

**I2181-B**  
**LINUX :**  
**APPELS SYSTÈME**

**IO**

# QUOI ?

- Appels système (*system calls* ou *syscalls*) :  
« En informatique, un appel système désigne le moment où un programme s'interrompt pour demander au système d'exploitation d'accomplir pour lui une certaine tâche. »
- En Linux (UNIX System V version)  
→ manuel : **man 7 standards**
- Programmes écrits en langage C



# system calls

The Linux kernel has code to do a lot of things

read from a hard drive

make network connections

create new process

kill process

change file permissions

keyboard drivers

your program doesn't know how to do those things



TCP? dude I have no idea how that works

NO I do not know how the ext4 filesystem is implemented I just want to read some files

programs ask Linux to do work for them using system calls



please write to this file

program

<switch to running kernel code>

done! I wrote 1097 bytes♥



Linux

<program resumes>

every program uses system calls



Python program

I use the 'open' syscall to open files



Java program

me too!

me three!



C program

and every system call has a number (eg chmod is #90)

So what's actually going on when you change a file's permissions is



program

run syscall #90 with these arguments

ok!



Linux

you can see which system calls a program is using with strace

```
$ strace ls /tmp
```

will show you every system call 'ls' uses! it's really fun!



strace is high overhead don't run it on your production database

# GÉNÉRALITÉS

- Manuel Linux :

tous les appels système: `man 2 syscalls`

appel système *name*: `man 2 name`

Thèmes : `open/close` – `read/write` – `fork` –  
`exec` – `pipe` – `signals` – `shared memory`  
– `semaphores` – `poll` – `sockets`

- **Vérifier systématiquement le retour des syscalls**  
**→ éviter un arrêt brutal**  
(utiliser les fonctions *check* du module *utils* fourni)



# GÉNÉRALITÉS

- On vous demande de travailler uniquement en **Linux**
- **Distribution** Linux Ubuntu  $\geq$  LTS 22.04

Après installation, commande shell :

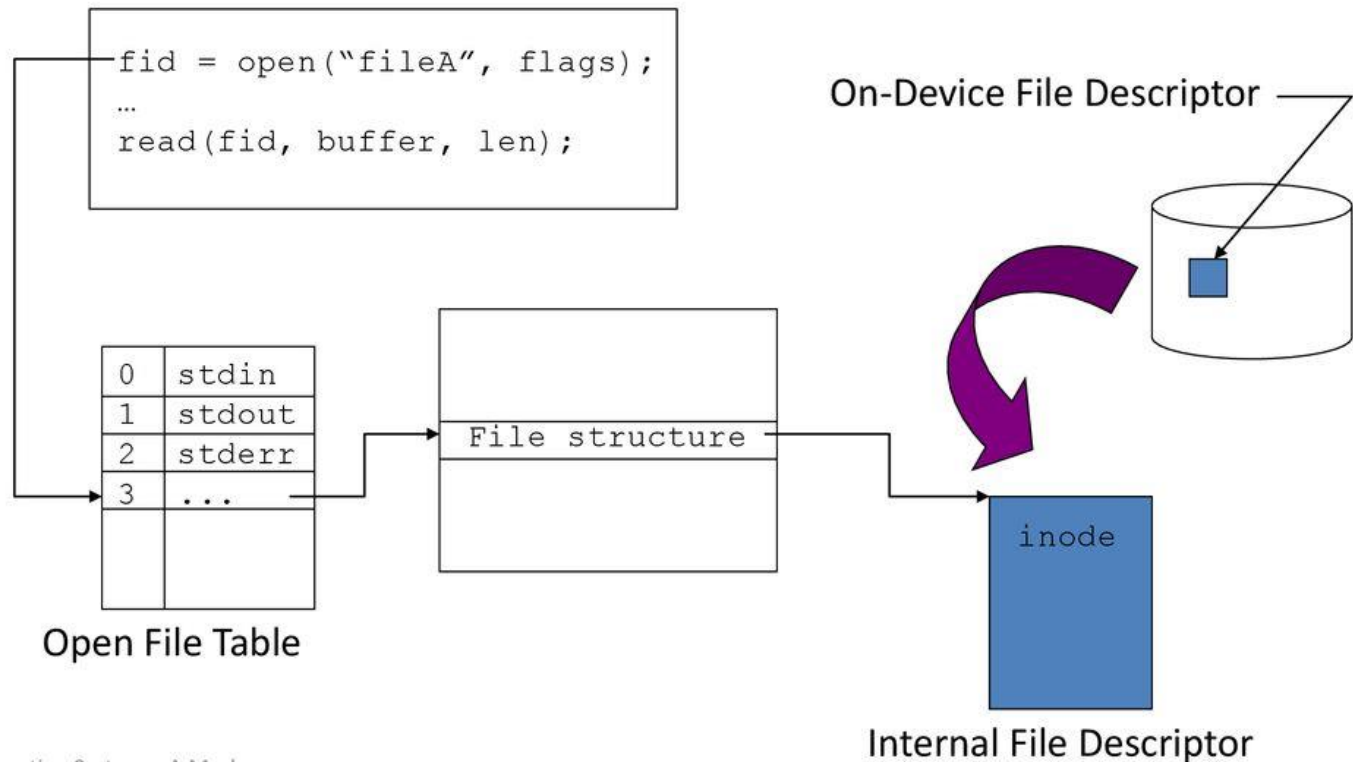
```
sudo apt install build-essential
```

- *makefile* **obligatoire**
- Compilation en C :

```
gcc -std=c11 -pedantic -Wall -Wvla -Werror  
-Wno-unused-variable  
-D_DEFAULT_SOURCE
```

# LES FICHIERS EN LINUX

## Opening a UNIX File





# OPEN

- Commande l'ouverture d'un fichier.

Création du fichier optionnelle.

- `int open (const char *pathname,  
                  int flags, mode_t mode)`

où `pathname` : absolute or relative name

`flags` : access mode | creation | file status

`mode` : définir droits d'accès (optionnel)

# HEADERS

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```



# PATHNAME

- Chaîne de caractères
- **Absolu** : commence par « / » → chemin complet à partir de la racine
- **Relatif** : ne commence pas par « / » → chemin à partir du répertoire de travail courant

# FLAGS

- Nombre entier
- Constantes séparées par opérateur | (« bitwise or »)
- Mode d'accès : O\_RDONLY ou O\_WRONLY ou O\_RDWR
- Création : O\_CREAT crée le fichier s'il n'existe pas
- Effacement du contenu : O\_TRUNC, le fichier est vidé (pas supprimé)
- ...



# MODE

- Type `mode_t` défini dans `<stat.h>`
- *bitwise or* de masques binaires de permissions, définis dans `<stat.h>`
- Soient « `r` » = `read` permission – valeur 4  
« `w` » = `write` permission – valeur 2  
« `x` » = `execute` permission – valeur 1

# MODE

- Chaque permission est octroyée pour le propriétaire (*user*), son groupe (*group*) et les autres (*others*)
- Mode = nb *octal* de 4 chiffres

1<sup>er</sup> chiffre = 0

2<sup>ième</sup> chiffre = somme des permissions « user »

3<sup>ième</sup> chiffre = somme des permissions « group »

4<sup>ième</sup> chiffre = somme des permissions « others »



# MODE

- Exemples :

0644 signifie

6 = 4 + 2 + 0, soit *rw\_* pour *user*

4 = 4 + 0 + 0, soit *r\_\_* pour *group*

4 = 4 + 0 + 0, soit *r\_\_* pour *others*

0750 signifie

7 = 4 + 2 + 1, soit *rwx* pour *user*

5 = 4 + 0 + 1, soit *r\_x* pour *group*

0 = 0 + 0 + 0, soit *\_\_\_* pour *others*

# MODE

- Mode a un effet uniquement lorsqu'un nouveau fichier est créé, p.ex. avec `O_CREAT`
- Doc : *man 2 chmod*



# RESULTAT

- *open* renvoie un nouveau *file descriptor* (*fd*) vers le fichier, ou -1 en cas d'erreur.
- Le *fd* identifie un *canal de communication* avec le fichier.
- Trois valeurs standards (<unistd.h>) :
  - fd = 0 pour le flux d'entrée standard (STDIN\_FILENO)
  - fd = 1 pour le flux de sortie standard (STDOUT\_FILENO)
  - fd = 2 pour le flux d'erreur standard (STDERR\_FILENO)

# RESULTAT

- En cas d'erreur, la variable `int errno` (définie dans `<errno.h>`) identifie l'erreur.
- Par exemple, `errno=EACCES` si problème d'autorisation d'accès à la ressource.
- Cf. `man errno`
- Cf. `man perror`



# CLOSE

- Commande la fermeture d'un *fd*.

Attention, une ressource est « libérée » lorsque tous les *fd* ouverts ont été fermés !

- « Libération » d'un *fd* aussi quand dernier processus qui utilise le fichier se termine, MAIS **mauvaise pratique !**

# CLOSE

```
#include <unistd.h>
```

```
int close(int fd)
```

- Renvoie 0 si ok, -1 si erreur.
- Par exemple, `errno=EIO` si erreur I/O.



# READ

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf,  
             size_t count)
```

- Lecture de max **count** bytes sur **fd** et stockage dans **\*buf**.
- Renvoie le nombre d'octets lus. Renvoie 0 si fin de fichier (EOF). Renvoie -1 si erreur.
- Pas de bufferisation

# READ

- `size_t` est un type entier non signé, plate-forme dépendant, permettant de représenter toute taille d'un objet stocké en mémoire.
- `ssize_t` équivaut à un `signed size_t`, permettant de recevoir un nombre négatif en cas d'erreur.



# WRITE

```
#include <unistd.h>
```

```
ssize_t write(int fd, const void *buf,  
              size_t count)
```

- Écriture de max **count** bytes sur **fd** à partir de **\*buf**.
- Renvoie le nombre d'octets écrits. Renvoie -1 si erreur.
- Pas de bufferisation

# READ/WRITE

- En cas d'erreur, `errno` identifie l'erreur.
- Par exemple, `errno=EIO` si erreur I/O.

D'autres erreurs seront passées en revue plus tard ...



# EXEMPLE

- Ouvrir fichier (nom = « test ») en le vidant
- Créer fichier si nécessaire
- Lire lignes sur entrée standard et écrire dans fichier
- Fermer le fichier
- Rouvrir le fichier
- Lire lignes dans fichier et écrire sur sortie standard

→ Voir archive *LAS\_01\_exemple et lastrace.sh*

*wget https://courslinux.ipl.be/~anthony.legrand/lastrace.sh*