

Cours de Réseau et communication Unix n°2

Edouard THIEL

Faculté des Sciences

Université d'Aix-Marseille (AMU)

Septembre 2016

Les transparents de ce cours sont téléchargeables ici :

<http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/rezo/>

Lien court : <http://j.mp/rezocom>

Plan du cours n°2

1. Les entrées-sorties
2. Les i-nodes, ou nœuds d'index
3. Les tables du système au niveau 1
4. Ouverture et fermeture au niveau 1
5. Lecture et écriture de fichiers
6. Les tubes

1 - Les entrées-sorties

Tout est fichier sous Unix

Les entrées-sorties : niveaux

Les E/S peuvent être faites à 2 niveaux.

Niveau 1 :

- ▶ bas niveau
- ▶ propre à Unix (ou à Windows)
- ▶ Intérêt : plus de contrôle

Niveau 2 :

- ▶ flux bufférisé
- ▶ bibliothèque standard C
- ▶ Intérêt : portabilité ; efficacité (bufférisé)

Manipulation des fichiers

Les fichiers sont désignés par une chaîne de caractère (chemin et nom) dans le système de fichiers.

Pour être manipulés, ils doivent être "ouverts" par le système ; il fournit en retour :

- ▶ Niveau 1 : `int fd` : descripteur de fichier
- ▶ Niveau 2 : `FILE *f` : struct opaque

Fichiers standards associés à chaque processus :

- ▶ Niveau 1 : 0 (entrée std), 1 (sortie std), 2 (sortie d'erreurs)
- ▶ Niveau 2 : `stdin`, `stdout`, `stderr`

Familles de fonctions

Fonctions normalisées POSIX, voir `man`

Niveau 1 :

`open`, `close`, `read`, `write`,
`pipe`, `socket`, `select`,
`dup`, `dup2`, `lseek`, `fstat`, `fcntl`, `fdopen` ...

Niveau 2 :

`fopen`, `fclose`, `fread`, `fwrite`,
`printf`, `scanf`, `fprintf`, `fscanf`, `vfprintf`, `perror`,
`getchar`, `gets`, `fgetc`, `fgets`,
`putchar`, `puts`, `fputc`, `fputs`,
`freopen`, `ftell`, `fseek`, `setbuffer`, `fflush`, `feof`, `fileno` ...

Mélange de niveaux

Éviter le mélange de fonctions des 2 niveaux sur un même fichier car effet imprévisible :

- ▶ lecture ou écriture,
- ▶ ouverture et fermeture,
- ▶ repositionnement, ...

Passerelles possibles :

```
FILE *fdopen(int fd, const char *mode);  
int fileno(FILE *stream);
```

2 - Les i-nodes, ou nœuds d'index

Un **i-node** est une structure enregistrée sur le disque dur.

Les i-nodes sur le disque

Les i-nodes sont créés une fois pour toute dans le volume lors de la création du système de fichiers.

Identifiée par un numéro unique sur le volume : le i-number

Décrit un fichier :

- ▶ Type fichier (régulier, répertoire, lien, etc)
- ▶ Droits d'accès
- ▶ Propriétaire
- ▶ Nombre de liens matériels (vus ensuite)
- ▶ Adresses des blocs sur le disque

Mais : ne contient pas le nom du fichier

Informations sur un i-node

```
<> stat slides-c02.tex
  File: "slides-c02.tex"
  Size: 16506      Blocks: 40      IO Block: 4096   fichier
Device: 80ah/2058d Inode: 13120462  Links: 1
Access: (0644/-rw-r--r--)
Uid: ( 1000/   thiel)  Gid: ( 1000/   thiel)
Access: 2016-09-19 10:48:21.000000000 +0200
Modify: 2016-09-19 10:48:21.000000000 +0200
Change: 2016-09-19 10:48:21.000000000 +0200
```

Fonctions stat, fstat

Répertoire

Un répertoire = fichier de type répertoire, contenant une liste de couples (i-number, nom de fichier)

```
<> ls -ai
13119235 ./
13119233 ../
13120226 figs/
13120490 slides-c02.pdf
13120462 slides-c02.tex
13119946 svg/
```

→ Notion de **Lien** :

- ▶ lien matériel : sur un i-node
- ▶ lien symbolique : sur un chemin

Lien matériel

Un lien matériel = une entrée dans un répertoire.

Création de liens matériels avec `ln`, suppression avec `rm` :

```
<> touch cible
<> ls -li *cible
13120165 -rw-rw-r-- 1 thiel thiel 0 sept. 23 10:42 cible
<> ln cible liencible
<> ls -li *cible
13120165 -rw-rw-r-- 2 thiel thiel 0 sept. 23 10:42 cible
13120165 -rw-rw-r-- 2 thiel thiel 0 sept. 23 10:42 liencible
<> rm cible
<> ls -li *cible
13120165 -rw-rw-r-- 1 thiel thiel 0 sept. 23 10:42 liencible
```

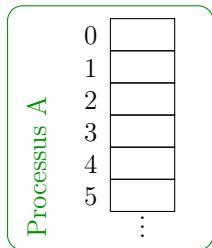
Le i-node stocke le nombre de liens matériels

3 - Les tables du système au niveau 1

Pour gérer les fichiers ouverts.

Les tables de descripteurs

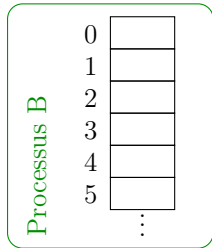
TDs



Chaque processus possède 1 TD

Indice dans cette table = "descripteur de fichier"
`int fd;`

permet au processus de manipuler un fichier ouvert
au niveau 1



3 premiers indices :

- 0 entrée standard
- 1 sortie standard
- 2 sortie d'erreur

Afficher la TD d'un processus

Dans un terminal :

```
cat | sort > /tmp/toto 5> /tmp/tutu
```

Dans un autre terminal :

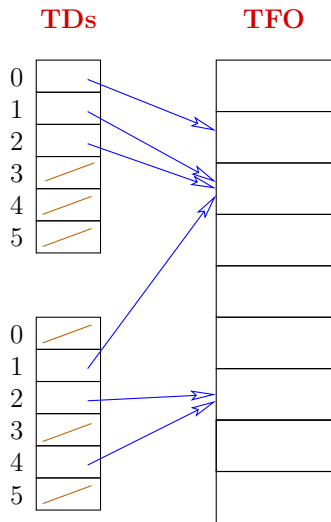
```
<> pidof sort
```

```
652
```

```
<> lsof -p 652 -a -d 0-10
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
sort	652	thiel	0r	FIFO	0,8	0t0	303133	pipe
sort	652	thiel	1w	REG	8,1	0	15991234	/tmp/toto
sort	652	thiel	2u	CHR	136,2	0t0	5	/dev/pts/2
sort	652	thiel	5w	REG	8,1	0	15991235	/tmp/tutu

Table des fichiers ouverts



TD = tableau de struct

Dans chaque case :

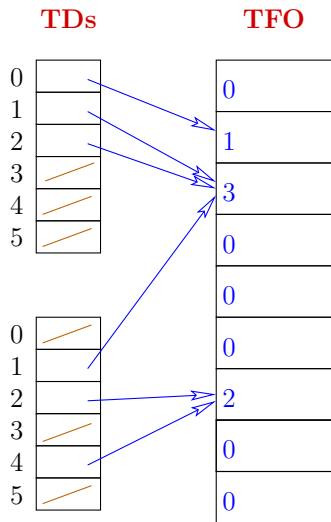
- ▶ pointeur sur une case de TFO, ou NULL (case libre)

TFO = Table des fichiers ouverts

Table unique, gérée par le système

Il peut y avoir plusieurs descripteurs pour un même fichier ouvert.

Cases de TFO

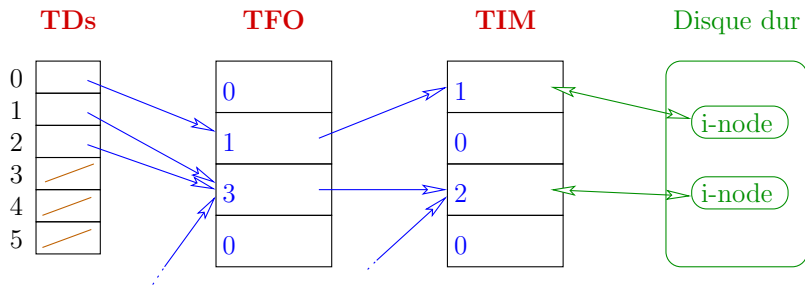


TFO = tableau de struct

Dans chaque case :

- ▶ `ref_counter` : compteur de références
- ▶ mode d'ouverture (lecture, écriture, etc)
- ▶ `offset` : position courante dans le fichier
- ▶ pointeur sur le i-node en mémoire

Table des i-nodes en mémoire



Lorsqu'un fichier est ouvert, le i-node correspondant est chargé dans TIM (s'il n'y est pas déjà)

Chaque case contient

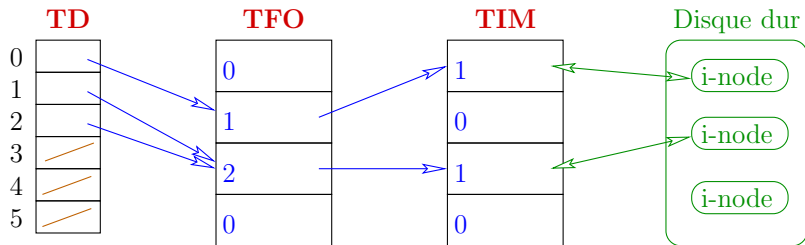
- ▶ Une copie du i-node
- ▶ `ref_counter` : le nombre total d'ouvertures
- ▶ l'état (modifié, verrouillé, etc)

4 - Ouverture et fermeture au niveau 1

Mise à jour des tables par le système.

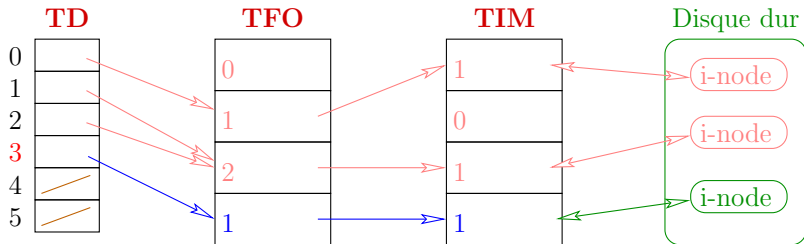
Ouverture d'un fichier

- ▶ Charge éventuellement le i-node dans TIM
- ▶ Alloue une entrée dans TFO
- ▶ Alloue un descripteur dans la TD du processus dans **la première case libre**
- ▶ Renvoie l'indice $fd \geq 0$ ou -1 erreur (cf `errno`)



Ouverture d'un fichier

- ▶ Charge éventuellement le i-node dans TIM
- ▶ Alloue une entrée dans TFO
- ▶ Alloue un descripteur dans la TD du processus dans **la première case libre**
- ▶ Renvoie l'indice $fd \geq 0$ ou -1 erreur (cf `errno`)



Ouverture de fichier

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

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

flags : union binaire de constantes C1 | C2 | ...

O_RDONLY	en lecture seule	
O_WRONLY	en écriture seule	1 de ces 3
O_RDWR	en lecture écriture	

O_TRUNC	vide le fichier	
O_CREAT	créé un fichier vide	

→ utiliser 3^e paramètre mode = droits en octal

Ouverture de fichier

<code>O_APPEND</code>	écriture en fin de fichier
<code>O_NONBLOCK</code>	rend ouverture non bloquante (tubes nommés)
<code>O_NDELAY</code>	rend lectures/écritures non bloquantes

Exemple :

```
int fd = open ("tmp.txt", //en octal
               O_WRONLY|O_TRUNC|O_CREAT, 0644);
if (fd < 0) { perror("open"); exit(1); }
```

Fermeture d'un fichier

Ferme le descripteur `fd` du processus : `close`

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

Renvoie 0 succès, ou -1 erreur (cf `errno`)

Effets en cascade :

Case TD fermée

ref_counter décrémenté dans case TFO

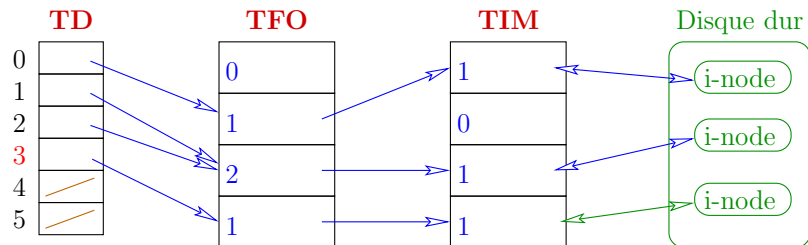
si ref_counter == 0

→ case TFO libérée

ref_counter décrémenté dans case TIM

si ref_counter == 0

→ case TIM libérée



Effets en cascade :

Case TD fermée

ref_counter décrémenté dans case TFO

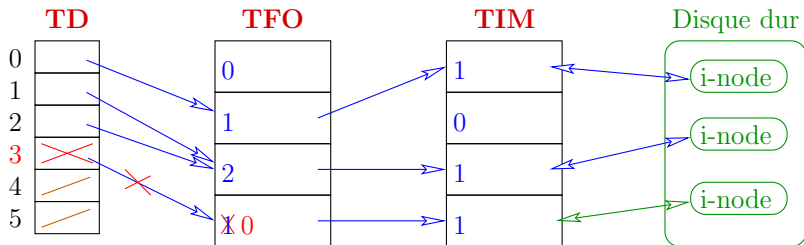
si ref_counter == 0

→ case TFO libérée

ref_counter décrémenté dans case TIM

si ref_counter == 0

→ case TIM libérée



Effets en cascade :

Case TD fermée

ref_counter décrémenté dans case TFO

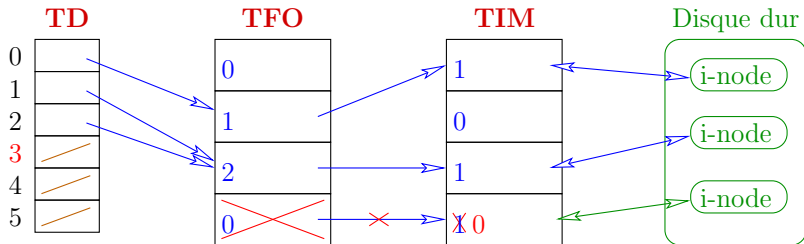
si ref_counter == 0

→ case TFO libérée

ref_counter décrémenté dans case TIM

si ref_counter == 0

→ case TIM libérée



Effets en cascade :

Case TD fermée

ref_counter décrémenté dans case TFO

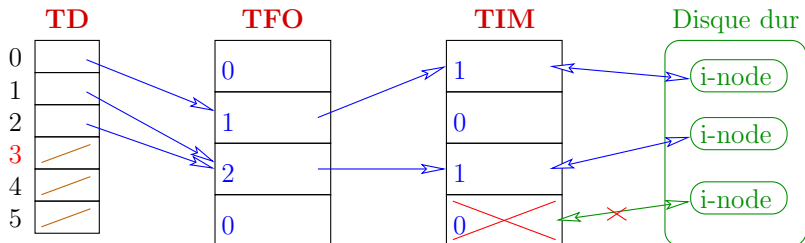
si ref_counter == 0

→ case TFO libérée

ref_counter décrémenté dans case TIM

si ref_counter == 0

→ case TIM libérée



Suppression physique du i-node

Si le i-node n'est (pas) plus en mémoire,
et si le compteur de liens matériels du i-noeud est = 0
alors le fichier est physiquement supprimé du disque
= le i-node et les blocs sont libérés.

Pour décrémenter le compteur de liens matériels :

commande `rm`

```
#include <unistd.h>
int unlink(const char *pathname);
```

5 - Lecture et écriture de fichiers

Au niveau 1.

Écriture dans un fichier

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

fd descripteur ouvert en écriture

buf adresse base zone mémoire

count nombre d'octets à écrire

Renvoie -1 : erreur

> 0 nombre d'octets écrits (\leq count)

0 ssi count = 0 (et pas d'erreur)

Exemple d'écriture

```
char *s = "bonjour";  
int r = write (1, s, strlen(s));  
if (r < 0) perror ("write");
```

Chaîne "bien formée" : avec '\0' terminal

write n'écrit pas le '\0' terminal de la chaîne s

Lecture dans un fichier

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

fd descripteur ouvert en lecture

buf adresse base zone mémoire

count nombre d'octets à lire au plus et à mémoriser dans buf

Renvoie -1 : erreur

 > 0 nombre d'octets lus (\leq count)

 0 si count = 0 ou si fin de fichier atteinte

(EOF uniquement niveau 2)

Exemple de lecture

```
char s[100];  
int r = read (0, s, sizeof(s)-1);  
if (r < 0) { perror("read"); exit (1); }  
s[r] = 0; // Rajoute '\0' terminal
```

Importance de rajouter le '\0' terminal pour avoir une chaîne régulière → printf, strcmp, etc

Bon usage avec le '\0' terminal :

```
write (1, s, strlen(s));  
read (0, s, sizeof(s)-1);
```

6 - Les tubes

Mécanisme efficace de communication.

Les tubes

Mécanisme de communication entre processus appartenant au système de fichiers

Un tube correspond à un i-node

- ▶ dans le disque physique : tube nommé
- ▶ dans le disque logique des tubes : tube anonyme

Désigné par descripteurs de fichiers

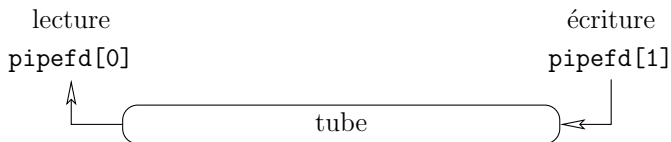
→ manipulé par `read`, `write`, `close`

Création d'un tube anonyme

```
#include <unistd.h>  
int pipe(pipefd[2]);
```

Crée un tube et mémorise les extrémités dans pipefd

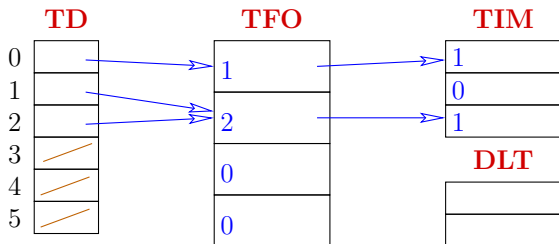
Renvoie 0 succès, -1 erreur



Effet de pipe dans les tables

```
int pipe(int pipefd[2]);
```

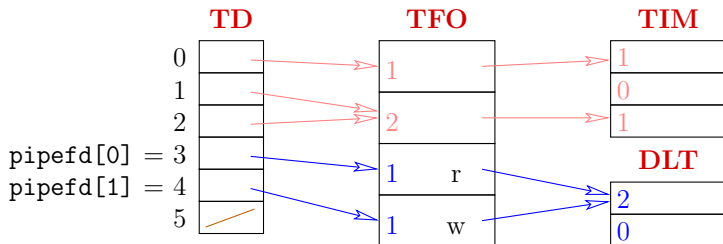
- ▶ Alloue un i-node sur le Disque Logique des Tubes
- ▶ crée 2 entrées dans TFO (1 en lecture et 1 en écriture)
- ▶ alloue 2 descripteurs dans la TD du processus appelant :
pipefd[0] en lecture, pipefd[1] en écriture



Effet de pipe dans les tables

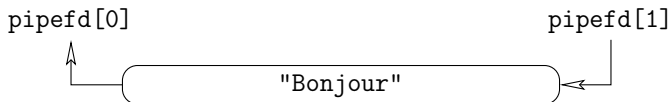
```
int pipe(int pipefd[2]);
```

- ▶ Alloue un i-node sur le Disque Logique des Tubes
- ▶ crée 2 entrées dans TFO (1 en lecture et 1 en écriture)
- ▶ alloue 2 descripteurs dans la TD du processus appelant :
pipefd[0] en lecture, pipefd[1] en écriture

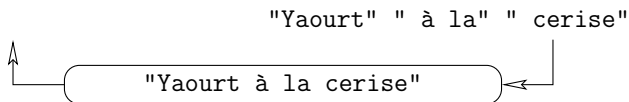


Propriétés des tubes

- ▶ Gestion FIFO (First In First Out) : caractères lus dans l'ordre où ils ont été écrits

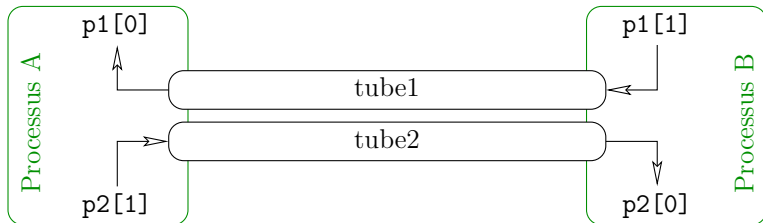


- ▶ Lecture destructrice : 1 caractère ne peut être lu qu'une fois
- ▶ Vu comme un flux continu de caractères : les caractères sont "concaténés" en écriture



Propriétés des tubes

- ▶ Les tubes sont unidirectionnels → 2 tubes pour dialoguer



- ▶ Capacité finie (Linux : 64k)
- ▶ Un tube peut être plein → écriture bloquante
- ▶ Un tube peut être vide → lecture bloquante

Lecteurs et écrivains

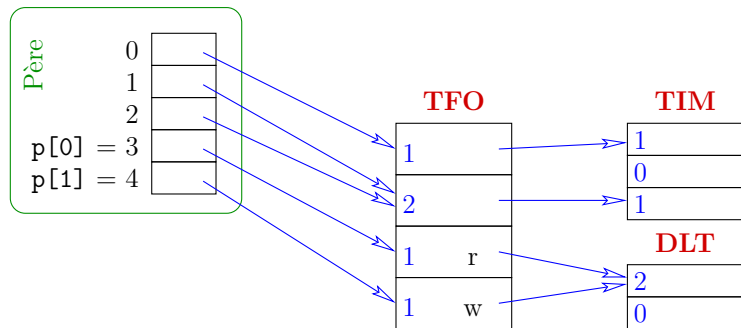
Un processus qui possède un descripteur du tube :

- ▶ en lecture est un lecteur ;
- ▶ en écriture est un écrivain

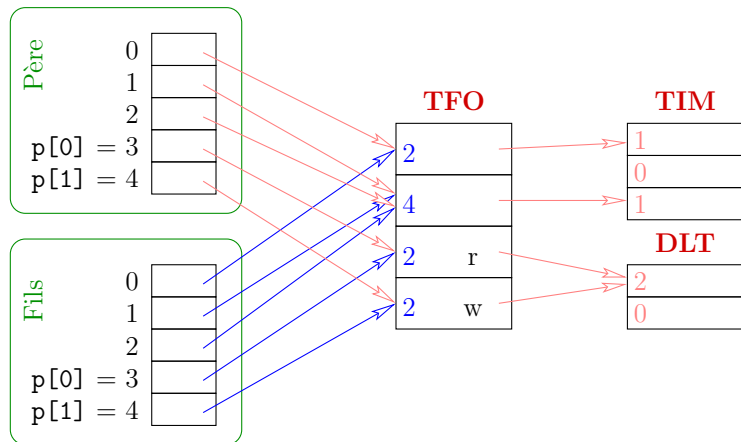
Un tube peut avoir plusieurs lecteurs ou écrivains :

- ▶ Duplication avec `dup` (vue plus tard)
- ▶ Recopie par héritage avec `fork`


Recopie par héritage avec fork



Recopie par héritage avec fork



Nombre de lecteurs et d'écrivains

- Si nombre de lecteurs = 0
 - ▶ Interdit toute écriture
 - ▶ Envoi de SIGPIPE si write  Signal mortel (message "broken pipe")
 - Si nombre d'écrivains = 0 et si le tube est vide
 - ▶ read renvoie 0 → Fin de fichier atteinte
- Il faut fermer tous les écrivains pour détecter la fin de fichier

Règle :

Ne conserver que les descripteurs utiles ;
fermer systématiquement tous les autres dès que possible.

Sous Windows

Les fichiers sont manipulés par des `Handle`, comme tous les objets du système.

Les tubes anonymes existent de façon équivalente :

- ▶ Créés par `CreatePipe`
- ▶ Lecture et écriture avec `ReadFile` et `WriteFile`
- ▶ Les handles d'un tube sont hérités à la création d'un fils.