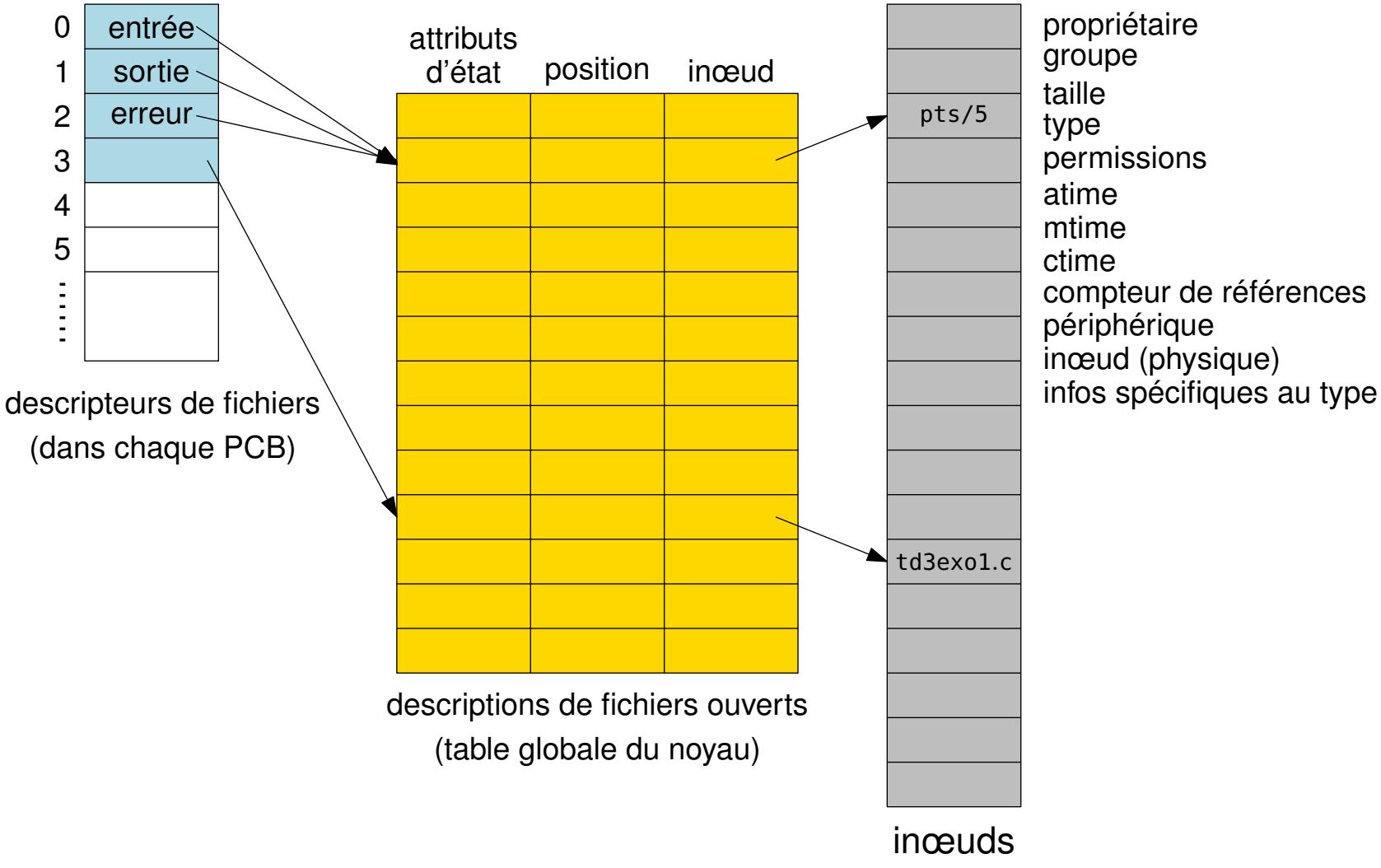
UNIVERSITÉ PARIS SUD

IUT D'ORSAY

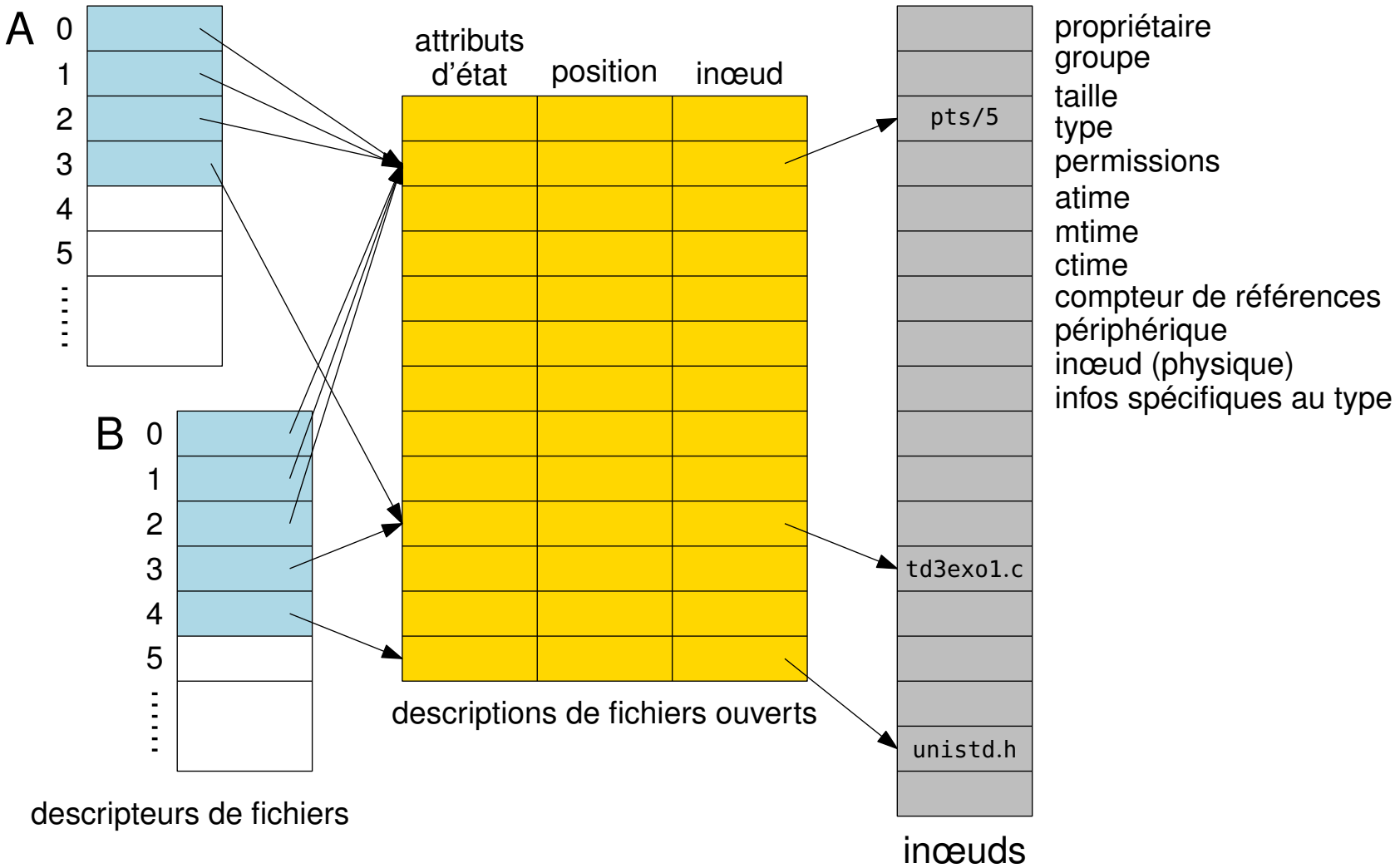
M3101 · Principes des systèmes d'exploitation

Les fichiers

Les fichiers, les processus et le noyau



Les fichiers, les processus et le noyau



Les primitives

`open()` crée une nouvelle description de fichier ouvert dans la table globale des fichiers ouverts. Réserve un nouveau numéro descripteur de fichier pour le processus (et le lui renvoie).

`read()` transfère des données depuis le fichier vers la mémoire. Avance la position dans la description de fichier ouvert.

`write()` transfère des données depuis la mémoire vers le fichier. Avance la position dans la description de fichier ouvert.

`lseek()` modifie la position dans la description de fichier ouvert.

`close()` libère le numéro descripteur et, éventuellement, détruit la description de fichier ouvert.

`fcntl()` modifie les attributs d'un fichier ouvert.

Toutes les primitives travaillent avec des descripteurs de fichiers.

Opérations sur les fichiers

Selon le niveau du langage et des bibliothèques utilisés :

primitives système	**C / libc**	**C++**
descripteur de fichier	structure « fichier »	objet flux d'entrée-sortie
`int`	`FILE*`	`iostream`
`0`	`stdin`	`cin`
`1`	`stdout`	`cout`
`2`	`stderr`	`cerr`
`read()`	`fscanf()`	`>>`
`write()`	`fprintf()`	`<<`
pas de conversion,	conversion selon	conversion selon
données brutes	le code de format	le type d'argument

Ouvrir un fichier

```
int open(const char *chemin, int drapeaux);
```

```
int open(const char *chemin, int drapeaux, mode_t mode);
```

```
int creat(const char *chemin, mode_t mode);
```

`chemin` le nom du fichier à ouvrir

`drapeaux` Un et un seul de `O_RDONLY`, `O_WRONLY` ou `O_RDWR`.
En plus, avec l'opération '|' (« ou » bit-par-bit) :
`O_APPEND`, `O_CREAT`, `O_EXCL`, `O_TRUNC`,
`O_NONBLOCK`, `O_DIRECT`, `O_SYNC`, `O_ASYNC`...

`mode` en cas de création, par `O_CREAT` ou `creat()`,
le droits d'accès en octal, comme pour `chmod` :
par exemple, `0644` correspond à `rw-r--r--`

Renvoient le numéro de descripteur attribué au fichier ouvert (ou -1).

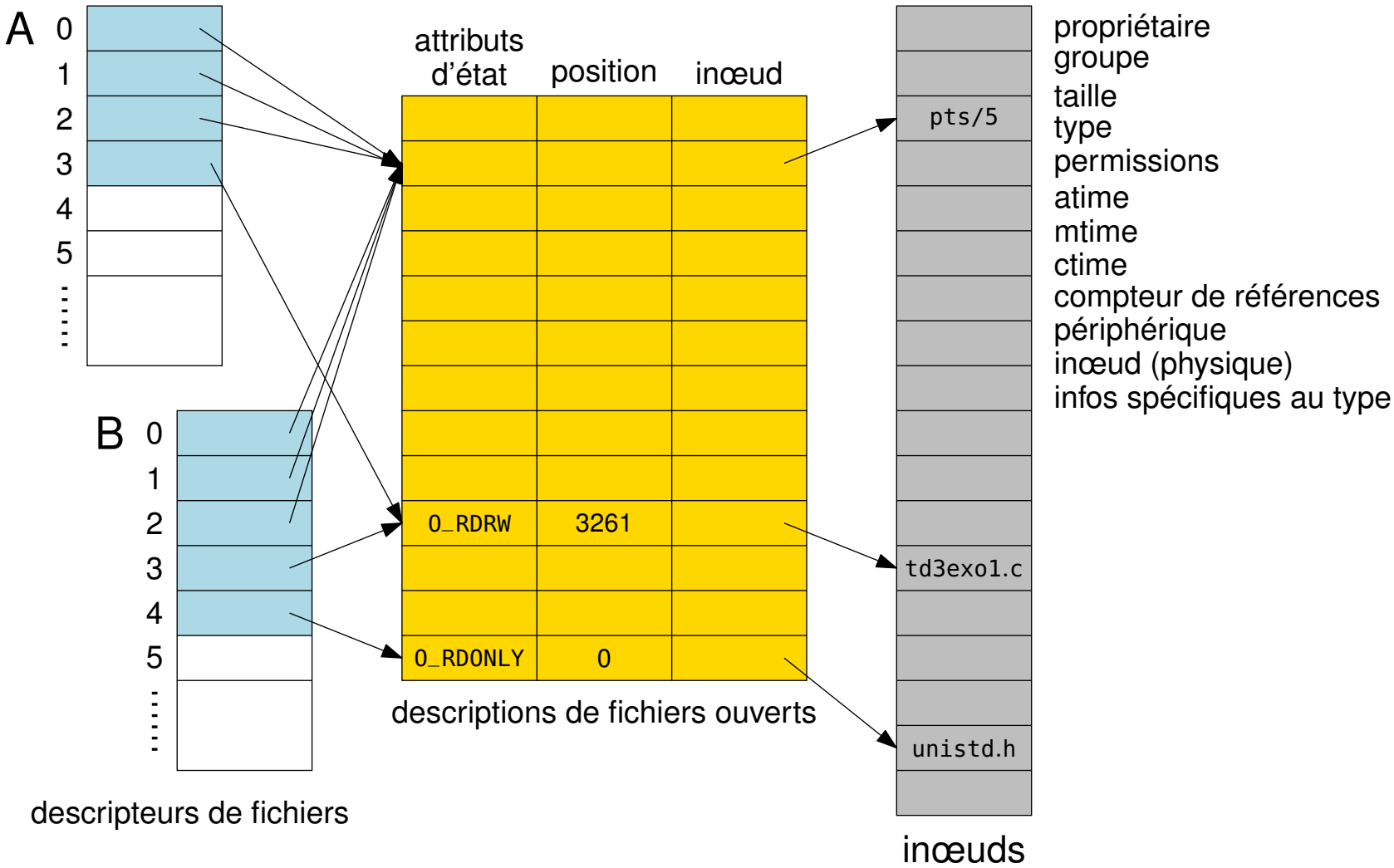
Ouvrir un fichier

1. Le noyau retrouve l'inœud du fichier indiqué par le chemin
   - `O_CREAT` est parmi les drapeaux et le fichier n'existe pas
     $\Rightarrow$ un nouveau inœud est créé avec les permissions dans mode
   - `O_CREAT` et `O_EXCL` sont parmi les drapeaux et le fichier existe
     $\Rightarrow$ `open()` échoue
   - `O_WRONLY` / `O_RDWR` et `O_TRUNC` sont parmi les drapeaux
     $\Rightarrow$ le fichier est remis à vide

2. Une nouvelle description de fichier ouvert est créée :
   - les attributs d'état sont les drapeaux (modifiables par `fcntl()`)
   - la position (le n° du prochain octet à lire ou à modifier) est mise à 0
   - pointe sur l'inœud du fichier

3. Le premier descripteur libre dans la table de descripteurs du processus pointe sur la nouvelle description.

Ouvrir un fichier



Fermer un fichier

```
int close(int descripteur);
```

1. Libère le descripteur
2. S'il n'y a plus de pointeurs sur la description de fichier ouvert, alors elle est detruite.
3. Met à jour les informations dans l'inœud
   - si le nombre de références (*hard links*) dans l'inœud est 0
     et il n'y a plus de descriptions de fichier ouvert pour cet inœud,
     alors l'inœud est détruit et le fichier n'est plus accessible
   - si `close()` renvoie -1, alors l'état du fichier est inconnu,
     perte de données possible

Lire des données sur un fichier

```
int read(int fdesc, void *tampon, size_t nbOct);
```

Lit des données à partir de la position courante.

- `fdesc` le descripteur de fichier
- `tampon` stocker les données lues à partir de cette adresse
- `nbOct` le nombre maximum d'octets à lire

Renvoie le nombre d'octets lus ($\leqslant$ `nbOct`).
Augmente la valeur de position du même nombre.

Renvoie 0 en cas de fin de fichier.
Renvoie -1 en cas d'erreur.

Écrire des données sur un fichier

```
int write(int fdesc, const void *tampon, size_t nbOct);
```

Si O_APPEND est parmi les attributs d'état, met la position à la fin du fichier.

Écrit des données à partir de la position courante.

- `fdesc` le descripteur de fichier
- `tampon` prendre les données à écrire à partir de cette adresse
- `nbOct` le nombre maximum d'octets à écrire

Renvoie le nombre d'octets écrits ($\leqslant$ `nbOct`).
Augmente la valeur de position du même nombre.

Renvoie -1 en cas d'erreur.

Changer la position courante

```
off_t lseek(int fdesc, off_t depl, int origine);
```

Modifie la position courante dans la description de fichier ouvert.

- `fdesc` le descripteur de fichier
- `depl` le nombre d'octets à sauter
- `origine`
  - `SEEK_SET` — par rapport au début du fichier
  - `SEEK_CUR` — par rapport à la position courante
  - `SEEK_END` — par rapport à la fin du fichier

Renvoie la nouvelle valeur de position ou -1 en cas d'erreur.

Les descripteurs qui regardent la même description partagent la position.
$\Rightarrow$ `read()`, `write()`, `lseek()` dans un processus fils affectent le père

Exemple : un simple cat

```
1  int main (int argc, char ** argv) {
2    int fdesc, nbOct;
3    char tampon[256] = {0};
4    if (argc <= 1) exit(1); // contrôle paramètres
5    fdesc = open(argv[1], O_RDONLY); // ouverture
6    if (fdesc == -1) { perror("open() error"); exit(1); }
7    while (1) {
8      nbOct = read(fdesc, tampon, 256); // lecture
9      if (nbOct == -1) { perror("read() error"); exit(1); }
10     if (nbOct == 0) break; // fin du fichier
11     write(1, tampon, nbOct); // écriture
12   }
13   close(fdesc); // fermeture
14   return 0; }
```

Et pour les répertoires ?

Dans la bibliothèque standard de C : « flux répertoire », type `DIR`

`opendir()`	ouvre un répertoire et renvoie un `DIR*`
`closedir()`	ferme un flux répertoire
`readdir()`	lit l'entrée suivante depuis un flux répertoire
`telldir()`	renvoie la position courante dans un flux répertoire
`seekdir()`	change la position courante dans un flux répertoire
`rewinddir()`	positionne un flux répertoire au début du répertoire