

I2180
LINUX :
APPELS SYSTÈME
SEMAINE 1

ORGANISATION PRATIQUE

- 2 séances de 2 heures pendant 12 semaines (cf. Celcat)
- 9 semaines : capsules théoriques + exercices
- 3 semaines : projet – présence OBLIGATOIRE (30%, note reportée en septembre)
- Interro sur machine hors séances : mer. 18/03 (sem 7 ; 15%)
- Examen sur machine (entre 3h et 4h ; 55% juin ; 70% sept)

QUOI ?

- Appels systèmes :
« 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 version System V)
- Langage C

GÉNÉRALITÉS

- Manuel Linux : « `man 2 name` »
- Thèmes : `open/close` – `read/write` – `fork` –
`exec` – `pipe` – `signals` –
`shared memory` - `semaphores` – `poll`– `sockets`
- **Vérifier systématiquement le retour <>**
plantage brutal

GÉNÉRALITÉS

- On vous demande de travailler uniquement en Linux
- Distribution Linux Ubuntu \geq 16.04

Après installation, commande « console »

sudo apt-get install build-essentials

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

`gcc -std=c11 -pedantic -Wall -Wvla -Werror`

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 access mode (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

Chiffre 1 = 0 (pour l'instant)

Chiffre 2 = somme des permissions « user »

Chiffre 3 = somme des permissions « group »

Chiffre 4 = 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*

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 entrée standard (STDIN_FILENO)
 - fd = 1 pour sortie standard (STDOUT_FILENO)
 - fd = 2 pour erreur standard (STDERR_FILENO)

OPEN

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

CLOSE

- Commande la fermeture d'un *fd*.

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

« Libération » 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 nb d'octets lus. Renvoie 0 si fin de fichier (EOF). Renvoie -1 si erreur.

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 nb d'octets écrits. Renvoie `-1` si erreur.

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
- Revenir au début du fichier
- Lire lignes dans fichier et écrire sur sortie standard
- Voir fichier `ex1.c` + `Makefile`