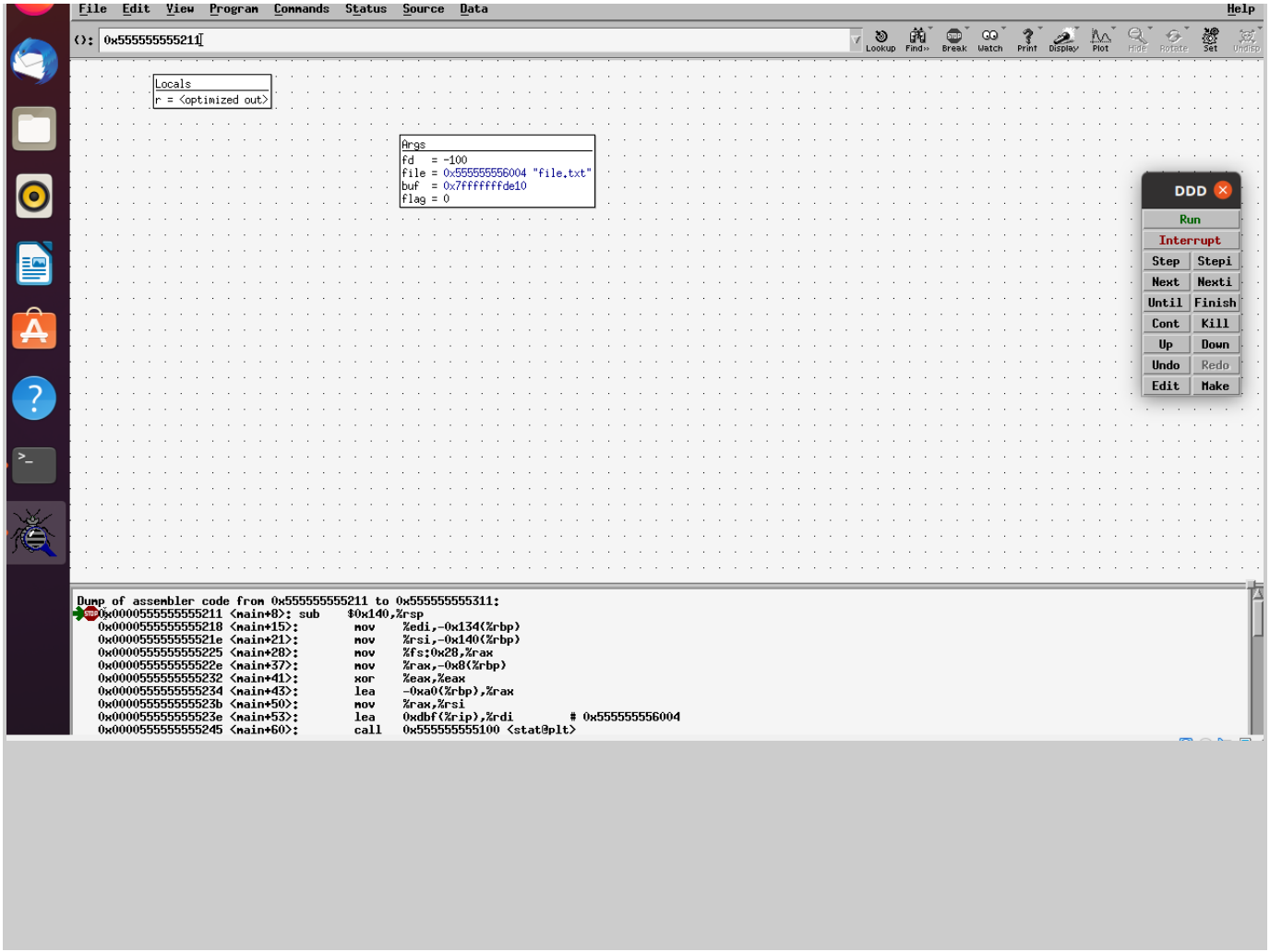
```
#include <sys/stat.h>
#include <unistd.h>
int stat (char *nom, struct stat *buffer)
```

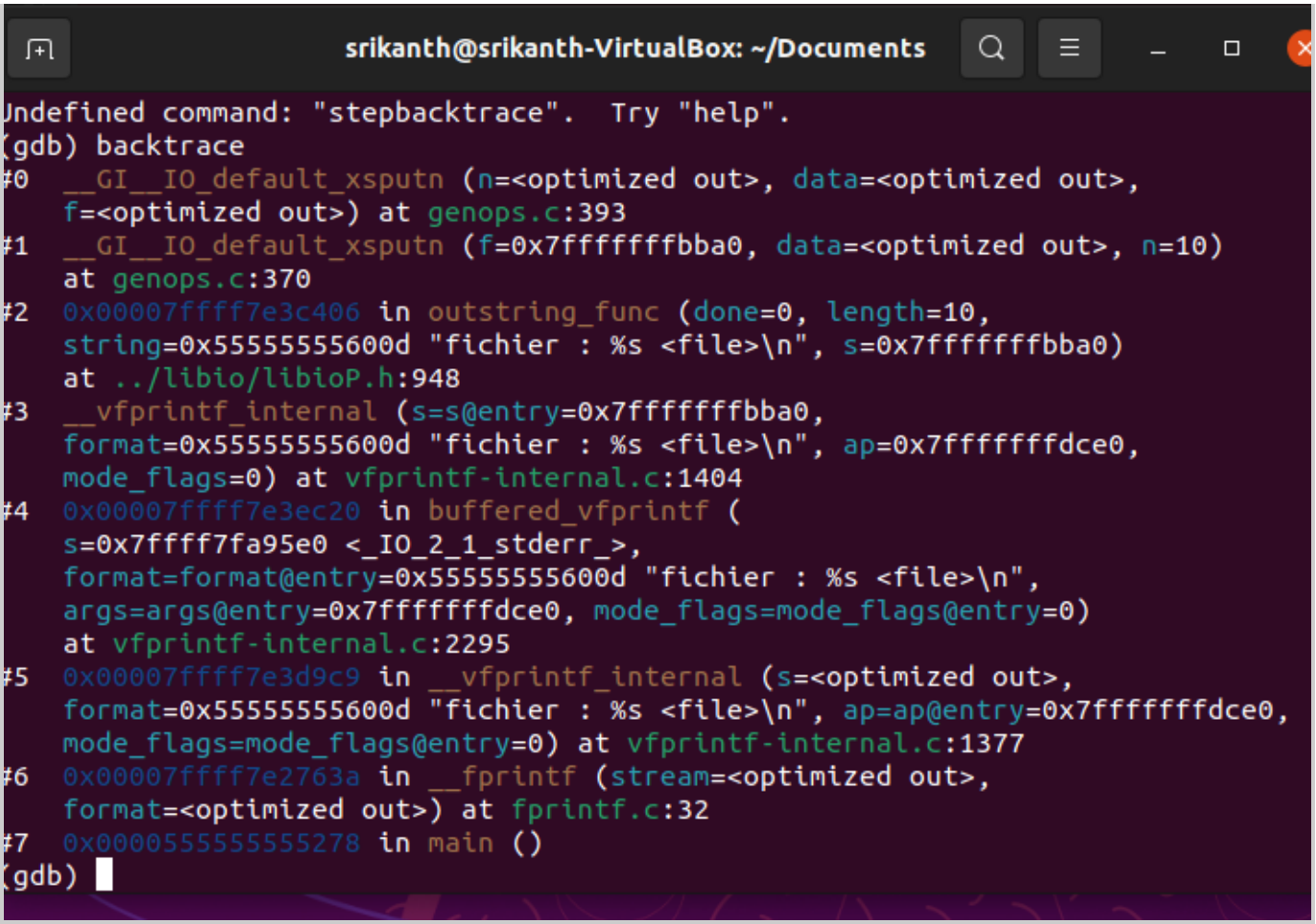
Explication des paramètres :

- char *nom : nom du fichier
- struct stat *buffer : pointeur vers une variable de type stat qui sera rempli 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.


Activities
Data Display Debugger
File Edit View Program Commands Status Source Data
(): `info locals`
Locals
arg = {...}
done = <optimized out>
Args
stream = 0x7ffff7fa95e0 <_IO_2_1_stderr_>
format = 0x55555555600d "fichier : %s <file>\n"
DDD: Vfprintf-internal.c
File Edit View Program Commands Status Source Data Help
(): `info args`
Locals
result = 0
unwind_buf = {...}
not_first_call = <optimized out>
Args
main = 0x555555555209 <main>
argc = 1
argv = 0x7fffffffdfa8
init = <optimized out>
fini = <optimized out>
rtld_fini = <optimized out>
stack_end = 0x7fffffffdf98
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
0x000055555555524a <main+65>: cmpl $0x2,-0x134(%rbp)
0x0000555555555251 <main+72>: je 0x555555555278 <main+111>
0x0000555555555253 <main+74>: mov -0x140(%rbp),%rax
0x000055555555525a <main+81>: mov (%rax),%rdx
0x000055555555525d <main+84>: mov 0x2dbc(%rip),%rax # 0x555555558020 <stderr@GLIBC_2.2.5>
0x0000555555555264 <main+91>: lea 0xda2(%rip),%rsi # 0x55555555600d
0x000055555555526b <main+98>: mov %rax,%rdi
0x000055555555526e <main+101>: mov $0x0,%eax
0x0000555555555273 <main+106>: call 0x5555555550f0 <fprintf@plt>
0x0000555555555278 <main+111>: mov -0x140(%rbp),%rax
0x000055555555527f <main+118>: add $0x8,%rax
0x0000555555555283 <main+122>: mov (%rax),%rax


File Edit View Program Commands Status Source Data Help
(): 0x555555555211
Lookup Find» Break Watch Print Display Plot Hide Rotate Set Undisp
Locals
r = <optimized out>
Args
fd = -100
file = 0x555555556004 "file.txt"
buf = 0x7fffffffde10
flag = 0
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Dump of assembler code from 0x555555555211 to 0x555555555311:
0x0000555555555211 <main+8>: sub $0x140,%rsp
0x0000555555555218 <main+15>: mov %edi,-0x134(%rbp)
0x000055555555521e <main+21>: mov %rsi,-0x140(%rbp)
0x0000555555555225 <main+28>: mov %fs:0x28,%rax
0x000055555555522e <main+37>: mov %rax,-0x8(%rbp)
0x0000555555555232 <main+41>: xor %eax,%eax
0x0000555555555234 <main+43>: lea -0xa0(%rbp),%rax
0x000055555555523b <main+50>: mov %rax,%rsi
0x000055555555523e <main+53>: lea 0xdbf(%rip),%rdi # 0x555555556004
0x0000555555555245 <main+60>: call 0x555555555100 <stat@plt>



En essayant de le debugger avec gdb on obtient ce résultat. (Capture au-dessus).

```
srikanth@srikanth-VirtualBox:~/Documents$ gcc part-3-2.c -o part
srikanth@srikanth-VirtualBox:~/Documents$ ./part file
 6294918
srikanth@srikanth-VirtualBox:~/Documents$ ./part new_rep/
 6296082
srikanth@srikanth-VirtualBox:~/Documents$
```

Les inodes du fichier file et du dossier new_rep

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)**.

```
srikanth@srikanth-VirtualBox:~/Documents$ ls
file  file1  part-3-2  part-3-2.c
srikanth@srikanth-VirtualBox:~/Documents$
```

```
srikanth@srikanth-VirtualBox:~/Documents$ ./part file
 6294918
  creation du fichier:  Fri Sep 10 15:55:59 2021
srikanth@srikanth-VirtualBox:~/Documents$
```

Pour le faire le code, j'ai regardé un tutoriel sur YouTube à l'adresse suivante : https://www.youtube.com/watch?v=IyA1YY21NqM&ab_channel=AsimCode

« . » 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.

Inode des fichiers répertoire « /home/student »

```
srikanth@srikanth-VirtualBox:~$ ls -i /home/srikanth
6291491 Desktop    6291492 Downloads  6295966 new_rep   6291494 Public     6291498 Videos
6291495 Documents  6291496 Music      6291497 Pictures  6291493 Templates
srikanth@srikanth-VirtualBox:~$
```

Inode des fichiers répertoire « /home/student/new_rep/.. »

```
srikanth@srikanth-VirtualBox:~$ ls -i /home/srikanth/new_rep/..
6291491 Desktop    6291492 Downloads  6295966 new_rep   6291494 Public     6291498 Videos
6291495 Documents  6291496 Music      6291497 Pictures  6291493 Templates
srikanth@srikanth-VirtualBox:~$
```

Inode des fichiers répertoire « / »

```
srikanth@srikanth-VirtualBox:~$ ls -i /
     13 bin     6291457 home          11 lost+found  2752513 root       12 swapfile
 262145 boot         14 lib     1703937 media             1 run         1 sys
3801089 cdrom        15 lib32   4456449 mnt              18 sbin   9437185 tmp
      1 dev          16 lib64   7077889 opt          7995393 snap   9699329 usr
1572865 etc          17 libx32        1 proc          786433 srv    2621441 var
srikanth@srikanth-VirtualBox:~$
```

Inode des fichiers répertoire « /. »

```
srikanth@srikanth-VirtualBox:~$ ls -i /.
     13 bin     6291457 home          11 lost+found  2752513 root       12 swapfile
 262145 boot         14 lib     1703937 media             1 run         1 sys
3801089 cdrom        15 lib32   4456449 mnt              18 sbin   9437185 tmp
      1 dev          16 lib64   7077889 opt          7995393 snap   9699329 usr
1572865 etc          17 libx32        1 proc          786433 srv    2621441 var
srikanth@srikanth-VirtualBox:~$
```

Inode des fichiers répertoire « /.. »

```
srikanth@srikanth-VirtualBox:~$ ls -i /..
     13 bin     6291457 home          11 lost+found  2752513 root       12 swapfile
 262145 boot         14 lib     1703937 media             1 run         1 sys
3801089 cdrom        15 lib32   4456449 mnt              18 sbin   9437185 tmp
      1 dev          16 lib64   7077889 opt          7995393 snap   9699329 usr
1572865 etc          17 libx32        1 proc          786433 srv    2621441 var
srikanth@srikanth-VirtualBox:~$
```

Les Inode des fichiers répertoire « /home/student » et « /home/student/new_rep/.. » : leurs numéros d'inode se succèdent.

Alors que dans Les Inode des fichiers répertoire « / », « /. » et « /.. » : leurs numéros d'inodes sont désordonnées ne se succèdent pas.

On remarque aussi que « . » et « .. » ont le même inode., ca signifie qu'ils sont créés séparément.

4.3 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 (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 »

```
srikanth@srikanth-VirtualBox:~$ ls -l
total 8404
-rwxrwxr-x 1 srikanth srikanth   17040 sept.  7 10:15 a.out
-rw------- 1 srikanth srikanth 8536064 sept.  5 14:07 core
drwxr-xr-x 2 srikanth srikanth    4096 août  31 09:20 Desktop
drwxr-xr-x 3 srikanth srikanth    4096 sept.  6 23:17 Documents
drwxr-xr-x 3 srikanth srikanth    4096 sept.  6 21:39 Downloads
-rw-r--r-- 1 srikanth srikanth    1087 août  31 10:54 etc
drwxr-xr-x 2 srikanth srikanth    4096 août  31 09:20 Music
-rw------- 1 srikanth srikanth       0 sept.  7 11:02 new_file
-rw-rw-r-- 1 srikanth srikanth       0 sept.  7 11:02 new_file2
drwxr-xr-x 2 srikanth srikanth    4096 août  31 16:02 Pictures
-rw-rw-r-- 1 srikanth srikanth       0 sept.  7 10:43 prog2.c
-rw-rw-r-- 1 srikanth srikanth     644 sept.  7 10:43 prog.c
drwxr-xr-x 2 srikanth srikanth    4096 août  31 09:20 Public
drwxr-xr-x 3 srikanth srikanth    4096 sept.  6 14:18 snap
drwxr-xr-x 2 srikanth srikanth    4096 août  31 09:20 Templates
d---r-x--- 2 srikanth srikanth    4096 août  31 11:10 tmp
drwxr-xr-x 2 srikanth srikanth    4096 août  31 09:20 Videos
srikanth@srikanth-VirtualBox:~$ chown srikanth2 new_file
chown: changing ownership of 'new_file': Operation not permitted
srikanth@srikanth-VirtualBox:~$ chown srikanth2 new_file
chown: changing ownership of 'new_file': Operation not permitted
srikanth@srikanth-VirtualBox:~$ sudo chown srikanth2 new_file
srikanth@srikanth-VirtualBox:~$ ls -l new_file
-rw------- 1 srikanth2 srikanth 0 sept.  7 11:02 new_file
srikanth@srikanth-VirtualBox:~$
```

On a bien effectué le changement de propriétaire sur srikanth2.

```
srikanth@srikanth-VirtualBox:~$ ls -l new_file
-rw------- 1 srikanth2 srikanth 0 sept.  7 11:02 new_file
srikanth@srikanth-VirtualBox:~$ ls -l new_file2
-rw-rw-r-- 1 srikanth srikanth 0 sept.  7 11:02 new_file2
srikanth@srikanth-VirtualBox:~$
```

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 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.

```
srikanth@srikanth-VirtualBox:~/Documents$ ./part part-3-3.c
les droit du fichiers
-rw-rw-r-- 1 srikanth srikanth 474 sept. 11 17:17 part-3-3.c
srikanth@srikanth-VirtualBox:~/Documents$ gcc part-3-3.c -o part
srikanth@srikanth-VirtualBox:~/Documents$ ./part file
les droit du fichiers
-rw-rw-r-- 1 srikanth srikanth 0 sept.  9 21:31 file
srikanth@srikanth-VirtualBox:~/Documents$
```

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 « 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.

```
@srikanth-VirtualBox:~$ nano  prog_LAB3_collaty.c
srikanth@srikanth-VirtualBox:~$ gcc prog_LAB3_collaty.c -o prog
srikanth@srikanth-VirtualBox:~$ ./prog prog.xml
srikanth
2021_09_12
12
14:46
1000
ohnu1aiChe
-rw-rw-r-- 1 srikanth srikanth 752 sept. 12 14:46 prog_LAB3_collaty.c
srikanth@srikanth-VirtualBox:~$
```

On dans la capture, on affiche bien le nom d'utilisateur, la date, le jour, l'heure actuelle, le numéro d'UID, un mot de passe aléatoire et les droits du fichier générer.

Contenue du fichier :

```
srikanth@srikanth-VirtualBox:~$ cat prog.xml
<sc_user>
<sc_date></sc_date><sc_days></sc_days<sc_clock>/sc_clock<sc_UID></sc_UID><sc_random-password></sc_random-password></sc_user>om-passw
ord></sc_user>
srikanth@srikanth-VirtualBox:~$
```

Code :

```
  GNU nano 5.4                                          prog_LAB3_collaty.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{

    FILE *file=NULL ;
if(argc < 2){
printf("mettre un argument  <file>");
exit(0);
}

    file = fopen(argv[1], "w");


if (file !=NULL)
{
fputs("<sc_user> \n",file);
system(" echo $USER ");
fputs("<sc_date>",file);
system("date '+%Y_%m_%d'\n");
fputs("</sc_date>",file);
fputs("<sc_days>",file);
system("date '+%d'\n");
fputs("</sc_days",file);
fputs("<sc_clock>",file);
system("date '+%R'\n");
fputs("/sc_clock",file);
fputs("<sc_UID>",file);
system("id -u");
fputs("</sc_UID>",file);
fputs("<sc_random-password>",file);
system("pwgen 10 1");
fputs("</sc_random-password>",file);
fputs("</sc_user>",file);


system("ls -l prog_LAB3_collaty.c");
fclose(file);
    }
    return 0;
}
```