

# Le système de gestion des fichiers, les entrées/sorties.

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,  
Centre de Mathématiques et Informatique,  
39, rue Joliot-Curie - F-13453 Marseille

19 octobre 2005

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

# Plan

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

# Les fichiers ordinaires

- Fichiers réguliers :  
stockage de l'information sur disque.
- Répertoires :  
organisation logique de l'information sur disque.
- Liens symboliques : partage des ressources.

```
[lsantoca@localhost lecture2]$ touch a b
[lsantoca@localhost lecture2]$ ln -s b c
[lsantoca@localhost lecture2]$ mkdir d
[lsantoca@localhost lecture2]$ ls -pdl a b c d
-rw-r--r--    1 lsantoca lsantoca      0 sep  3 15:09 a
-rw-r--r--    1 lsantoca lsantoca      0 sep  3 15:09 b
lrwxrwxrwx    1 lsantoca lsantoca      1 sep  3 15:09 c -> b
drwxr-xr-x    2 lsantoca lsantoca  4096 sep  3 15:10 d/
```

(Aujourd'hui, semaine prochaine)

# Les fichiers ordinaires

- Fichiers réguliers :  
stockage de l'information sur disque.
- Répertoires :  
organisation logique de l'information sur disque.
- Liens symboliques : partage des ressources.

```
[lsantoca@localhost lecture2]$ touch a b
[lsantoca@localhost lecture2]$ ln -s b c
[lsantoca@localhost lecture2]$ mkdir d
[lsantoca@localhost lecture2]$ ls -pdl a b c d
-rw-r--r--    1 lsantoca lsantoca          0 sep  3 15:09 a
-rw-r--r--    1 lsantoca lsantoca          0 sep  3 15:09 b
lrwxrwxrwx    1 lsantoca lsantoca          1 sep  3 15:09 c -> b
drwxr-xr-x    2 lsantoca lsantoca       4096 sep  3 15:10 d/
```

(Aujourd'hui, semaine prochaine)

# Tubes (pipes)

But : communication entre processus.

- tube anonyme :  
communication entre processus de la même famille,
- tube nommé :  
il lui correspond une référence (nom de fichier),  
accessible à tous les processus.

```
[lsantoca@localhost lecture2]$ ls -lp tube  
prw-r--r--  1 lsantoca lsantoca      0 sep  3 15:17 tube|
```

(Plus tard dans ce cours)

# Tubes (pipes)

But : communication entre processus.

- tube anonyme :  
communication entre processus de la même famille,
- tube nommé :  
il lui correspond une référence (nom de fichier),  
accessible à tous les processus.

```
[lsantoca@localhost lecture2]$ ls -lp tube  
prw-r--r--  1 lsantoca lsantoca      0 sep  3 15:17 tube|
```

(Plus tard dans ce cours)

## Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw----- 1 root    root      1,  0 jan  1 1970 /dev/ram0
crw-rw-rw- 1 root    root      5,  0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque :

1,5 : nombres majeurs, nombres du pilote de peripherique.

0 : nombre mineur, paramètre pour le pilote.



## Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw----- 1 root    root      1,  0 jan  1 1970 /dev/ram0
crw-rw-rw- 1 root    root      5,  0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque :

1,5 : nombres majeurs, nombres du pilote de peripherique.

0 : nombre mineur, paramètre pour le pilote.

## Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw----- 1 root root 1, 0 jan 1 1970 /dev/ram0
crw-rw-rw- 1 root root 5, 0 sep 3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque :

- 1,5 : nombres majeurs, nombres du pilote de peripherique.
- 0 : nombre mineur, paramètre pour le pilote.

# Sockets

But : communication sur un réseau

```
[lsantoca@localhost lecture2]$ ls -lp /tmp/agent.2416  
srwxrwxr-x    1 lsantoca lsantoca    0 déc 15  2003 /tmp/agent.2416
```

(Cours réseaux année prochaine)

# Plan

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

# Structure hiérarchique : première approximation

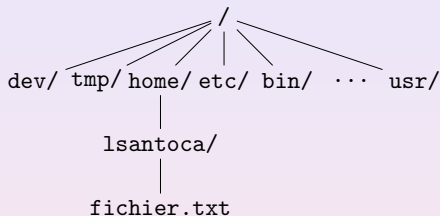
Les fichiers sont organisés en arbre :

- noeuds internes : répertoires,
- feuilles : fichiers réguliers, liens symboliques,  
tubes, sockets, fichiers dev.

Chaque fichier est dénoté par une référence (ou chemin):

- à partir de la racine /  
ex. : `/home/lsantoca/fichier.txt`
- à partir du répertoire courante \$PWD  
ex. : `fichier.txt` si `PWD=/home/lsantoca`

# Exemple



```
[lsantoca@localhost lsantoca]$ ls /home/lsantoca/fichier.txt
/home/lsantoca/fichier.txt
[lsantoca@localhost lsantoca]$ pwd
/home/lsantoca
[lsantoca@localhost lsantoca]$ ls fichier.txt
fichier.txt
```

## Structure hiérarchique : deuxième approximation

Les fichiers sont organisés en DAG

(directed acyclic graph, graphes orientés acycliques) :

- noeud interne : répertoire,
- arête : répertoire  $\rightarrow$  fichier appartenant à ce répertoire,
- feuille : fichier régulier sur disque, autre type de fichier.

Par conséquent :

- un fichier régulier peut avoir plusieurs parents,
- plusieurs chemins, liens (=noms de fichiers) pour le même fichier régulier.

Chaque sommet du graphe est déterminé par :

- son no. périphérique logique,
- son no. de i-noeuds.

## Structure hiérarchique : deuxième approximation

Les fichiers sont organisés en DAG

(directed acyclic graph, graphes orientés acycliques) :

- noeud interne : répertoire,
- arête : répertoire  $\rightarrow$  fichier appartenant à ce répertoire,
- feuille : fichier régulier sur disque, autre type de fichier.

Par conséquent :

- un fichier régulier peut avoir plusieurs parents,
- plusieurs chemins, liens (=noms de fichiers) pour le même fichier régulier.

Chaque sommet du graphe est déterminé par :

- son no. périphérique logique,
- son no. de i-noeuds.



## Structure hiérarchique : deuxième approximation

Les fichiers sont organisés en DAG

(directed acyclig graph, graphes orientés acycliques) :

- noeud interne : répertoire,
- arête : répertoire  $\rightarrow$  fichier appartenant à ce répertoire,
- feuille : fichier régulier sur disque, autre type de fichier.

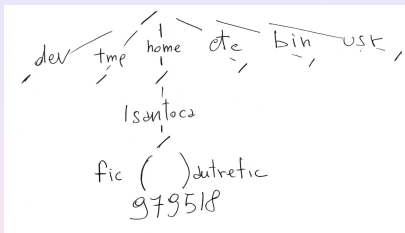
Par conséquent :

- un fichier régulier peut avoir plusieurs parents,
- plusieurs chemins, liens (=noms de fichiers) pour le même fichier régulier.

Chaque sommet du graphe est déterminé par :

- son no. périphérique logique,
- son no. de i-noeuds.

# Les liens durs



```

[lsantoca@localhost lecture2]$ touch fic
[lsantoca@localhost lecture2]$ ln fic autrefic
[lsantoca@localhost lecture2]$ stat fic
...
Device: 308h/776d      Inode: 979518      Links: 2
...
[lsantoca@localhost lecture2]$ stat autrefic
...
Device: 308h/776d      Inode: 979518      Links: 2
...

```

# Plan

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

# Disques physiques

Floppies, disques durs, mémoires SDA, etc.

Manipulation de :

cyindres, pistes, déplacement de la tête de lecture,

gérée par le pilote de périphérique.

# Disques physiques

Floppies, disques durs, mémoires SDA, etc.

Manipulation de :

    cylindres, pistes, déplacement de la tête de lecture,

gérée par le pilote de périphérique.

# Disques logiques :

Disque logique de fichiers ordinaires.

Disques « swap »:

contiennent les processus endormis, hors de mémoire vive.

Manipulation de :

noeuds d'information (i-node), blocs,  
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

# Disques logiques :

Disque logique de fichiers ordinaires.

Disques « swap »:

contiennent les processus endormis, hors de mémoire vive.

Manipulation de :

noeuds d'information (i-node), blocs,  
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

# Organisation d'un disque logique (SGF)

Un disque logique est :

- une suite de cases/cellules, pour y stocker l'information :  
les « blocs »,
- une suite d'objets pour organiser plusieurs blocs dans une seule entité :  
les « i-noeuds », les « fichiers ».
- informations à propos de lui même  
(espace libre sur disque, type, etc.) :  
le « super bloc ».



# Organisation d'un disque logique (SGF)

Un disque logique est :

- une suite de cases/cellules, pour y stocker l'information :  
les « blocs »,
- une suite d'objets pour organiser plusieurs blocs dans une seule entité :  
les « i-noeuds », les « fichiers ».
- informations à propos de lui-même  
(espace libre sur disque, type, etc.) :  
le « super bloc ».

# Organisation d'un disque logique (SGF)

Un disque logique est :

- une suite de cases/cellules, pour y stocker l'information :  
les « blocs »,
- une suite d'objets pour organiser plusieurs blocs dans une seule entité :  
les « i-noeuds », les « fichiers ».
- informations à propos de lui-même  
(espace libre sur disque, type, etc.) :  
le « super bloc ».

# Les SGF System V

bloc initialisation
super bloc
table des i-noeuds
⋮
blocs de données
⋮

utilisé au chargement du système

informations générales sur le SGF

chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires :  
 enregistrements taille fixe 16  
 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

# Les SGF System V

utilisé au chargement du système

<b>bloc initialisation</b>
super bloc
table des i-noeuds
⋮
blocs de données
⋮

informations générales sur le SGF

chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires :  
 enregistrements taille fixe 16  
 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

# Les SGF System V

bloc initialisation
<b>super bloc</b>
table des i-noeuds
⋮
blocs de données
⋮

Répertoires :  
enregistrements taille fixe 16  
caractères, liens 14 caractères

utilisé au chargement du système

informations générales sur le SGF

chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

chaque bloc : 512/1024/2048/4096 octets

# Les SGF System V

bloc initialisation
super bloc
<b>table des i-noeuds</b>
⋮
blocs de données
⋮

Répertoires :  
 enregistrements taille fixe 16  
 caractères, liens 14 caractères

utilisé au chargement du système

informations générales sur le SGF

chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

chaque bloc : 512/1024/2048/4096 octets

# Les SGF System V

bloc initialisation
super bloc
table des i-noeuds
⋮ <b>blocs de données</b> ⋮

Répertoires :  
 enregistrements taille fixe 16  
 caractères, liens 14 caractères

utilisé au chargement du système

informations générales sur le SGF

chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

chaque bloc : 512/1024/2048/4096 octets

# Les SGF System V

bloc initialisation
super bloc
table des i-noeuds
⋮
blocs de données
⋮

Répertoires :  
 enregistrements taille fixe 16  
 caractères, liens 14 caractères

utilisé au chargement du système

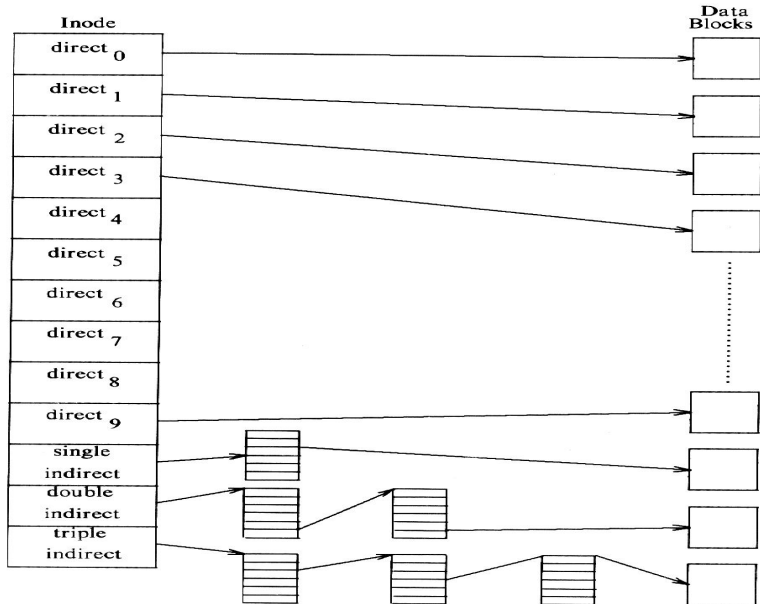
informations générales sur le SGF

chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

chaque bloc : 512/1024/2048/4096 octets





# Les SGF ffs/ufs (BSD)

bloc initialisation
super bloc
tables de groupe de cylindre
table des i-noeuds
⋮
blocs de données
⋮

optimisation allocation des blocs  
par rapport à la tête du disque

chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,  
fragmentation des blocs

Répertoires :  
enregistrements taille variable,  
liens taille max 255 caractères

# Les SGF ffs/ufs (BSD)

bloc initialisation
super bloc
tables de groupe de cylindre
table des i-noeuds
⋮
blocs de données
⋮

optimisation allocation des blocs  
par rapport à la tête du disque

chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,  
fragmentation des blocs

Répertoires :  
enregistrements taille variable,  
liens taille max 255 caractères

# Les SGF ffs/ufs (BSD)

bloc initialisation
super bloc
tables de groupe de cylindre
table des i-noeuds
⋮
blocs de données
⋮

optimisation allocation des blocs  
par rapport à la tête du disque

chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,  
fragmentation des blocs

Répertoires :  
enregistrements taille variable,  
liens taille max 255 caractères

# Les SGF ffs/ufs (BSD)

bloc initialisation
super bloc
tables de groupe de cylindre
table des i-noeuds
⋮
<b>blocs de données</b>
⋮

optimisation allocation des blocs  
par rapport à la tête du disque

chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,  
fragmentation des blocs

Répertoires :  
enregistrements taille variable,  
liens taille max 255 caractères

# Les SGF ffs/ufs (BSD)

bloc initialisation
super bloc
tables de groupe de cylindre
table des i-noeuds
⋮
blocs de données
⋮

optimisation allocation des blocs  
par rapport à la tête du disque

chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,  
fragmentation des blocs

Répertoires :  
enregistrements taille variable,  
liens taille max 255 caractères

# Plan

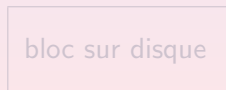
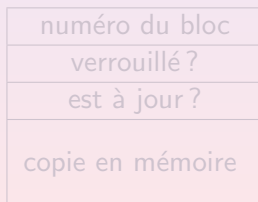
- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - **Les caches**
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

# L'accès aux blocs

Couteaux si les blocs sont résident sur le disque.

Solution :

- faire une copie en mémoire des blocs, en ajoutant un en-tête,
- mettre à jour les blocs sur disque seulement quand il se rend nécessaire.



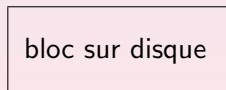
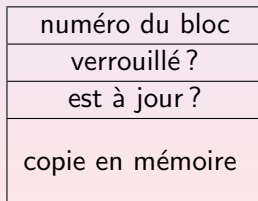


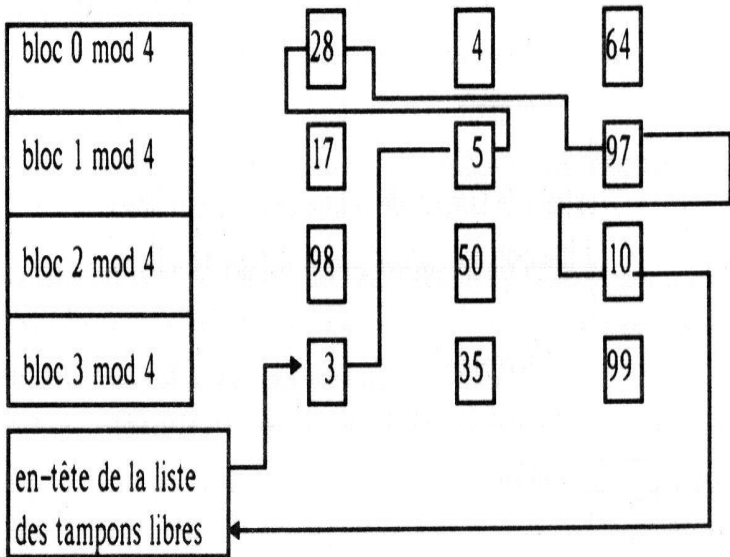
# L'accès aux blocs

Couteaux si les blocs sont résident sur le disque.

Solution :

- faire une copie en mémoire des blocs, en ajoutant un en-tête,
- mettre à jour les blocs sur disque seulement quand il se rend nécessaire.





# Programme : getblk.c

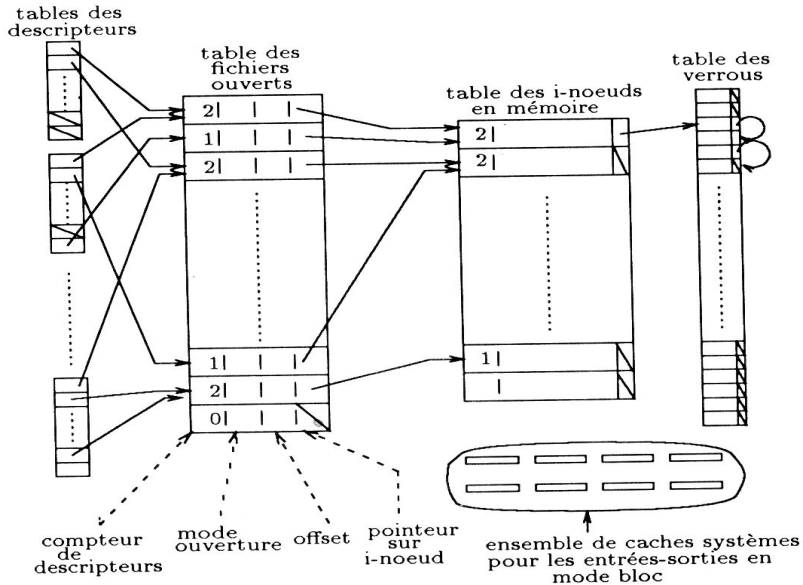
```
1 struct tampon * getblk(no_bloc)
2 {
3     while(tampon non retourné)
4     {
5         if(
6             (tampon = (chercher no_bloc dans la file à hachage))
7             == trouvé
8             )
9             { /* Le bloc demandé est
10                dans la file d'hachage */
11                if(tampon verrouillé)
12                    { /* Scénario 5 */
13                        sleep(jusqu à tampon libre);
14                        continue;
15                    }
16                /* Scénario 1 */
17                verrouiller(tampon);
18                extraire tampon file_tampons_libres;
19                return tampon;
20            }
```

# Programme : getblk.c(II)

```
21     else
22     {
23         /* Le bloc demandé se trouve sur disque */
24         if(file_tampons_libres est vide)
25             { /* Scénario 4 */
26                 sleep(jusqu à tampon libre);
27                 continue;
28             }
29         tampon = tête de la file_tampons_libres;
30         if(tampon marqué pour écriture différée)
31             { /* Scénario 3 */
32                 écriture asynchrone tampon sur le disque ;
33                 continue;
34             }
35         /* Scénario 2 */
36         enlever tampon de la file_tampons_libres;
37         insérer tampon dans la file à hachage;
38         return tampon;
39     }
40 }
41 }
```

# Plan

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX



# Les tables du système

- Tables des descripteurs, appartenant à un processus.

Descripteurs conventionnels :

- 0 STDIN\_FILENO
- 1 STDOUT\_FILENO
- 2 STDERR\_FILENO

- Table des fichiers ouverts (appartenant au noyau) :

- compteur des descripteurs,
- mode d'ouverture,
- position courante,
- pointeur sur le i-noeud en mémoire.

- Table des i-noeuds en cache (noyau):

- nombre total d'ouvertures,
- id du disque logique,
- numéro du noeud sur ce disque,
- état du noeud.

# Les tables du système

- Tables des descripteurs, appartenant à un processus.

Descripteurs conventionnels :

0 STDIN\_FILENO

1 STDOUT\_FILENO

2 STDERR\_FILENO

- Table des fichiers ouverts (appartenant au noyau) :

- compteur des descripteurs,
- mode d'ouverture,
- position courante,
- pointeur sur le i-noeud en mémoire.

- Table des i-noeuds en cache (noyau):

- nombre total d'ouvertures,
- id du disque logique,
- numéro du noeud sur ce disque,
- état du noeud.



# Les tables du système

- Tables des descripteurs, appartenant à un processus.

Descripteurs conventionnels :

0 STDIN\_FILENO

1 STDOUT\_FILENO

2 STDERR\_FILENO

- Table des fichiers ouverts (appartenant au noyau) :

- compteur des descripteurs,
- mode d'ouverture,
- position courante,
- pointeur sur le i-noeud en mémoire.

- Table des i-noeuds en cache (noyau):

- nombre total d'ouvertures,
- id du disque logique,
- numéro du noeud sur ce disque,
- état du noeud.

# Plan

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

# fopen, fclose

```
#include <stdio.h>
```

```
FILE * fopen(const char * fic, const char * mode);
```

*fic* : le nom du fichier à ouvrir

*mode* : le mode d'ouverture: "r", "w", "a", "r+", "w+", "a".

Retourne : pointeur sur struct FILE, NULL si erreur.

```
int fclose(FILE * fic);
```

*fic* : pointeur sur le fichier ouvert qu'on veut fermer

Retourne : 0 ou EOF si erreur

# fopen, fclose

```
#include <stdio.h>
```

```
FILE * fopen(const char * fic, const char * mode);
```

*fic* : le nom du fichier à ouvrir

*mode* : le mode d'ouverture: "r", "w", "a", "r+", "w+", "a".

Retourne : pointeur sur struct FILE, NULL si erreur.

```
int fclose(FILE * fic);
```

*fic* : pointeur sur le fichier ouvert qu'on veut fermer

Retourne : 0 ou EOF si erreur

# fopen, fclose

```
#include <stdio.h>
```

```
FILE * fopen(const char * fic, const char * mode);
```

*fic* : le nom du fichier a ouvrir

*mode* : le mode d'ouverture: "r", "w", "a", "r+", "w+", "a".

Retourne : pointeur sur struct FILE, NULL si erreur.

```
int fclose(FILE * fic);
```

*fic* : pointeur sur le fichier ouvert qu'on veut fermer

Retourne : 0 ou EOF si erreur

# Modes d'ouverture

"r" (read) : lecture, positionnement au début.

"w" (write): écriture, positionnement au début,  
création/écrasement du fichier.

"a" (append) : écriture, positionnement à la fin,  
pas d'écrasement du fichier.

"r+" : lecture/écriture, positionnement au début,  
pas d'écrasement du fichier existant.

"w+" : lecture/écriture, positionnement au début,  
écrasement du fichier.

"a+" : lecture/écriture, positionnement à la fin,  
pas d'écrasement du fichier.

# fprintf, fscanf, fgetc, fgets

```
#include <stdio.h>
```

```
int fprintf(FILE * flot, const char format, ...);
```

Retourne : nombre d'octets écrits, ou valeur négatif si erreur.

```
int fscanf(FILE * flot, const char format, ...);
```

Retourne : nombre de paramètres reconnus, ou EOF si erreur de *flot*.

```
int fgetc(FILE * flot);
```

Retourne : le caractère ou EOF.

```
char * fgets(char * s tampon, int maxtaille, FILE * flot);
```

Retourne : *tampon* ou NULL si erreur ou EOF.

Remarques : Caractère fin de ligne \n stocké dans le tampon

# fprintf, fscanf, fgetc, fgets

```
#include <stdio.h>
```

```
int fprintf(FILE * flot, const char format, ...);
```

Retourne : nombre d'octets écrits, ou valeur négatif si erreur.

```
int fscanf(FILE * flot, const char format, ...);
```

Retourne : nombre de paramètres reconnus, ou EOF si erreur de *flot*.

```
int fgetc(FILE * flot);
```

Retourne : le caractère ou EOF.

```
char * fgets(char * s tampon, int maxtaille, FILE * flot);
```

Retourne : *tampon* ou NULL si erreur ou EOF.

Remarques : Caractère fin de ligne `\n` stocké dans le tampon



# fprintf, fscanf, fgetc, fgets

```
#include <stdio.h>
```

```
int fprintf(FILE * flot, const char format, ...);
```

Retourne : nombre d'octets écrits, ou valeur négatif si erreur.

```
int fscanf(FILE * flot, const char format, ...);
```

Retourne : nombre de paramètres reconnus, ou EOF si erreur de *flot*.

```
int fgetc(FILE * flot);
```

Retourne : le caractère ou EOF.

```
char * fgets(char * s tampon, int maxtaille, FILE * flot);
```

Retourne : *tampon* ou NULL si erreur ou EOF.

Remarques : Caractère fin de ligne `\n` stocké dans le tampon

# fprintf, fscanf, fgetc, fgets

```
#include <stdio.h>
```

```
int fprintf(FILE * flot, const char format, ...);
```

Retourne : nombre d'octets écrits, ou valeur négatif si erreur.

```
int fscanf(FILE * flot, const char format, ...);
```

Retourne : nombre de paramètres reconnus, ou EOF si erreur de *flot*.

```
int fgetc(FILE * flot);
```

Retourne : le caractère ou EOF.

```
char * fgets(char * s tampon, int maxtaille, FILE * flot);
```

Retourne : *tampon* ou NULL si erreur ou EOF.

Remarques : Caractère fin de ligne `\n` stocké dans le tampon

# fprintf, fscanf, fgetc, fgets

```
#include <stdio.h>
```

```
int fprintf(FILE * flot, const char format, ...);
```

Retourne : nombre d'octets écrits, ou valeur négatif si erreur.

```
int fscanf(FILE * flot, const char format, ...);
```

Retourne : nombre de paramètres reconnus, ou EOF si erreur de *flot*.

```
int fgetc(FILE * flot);
```

Retourne : le caractère ou EOF.

```
char * fgets(char * s tampon, int maxtaille, FILE * flot);
```

Retourne : *tampon* ou NULL si erreur ou EOF.

Remarques : Caractère fin de ligne `\n` stocké dans le tampon.

## Programme : comptage.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void compter(char nom[], FILE *f)
5 {
6     int c , cars = 0, lignes = 0;
7
8     while( (c = fgetc(f)) != EOF ){
9         cars ++;
10        if (c == '\n') lignes ++;
11    }
12    printf("%s : %d caractères, %d lignes.\n",
13           nom, cars, lignes);
14 }
```

## Programme : comptage.c(II)

```
16 int main(int argc, char *argv[])
17 {
18     FILE * f;
19
20     if (argc != 2)
21         exit(EXIT_FAILURE);
22
23     if ( (f=fopen(argv[1], "r")) == NULL)
24         {
25             perror("fopen");
26             exit(EXIT_FAILURE);
27         }
28     compter(argv[1], f);
29     fclose(f);
30     exit(EXIT_SUCCESS);
31 }
```

## fflush

```
#include <stdio.h>
int fflush(FILE * flot);
```

*flot* : flot ouvert en écriture.

Retourne : 0/EOF

Remarques : Vide le tampon du flot *flot*.

# fflush

```
#include <stdio.h>  
int fflush(FILE * flot);
```

*flot* : flot ouvert en écriture.

Retourne : 0/EOF

Remarques : Vide le tampon du flot *flot*.

# fflush

```
#include <stdio.h>
int fflush(FILE * flot);
```

*flot* : flot ouvert en écriture.

Retourne : 0/EOF

Remarques : Vide le tampon du flot *flot*.



# fflush

```
#include <stdio.h>
int fflush(FILE * flot);
```

*flot* : flot ouvert en écriture.

Retourne : 0/EOF

Remarques : Vide le tampon du flot *flot*.

# stdin, stdout, stderr, printf, scanf, getc, gets

```
#include <stdio.h>
```

```
FILE * stdin, stdout, stderr;
```

```
int printf(const char format, ...);
```

Remarques : équivalent à `fprintf(stdout, ...)`.

```
int scanf(const char format, ...);
```

Remarques : équivalent à `fscanf(stdin, ...)`.

```
int getc(FILE * flot);
```

Remarques : équivalent à `fgetc(stdin)`.

```
char * gets(FILE * flot);
```

Remarques : Ne pas utiliser !!!

# stdin, stdout, stderr, printf, scanf, getc, gets

```
#include <stdio.h>
```

```
FILE * stdin, stdout, stderr;
```

```
int printf(const char format, ...);
```

Remarques : équivalent à `fprintf(stdout, ...)`.

```
int scanf(const char format, ...);
```

Remarques : équivalent à `fscanf(stdin, ...)`.

```
int getc(FILE * flot);
```

Remarques : équivalent à `fgetc(stdin)`.

```
char * gets(FILE * flot);
```

Remarques : Ne pas utiliser !!!

# stdin, stdout, stderr, printf, scanf, getc, gets

```
#include <stdio.h>
```

```
FILE * stdin, stdout, stderr;
```

```
int printf(const char format, ...);
```

Remarques : équivalent à `fprintf(stdout, ...)`.

```
int scanf(const char format, ...);
```

Remarques : équivalent à `fscanf(stdin, ...)`.

```
int getc(FILE * flot);
```

Remarques : équivalent à `fgetc(stdin)`.

```
char * gets(FILE * flot);
```

Remarques : Ne pas utiliser !!!

# stdin, stdout, stderr, printf, scanf, getc, gets

```
#include <stdio.h>
```

```
FILE * stdin, stdout, stderr;
```

```
int printf(const char format, ...);
```

Remarques : équivalent à `fprintf(stdout, ...)`.

```
int scanf(const char format, ...);
```

Remarques : équivalent à `fscanf(stdin, ...)`.

```
int getc(FILE * flot);
```

Remarques : équivalent à `fgetc(stdin)`.

```
char * gets(FILE * flot);
```

Remarques : Ne pas utiliser !!!

# stdin, stdout, stderr, printf, scanf, getc, gets

```
#include <stdio.h>
```

```
FILE * stdin, stdout, stderr;
```

```
int printf(const char format, ...);
```

Remarques : équivalent à `fprintf(stdout, ...)`.

```
int scanf(const char format, ...);
```

Remarques : équivalent à `fscanf(stdin, ...)`.

```
int getc(FILE * flot);
```

Remarques : équivalent à `fgetc(stdin)`.

```
char * gets(FILE * flot);
```

Remarques : Ne pas utiliser !!!

# stdin, stdout, stderr, printf, scanf, getc, gets

```
#include <stdio.h>
```

```
FILE * stdin, stdout, stderr;
```

```
int printf(const char format, ...);
```

Remarques : équivalent à `fprintf(stdout, ...)`.

```
int scanf(const char format, ...);
```

Remarques : équivalent à `fscanf(stdin, ...)`.

```
int getc(FILE * flot);
```

Remarques : équivalent à `fgetc(stdin)`.

```
char * gets(FILE * flot);
```

Remarques : Ne pas utiliser !!!

## fread

```
#include <stdio.h>
size_t
fread(void * ptr,
      size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointeur à un tampon en mémoire (déjà allouée)

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut lire, tel que  
 $nb * taille \leq \text{sizeof}(ptr)$

*fic* : le pointer au flot (fichier ouvert en mode écriture) qu'on  
veut lire

Retourne : nombre d'objets lus ( $\leq nb$ ).



# fread

```
#include <stdio.h>
size_t
fread(void * ptr,
      size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointeur à un tampon en mémoire (déjà allouée)

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut lire, tel que  
 $nb * taille \leq \text{sizeof}(ptr)$

*fic* : le pointer au flot (fichier ouvert en mode écriture) qu'on  
veut lire

Retourne : nombre d'objets lus ( $\leq nb$ ).

# fread

```
#include <stdio.h>
size_t
fread(void * ptr,
      size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointeur à un tampon en mémoire (déjà allouée)

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut lire, tel que  
 $nb * taille \leq \text{sizeof}(ptr)$

*fic* : le pointer au flot (fichier ouvert en mode écriture) qu'on  
veut lire

Retourne : nombre d'objets lus ( $\leq nb$ ).

## fread

```
#include <stdio.h>
size_t
fread(void * ptr,
      size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointeur à un tampon en mémoire (déjà allouée)

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut lire, tel que  
 $nb * taille \leq \text{sizeof}(ptr)$

*fic* : le pointer au flot (fichier ouvert en mode écriture) qu'on  
veut lire

Retourne : nombre d'objets lus ( $\leq nb$ ).

## fread

```
#include <stdio.h>
size_t
fread(void * ptr,
      size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointeur à un tampon en mémoire (déjà allouée)

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut lire, tel que  
 $nb * taille \leq \text{sizeof}(ptr)$

*fic* : le pointer au flot (fichier ouvert en mode écriture) qu'on veut lire

Retourne : nombre d'objets lus ( $\leq nb$ ).

## fread

```
#include <stdio.h>
size_t
fread(void * ptr,
      size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointeur à un tampon en mémoire (déjà allouée)

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut lire, tel que  
 $nb * taille \leq \text{sizeof}(ptr)$

*fic* : le pointer au flot (fichier ouvert en mode écriture) qu'on  
veut lire

Retourne : nombre d'objets lus ( $\leq nb$ ).

## fwrite

```
#include <stdio.h>
size_t
fwrite(void * ptr,
        size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointer à un tampon en mémoire

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut écrire

*fic* : le flot (fichier ouvert en écriture) qu'on veut écrire

Retourne : nombre d'objets écrits ( $\leq nb$ ).

# fwrite

```
#include <stdio.h>
size_t
fwrite(void * ptr,
        size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointer à un tampon en mémoire

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut écrire

*fic* : le flot (fichier ouvert en écriture) qu'on veut écrire

Retourne : nombre d'objets écrits ( $\leq nb$ ).

## fwrite

```
#include <stdio.h>
size_t
fwrite(void * ptr,
        size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointer à un tampon en mémoire

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut écrire

*fic* : le flot (fichier ouvert en écriture) qu'on veut écrire

Retourne : nombre d'objets écrits ( $\leq nb$ ).



## fwrite

```
#include <stdio.h>
size_t
fwrite(void * ptr,
        size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointer à un tampon en mémoire

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut écrire

*fic* : le flot (fichier ouvert en écriture) qu'on veut écrire

Retourne : nombre d'objets écrits ( $\leq nb$ ).

## fwrite

```
#include <stdio.h>
size_t
fwrite(void * ptr,
        size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointer à un tampon en mémoire

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut écrire

*fic* : le flot (fichier ouvert en écriture) qu'on veut écrire

Retourne : nombre d'objets écrits ( $\leq nb$ ).

## fwrite

```
#include <stdio.h>
size_t
fwrite(void * ptr,
        size_t taille, size_t nb, FILE * fic);
```

*ptr* : un pointer à un tampon en mémoire

*taille* : nombre d'octets pour chaque objets

*nb* : le nombre d'objets qu'on veut écrire

*fic* : le flot (fichier ouvert en écriture) qu'on veut écrire

Retourne : nombre d'objets écrits ( $\leq nb$ ).

# Programme : copier.c

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #define TAILLETAMPON 1024
4
5 int main(int argc, char *argv[])
6 {
7     FILE *source, *dest;
8     char tampon[TAILLETAMPON];
9     size_t nolu;
10
11     if (argc != 3)
12     {
13         printf("Usage : %s source destination\n",
14             argv[0]); exit(EXIT_FAILURE);
15     }
16     if( (source = fopen(argv[1],"r")) == NULL
17         || (dest = fopen(argv[2],"w")) == NULL )
18     {
19         perror("fopen"); exit(EXIT_FAILURE);
20     }
```

## Programme : copier.c(II)

```
22  while(
23      (nolu = fread(tampon,
24                  sizeof(char), sizeof(tampon), source))
25      > 0)
26      fwrite(tampon, sizeof(char), nolu, dest) ;
27
28  fclose(source); fclose(dest);
29  exit(EXIT_SUCCESS);
30 }
```

# fseek

```
#include <stdio.h>
int
fseek(FILE * fic, long int offset, int or);
```

*fic* : pointeur au flot ouvert

*offset* : déplacement par rapport à origine

*or* : origine du déplacement :

SEEK\_SET : début du fichier.

SEEK\_CURR : position courante.

SEEK\_END : fin du fichier.

Retourne : 0 succès, -1 erreur.

# fseek

```
#include <stdio.h>
int
fseek(FILE * fic, long int offset, int or);
```

*fic* : pointeur au flot ouvert

*offset* : déplacement par rapport à origine

*or* : origine du déplacement :

SEEK\_SET : début du fichier.

SEEK\_CURR : position courante.

SEEK\_END : fin du fichier.

Retourne : 0 succès, -1 erreur.

# fseek

```
#include <stdio.h>
int
fseek(FILE * fic, long int offset, int or);
```

*fic* : pointeur au flot ouvert

*offset* : déplacement par rapport à origine

*or* : origine du déplacement :

SEEK\_SET : début du fichier.

SEEK\_CURR : position courante.

SEEK\_END : fin du fichier.

Retourne : 0 succès, -1 erreur.



## fseek

```
#include <stdio.h>
int
fseek(FILE * fic, long int offset, int or);
```

*fic* : pointeur au flot ouvert

*offset* : déplacement par rapport à origine

*or* : origine du déplacement :

SEEK\_SET : début du fichier.

SEEK\_CURR : position courante.

SEEK\_END : fin du fichier.

Retourne : 0 succès, -1 erreur.

## fseek

```
#include <stdio.h>
int
fseek(FILE * fic, long int offset, int or);
```

*fic* : pointeur au flot ouvert

*offset* : déplacement par rapport à origine

*or* : origine du déplacement :

SEEK\_SET : début du fichier.

SEEK\_CURR : position courante.

SEEK\_END : fin du fichier.

Retourne : 0 succès, -1 erreur.

# Plan

- 1 Organisation externe
  - Tout est un fichier : les types de fichiers
  - Les fichiers ordinaires : vue utilisateur
- 2 Organisation interne
  - Le système de gestion de fichiers
  - Les caches
  - Les tables des fichiers
- 3 L'interface de programmation du système de gestion de fichiers
  - Les E/S de la bibliothèque standard C
  - L'interface des E/S POSIX

## open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

*fic* : le nom du fichier à ouvrir

*mode* : disjonction bit-à-bit de :

O\_RDONLY, O\_WRONLY, O\_RDWR,  
O\_APPEND, O\_TRUNC,  
O\_CREAT, O\_EXCL,  
O\_NONBLOCK,

*droits* : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

## open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

*fic* : le nom du fichier à ouvrir

*mode* : disjonction bit-à-bit de :

- O\_RDONLY, O\_WRONLY, O\_RDWR,
- O\_APPEND, O\_TRUNC,
- O\_CREAT, O\_EXCL,
- O\_NONBLOCK,

*droits* : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

## open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

*fic* : le nom du fichier à ouvrir

*mode* : disjonction bit-à-bit de :

- O\_RDONLY, O\_WRONLY, O\_RDWR,
- O\_APPEND, O\_TRUNC,
- O\_CREAT, O\_EXCL,
- O\_NONBLOCK,

*droits* : ... voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

## open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

*fic* : le nom du fichier à ouvrir

*mode* : disjonction bit-à-bit de :

- O\_RDONLY, O\_WRONLY, O\_RDWR,
- O\_APPEND, O\_TRUNC,
- O\_CREAT, O\_EXCL,
- O\_NONBLOCK,

*droits* : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

## open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

*fic* : le nom du fichier à ouvrir

*mode* : disjonction bit-à-bit de :

- O\_RDONLY, O\_WRONLY, O\_RDWR,
- O\_APPEND, O\_TRUNC,
- O\_CREAT, O\_EXCL,
- O\_NONBLOCK,

*droits* : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.



# Les permissions sur un fichier

	USeR	GRouP	OTHerS
Read	S_IRUSR	S_IRGRP	S_IROTH
Write	S_IWUSR	S_IWGRP	S_IWOTH
eXecute	S_IXUSR	S_IXGRP	S_IXOTH

S\_IRWXU = S\_IRUSR | S\_IWUSR | S\_IXUSR

S\_IRWXG = S\_IRGRP | S\_IWGRP | S\_IXGRP

S\_IRWXO = S\_IROTH | S\_IWOTH | S\_IXOTH

# Les permissions sur un fichier

	USeR	GRouP	OTHerS
Read	S_IRUSR	S_IRGRP	S_IROTH
Write	S_IWUSR	S_IWGRP	S_IWOTH
eXecute	S_IXUSR	S_IXGRP	S_IXOTH

S\_IRWXU = S\_IRUSR | S\_IWUSR | S\_IXUSR

S\_IRWXG = S\_IRGRP | S\_IWGRP | S\_IXGRP

S\_IRWXO = S\_IROTH | S\_IWOTH | S\_IXOTH

## creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char * fic, mode_t droits);
```

*fic* : le nom du fichier à ouvrir,  
*droits* : voire les permissions.

Remarques : équivalent à  
`open(fic, O_WRONLY|O_CREAT|O_TRUNC,droits)`

# creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char * fic, mode_t droits);
```

*fic* : le nom du fichier à ouvrir,  
*droits* : voire les permissions.

Remarques : équivalent à  
`open(fic, O_WRONLY|O_CREAT|O_TRUNC,droits)`

# creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char * fic, mode_t droits);
```

*fic* : le nom du fichier à ouvrir,  
*droits* : voire les permissions.

Remarques : équivalent à  
`open(fic, O_WRONLY|O_CREAT|O_TRUNC,droits)`

# creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char * fic, mode_t droits);
```

*fic* : le nom du fichier à ouvrir,  
*droits* : voire les permissions.

Remarques : équivalent à  
`open(fic, O_WRONLY|O_CREAT|O_TRUNC,droits)`

## close

```
#include <unistd.h>  
int close(int desc);
```

*desc* : le descripteur du fichier qu'on veut fermer

Retourne : 0/-1

# close

```
#include <unistd.h>  
int close(int desc);
```

*desc* : le descripteur du fichier qu'on veut fermer

Retourne : 0/-1



# close

```
#include <unistd.h>  
int close(int desc);
```

*desc* : le descripteur du fichier qu'on veut fermer

Retourne : 0/-1

## read

```
#include <unistd.h>
ssize_t
read(int desc, void * ptr, ssize_t nb);
```

*desc* : descripteur d'un fichier ouvert en lecture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut lire, tel que  
 $nb \leq \text{sizeof}(ptr)$

Retourne : nombre d'octets lus/-1

## read

```
#include <unistd.h>
ssize_t
read(int desc, void * ptr, ssize_t nb);
```

*desc* : descripteur d'un fichier ouvert en lecture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut lire, tel que

$nb \leq \text{sizeof}(ptr)$

Retourne : nombre d'octets lus/-1

# read

```
#include <unistd.h>
ssize_t
read(int desc, void * ptr, ssize_t nb);
```

*desc* : descripteur d'un fichier ouvert en lecture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut lire, tel que  
`nb <= sizeof(ptr)`

Retourne : nombre d'octets lus/-1

# read

```
#include <unistd.h>
ssize_t
read(int desc, void * ptr, ssize_t nb);
```

*desc* : descripteur d'un fichier ouvert en lecture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut lire, tel que  
 $nb \leq \text{sizeof}(ptr)$

Retourne : nombre d'octets lus/-1

## read

```
#include <unistd.h>
ssize_t
read(int desc, void * ptr, ssize_t nb);
```

*desc* : descripteur d'un fichier ouvert en lecture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut lire, tel que

$nb \leq \text{sizeof}(ptr)$

Retourne : nombre d'octets lus/-1

## write

```
#include <unistd.h>
ssize_t
write(int desc, void * ptr, ssize_t nb);
```

*desc* : le descripteur du fichier ouvert en écriture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut écrire

Retourne : nombre d'octets écrits/-1

# write

```
#include <unistd.h>
ssize_t
write(int desc, void * ptr, ssize_t nb);
```

*desc* : le descripteur du fichier ouvert en écriture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut écrire

Retourne : nombre d'octets écrits/-1



## write

```
#include <unistd.h>
ssize_t
write(int desc, void * ptr, ssize_t nb);
```

*desc* : le descripteur du fichier ouvert en écriture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut écrire

Retourne : nombre d'octets écrits/-1

# write

```
#include <unistd.h>
ssize_t
write(int desc, void * ptr, ssize_t nb);
```

*desc* : le descripteur du fichier ouvert en écriture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut écrire

Retourne : nombre d'octets écrits/-1

# write

```
#include <unistd.h>
ssize_t
write(int desc, void * ptr, ssize_t nb);
```

*desc* : le descripteur du fichier ouvert en écriture

*ptr* : un pointer à un tampon en mémoire (déjà allouée)

*nb* : le nombre d'octets qu'on veut écrire

Retourne : nombre d'octets écrits/-1

# Programme : copierunix.c

```
1 #include <stdlib.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 #define TAILLETAMPON 1024
6
7 int main(int argc, char *argv[])
8 {
9     int source, dest;
10    char tampon[TAILLETAMPON];
11    ssize_t nolu;
12
13    if (argc != 3)
14        {
15            printf("Usage : %s source destination\n", argv[0]);
16            exit(EXIT_FAILURE);
17        }
18    if( (source = open(argv[1], O_RDONLY)) == -1
19        || (dest = open(argv[2], O_WRONLY)) == -1 )
20        { perror("open"); exit(EXIT_FAILURE); }
```

## Programme : copierunix.c(II)

```
22     while((nolu =
23             read(source, tampon, sizeof(tampon)))
24             > 0)
25         write(dest, tampon, nolu) ;
26
27     close(source); close(dest);
28     exit(EXIT_SUCCESS);
29 }
```

# lseek

```
off_t lseek(int fd, off_t offset, int or);
```

*fd* : descripteur du fichier sur lequel on veut agir

*offset* : déplacement par rapport à l'origine

*or* : origine, à savoir un de

SEEK\_SET : début du fichier

SEEK\_CURR : position courante

SEEK\_END : fin du fichier.

Retourne : position courante à partir de l'origine du fichier,  
(off\_t) -1 si erreur

# lseek

```
off_t lseek(int fd, off_t offset, int or);
```

*fd* : descripteur du fichier sur lequel on veut agir

*offset* : déplacement par rapport à l'origine

*or* : origine, à savoir un de

SEEK\_SET : début du fichier

SEEK\_CURR : position courante

SEEK\_END : fin du fichier.

Retourne : position courante à partir de l'origine du fichier,  
(off\_t) -1 si erreur

# lseek

```
off_t lseek(int fd, off_t offset, int or);
```

*fd* : descripteur du fichier sur lequel on veut agir

*offset* : déplacement par rapport à l'origine

*or* : origine, à savoir un de

SEEK\_SET : début du fichier

SEEK\_CURR : position courante

SEEK\_END : fin du fichier.

Retourne : position courante à partir de l'origine du fichier,  
(*off\_t*) -1 si erreur



# lseek

```
off_t lseek(int fd, off_t offset, int or);
```

*fd* : descripteur du fichier sur lequel on veut agir

*offset* : déplacement par rapport à l'origine

*or* : origine, à savoir un de

SEEK\_SET : début du fichier

SEEK\_CURR : position courante

SEEK\_END : fin du fichier.

Retourne : position courante à partir de l'origine du fichier,  
(off\_t) -1 si erreur

# lseek

```
off_t lseek(int fd, off_t offset, int or);
```

*fd* : descripteur du fichier sur lequel on veut agir

*offset* : déplacement par rapport à l'origine

*or* : origine, à savoir un de

SEEK\_SET : début du fichier

SEEK\_CURR : position courante

SEEK\_END : fin du fichier.

Retourne : position courante à partir de l'origine du fichier,  
(off\_t) -1 si erreur