UNIVERSITÉ PARIS SUD

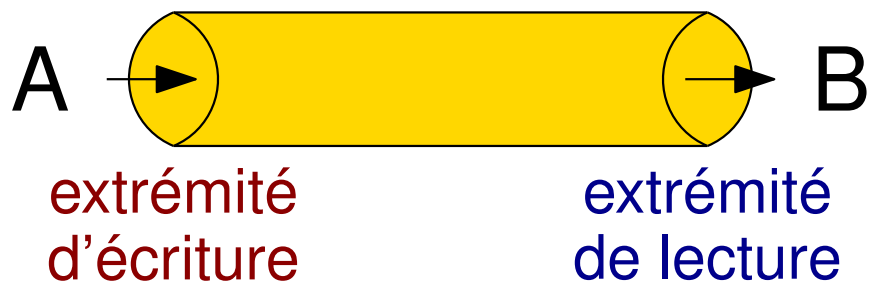
IUT D'ORSAY

M3101 · Principes des systèmes d'exploitation

Les tubes

Tube (*pipe*) un canal de communication entre les processus

- un flux d'octets entre l'entrée et la sortie
- accessible via deux descripteurs de fichier
- un tampon système de capacité limitée
- une donnée lue disparaît du tube
- anonyme — passe du père au fils



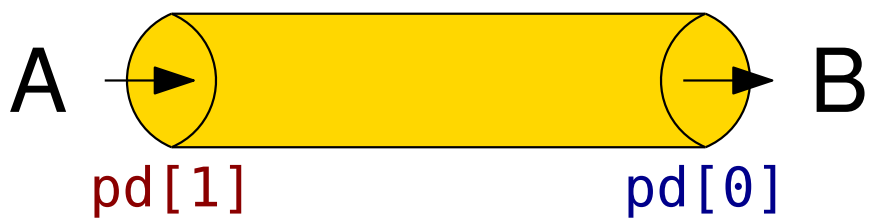
Créer un tube

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

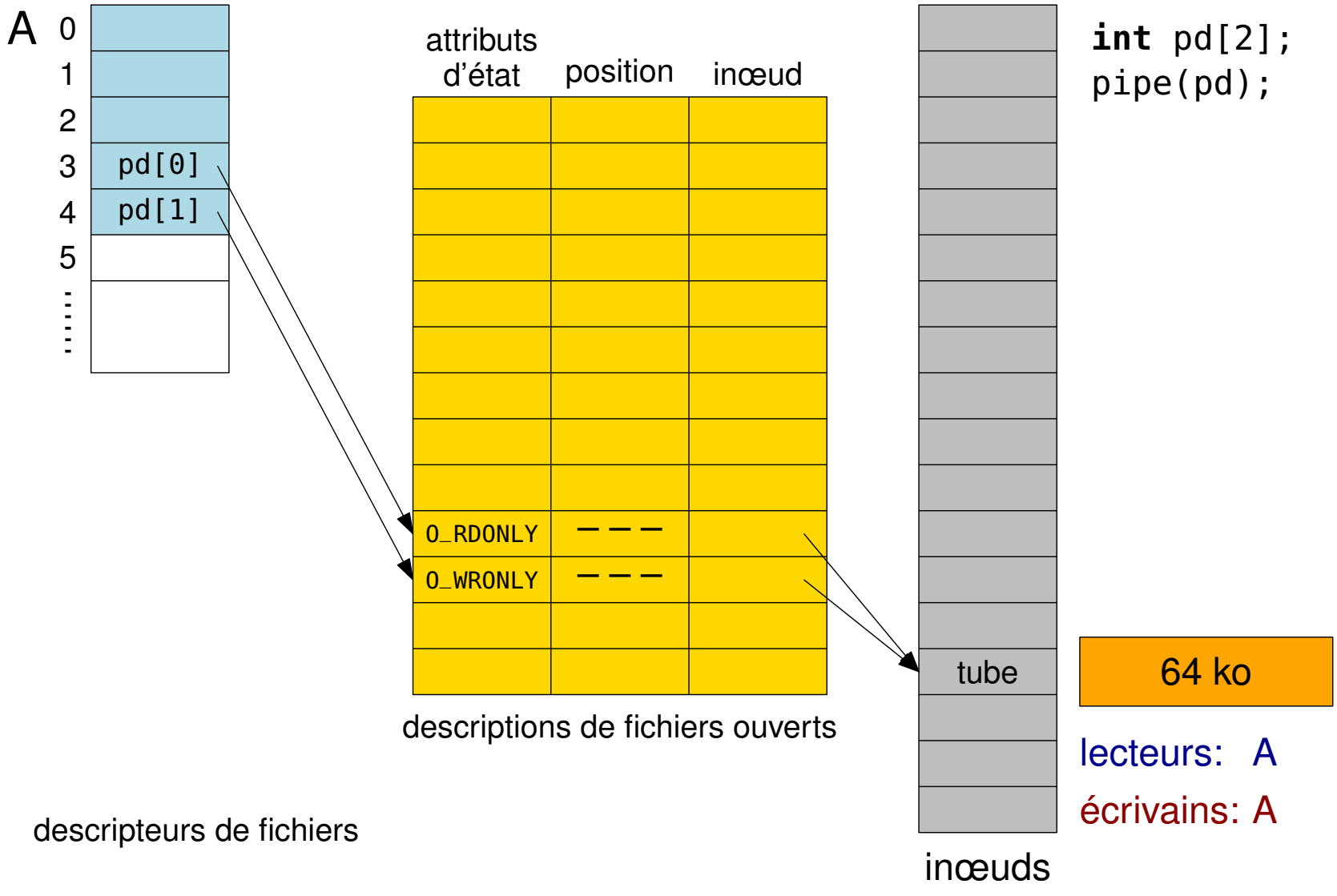
Crée un nouveau tube et place les descripteurs dans `pd`

- `pd[0]` — l'extrémité de lecture
- `pd[1]` — l'extrémité d'écriture

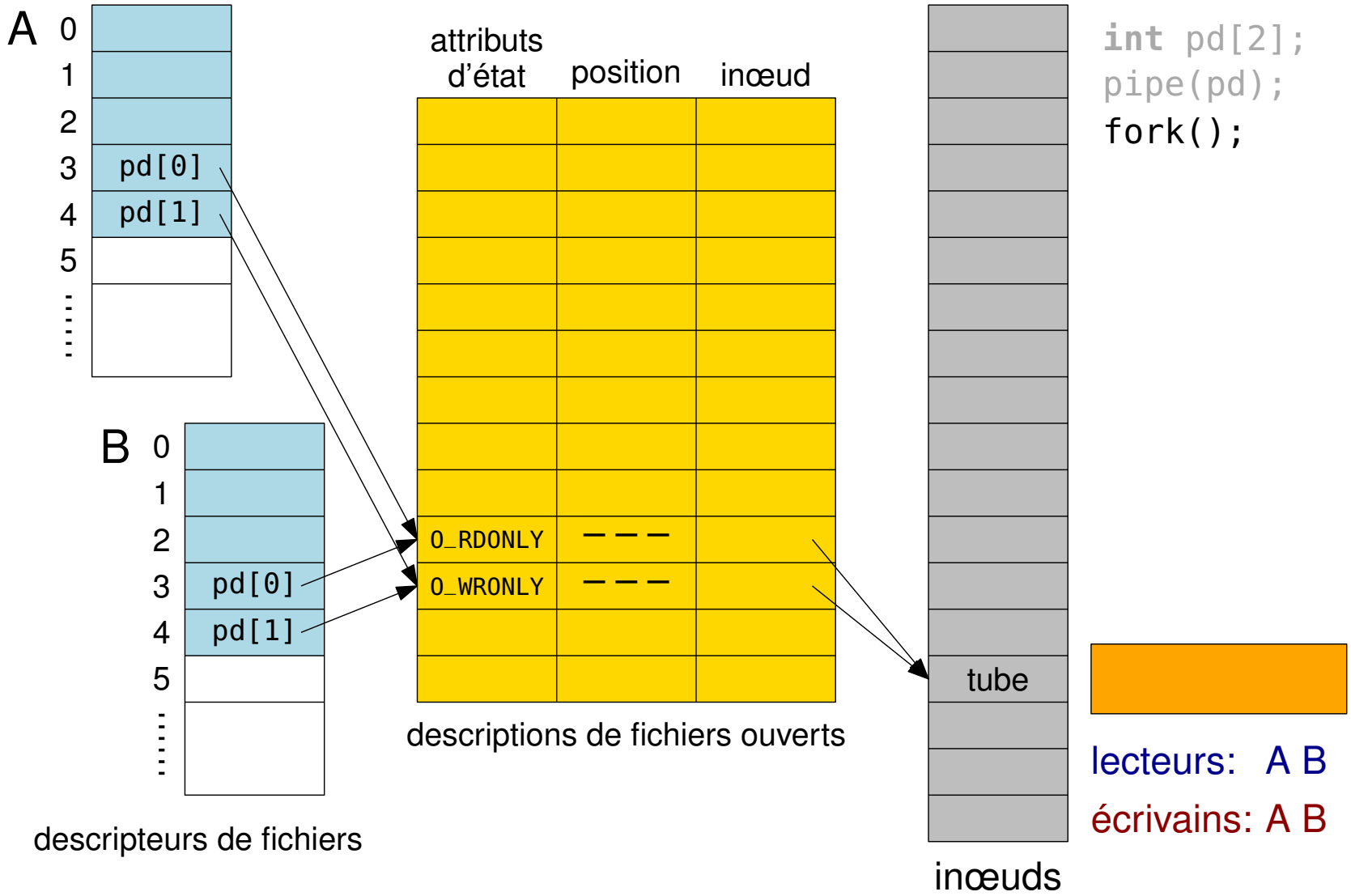
Renvoie 0 en cas de succès et -1 en cas d'erreur



Créer un tube



Créer un tube



Lecteurs et écrivains

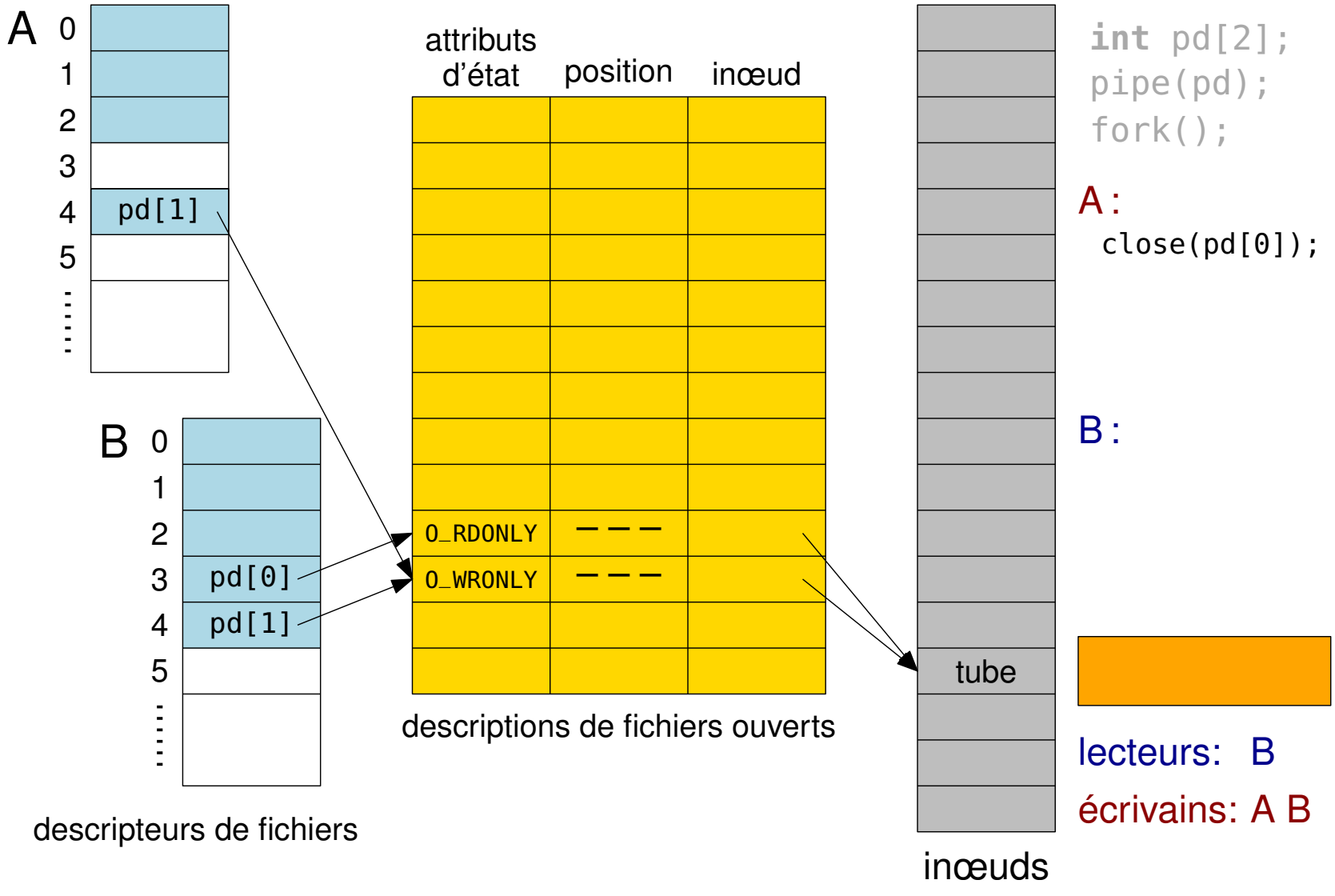
Les descripteurs d'un tube sont hérités par les processus fils.

lecteur tout processus qui possède un descripteur de lecture

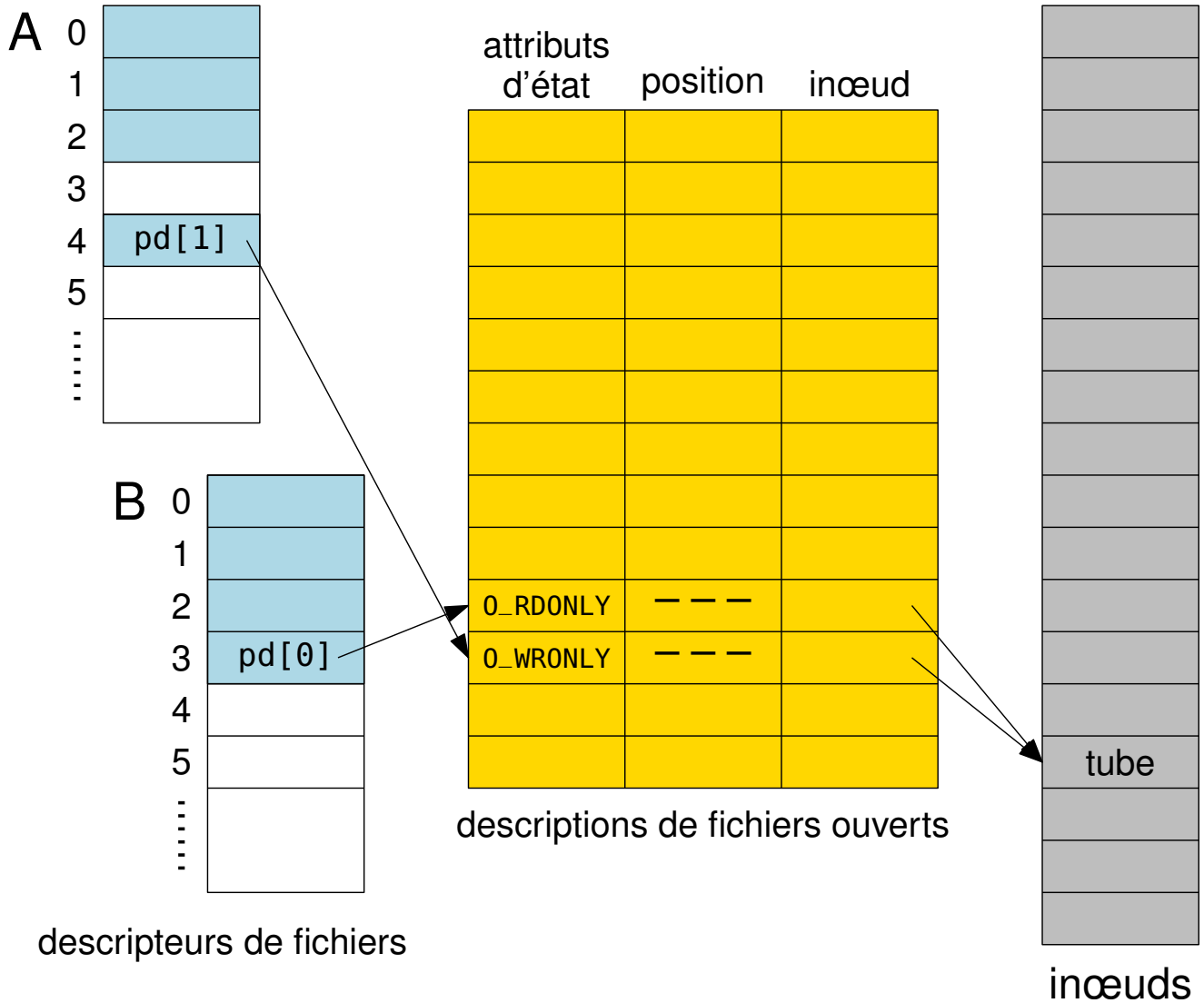
écrivain tout processus qui possède un descripteur d'écriture

Quand il n'y a plus de lecteurs ni écrivains, le tube est détruit.

Lecteurs et écrivains



Lecteurs et écrivains



Lire depuis un tube

```
int read(pd[0], const void *tampon, size_t nbOct);
```

Renvoie le nombre d'octets lus ($\leqslant$ `nbOct`) ou -1 en cas d'erreur

Si le tampon système du tube est vide
$\Rightarrow$ `read()` se bloque jusqu'à ce qu'un écrivain y mette des données

Si le tube est vide et il n'y a plus de processus écrivains
$\Rightarrow$ `read()` renvoie 0 (*end-of-file*)

Écrire dans un tube

```
int write(pd[1], const void *tampon, size_t nbOct);
```

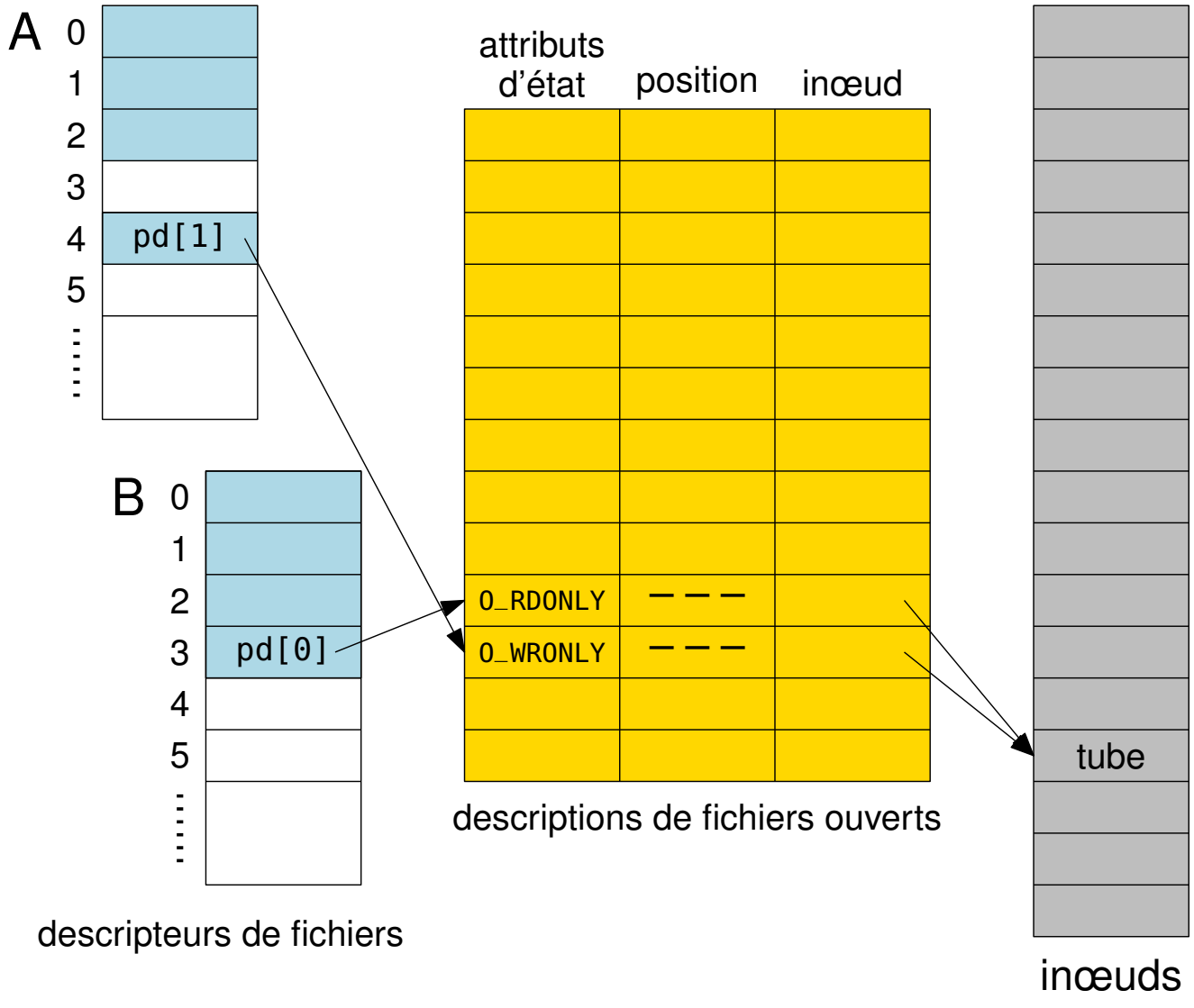
Renvoie le nombre d'octets écrits ou -1 en cas d'erreur

Si le tampon système du tube est plein ($\leqslant$ `nbOct` octets disponibles)
$\Rightarrow$ `write()` se bloque jusqu'à ce qu'un lecteur décharge le tube

S'il n'y a plus de processus lecteurs
$\Rightarrow$ le processus est tué par `SIGPIPE`

S'il n'y a plus de lecteurs et `SIGPIPE` est bloqué, ignoré ou capté
$\Rightarrow$ `write()` renvoie -1

Lecture et écriture




```
int pd[2];
pipe(pd);
fork();
```

A :

```
close(pd[0]);
write(pd[1],
      buf,len);
```

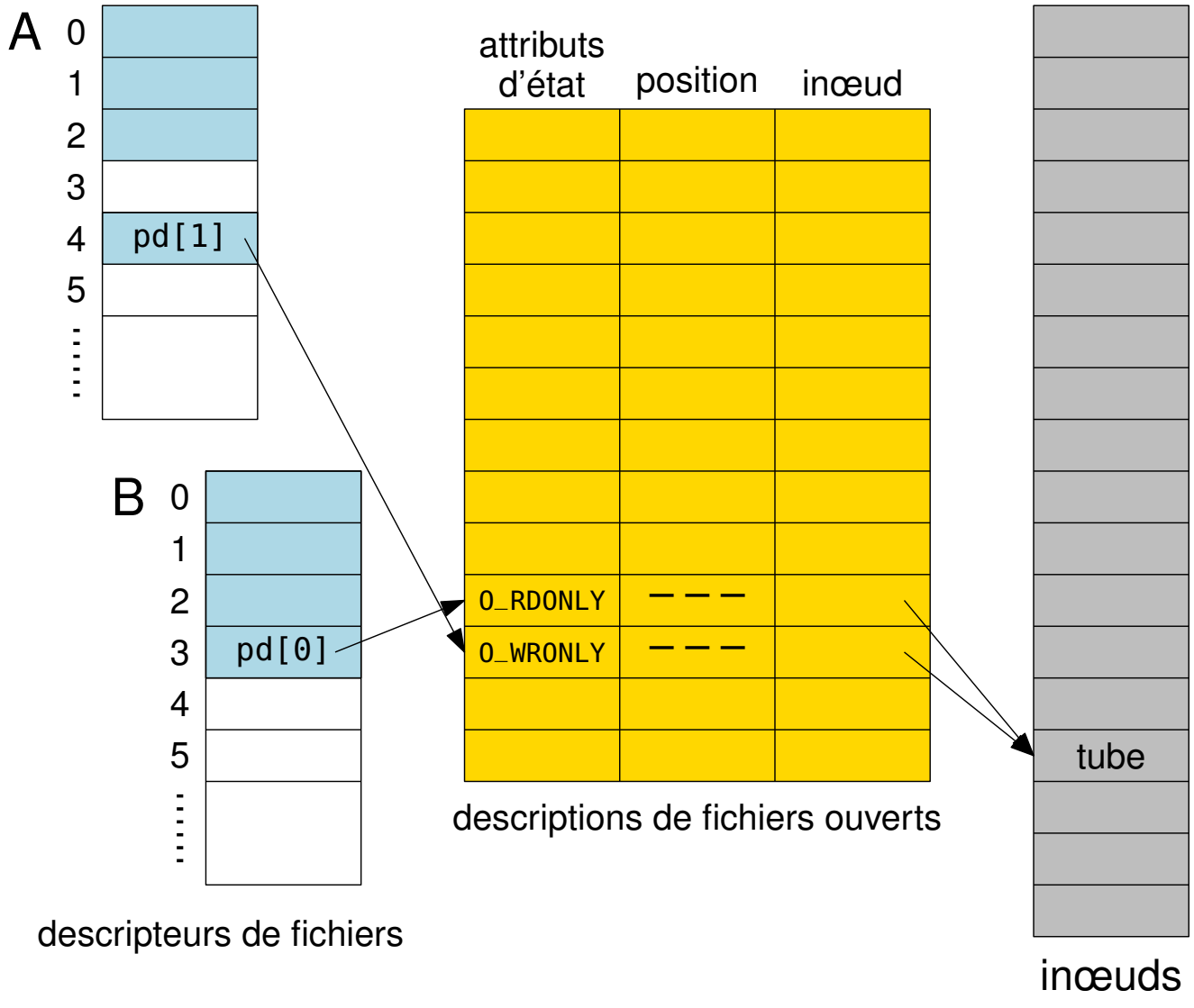
B :

```
close(pd[1]);
```

lecteurs: B

écrivains: A

Lecture et écriture




```
int pd[2];
pipe(pd);
fork();
```

A :

```
close(pd[0]);
write(pd[1],
      buf,len);
```

B :

```
close(pd[1]);
read(pd[0],
        &c,1);
```

len-1

lecteurs: B

écrivains: A

Dupliquer un descripteur de fichier

```
int dup(int ancien_fd);
```

- trouve le plus petit descripteur nouveau_fd disponible
- fait nouveau_fd une copie de ancien_fd

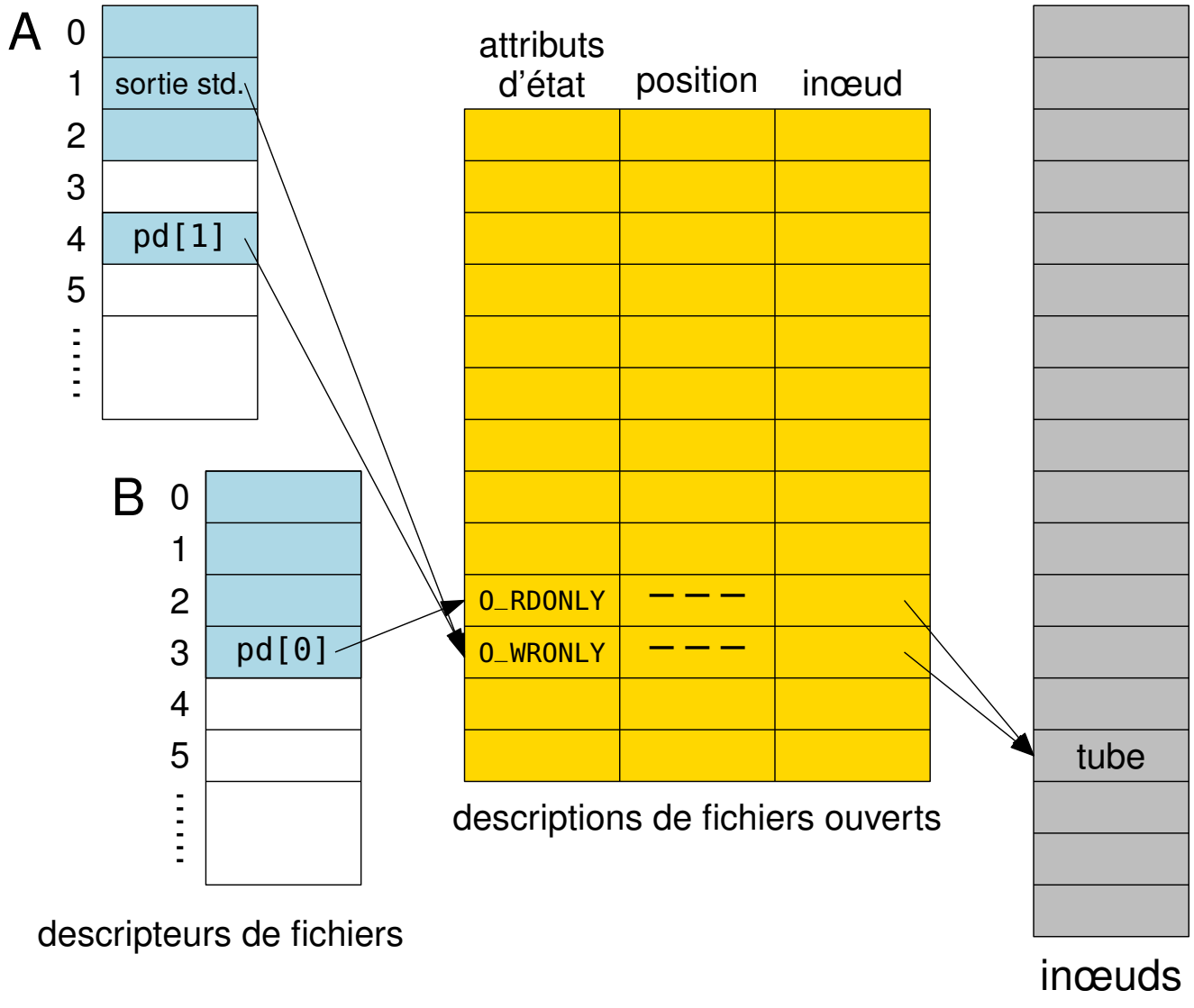
```
int dup2(int ancien_fd, int nouveau_fd);
```

- ferme nouveau_fd s'il est ouvert
- fait nouveau_fd une copie de ancien_fd

Renvoient nouveau_fd ou -1 en cas d'erreur

nouveau_fd et ancien_fd pointent sur la même description de fichier ouvert

Dupliquer un descripteur de fichier




```
int pd[2];
pipe(pd);
fork();
```

A :

```
close(pd[0]);
dup2(pd[1],1);
```

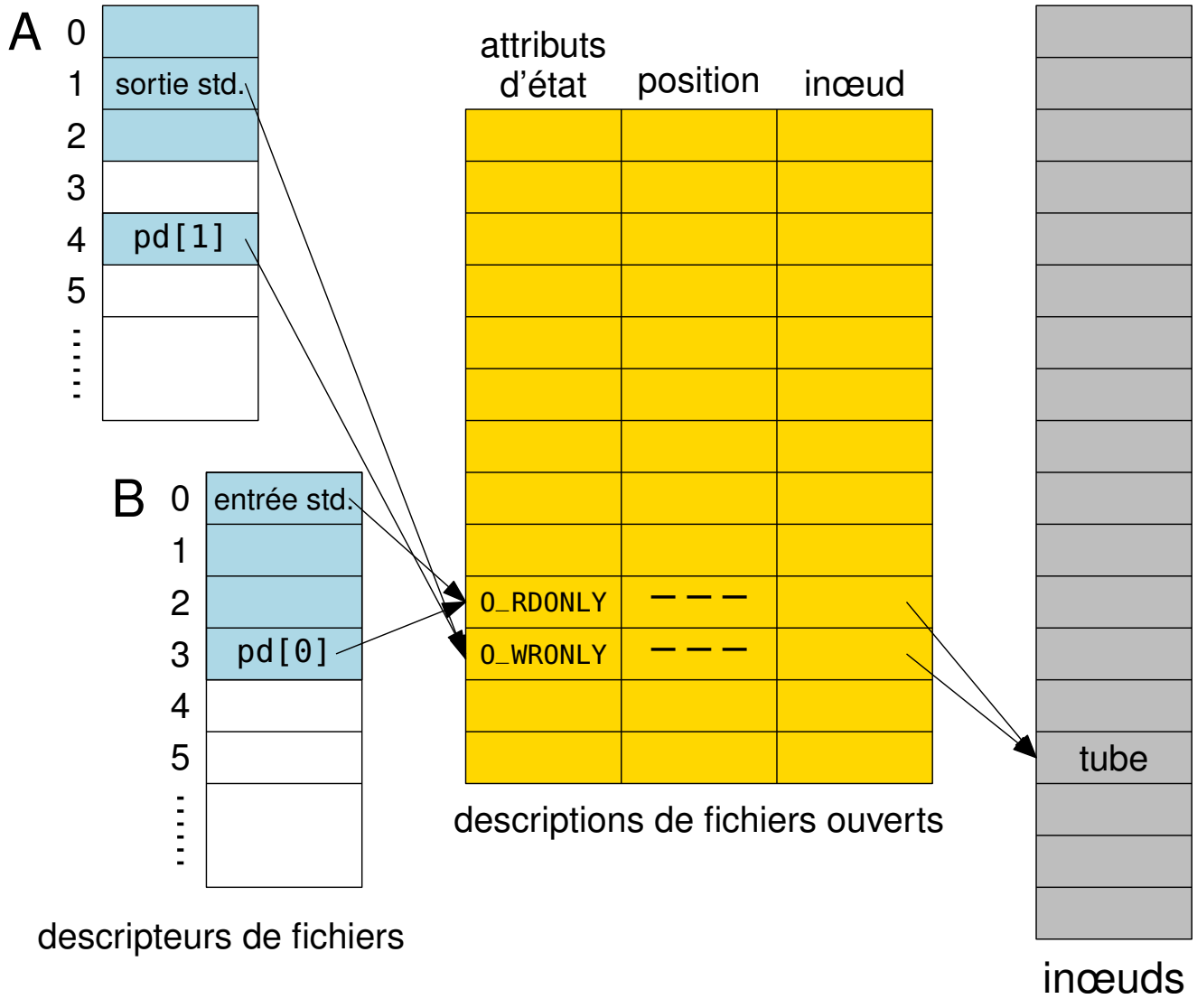
B :

```
close(pd[1]);
```

lecteurs: B

écrivains: A

Dupliquer un descripteur de fichier




```
int pd[2];
pipe(pd);
fork();
```

A :

```
close(pd[0]);
dup2(pd[1],1);
```

B :

```
close(pd[1]);
dup2(pd[0],0);
```

lecteurs: B

écrivains: A

Exécuter un programme

```
int  execl(const char *path, const char *arg0,..., NULL);
```

```
int execlp(const char *path, const char *arg0,..., NULL);
```

Exécute un fichier dans le processus courant

- `path` — chemin vers l'exécutable ou le nom de commande
- `arg0` — paramètre 0 (le nom de commande)
- `arg1`, ... — autres paramètres
- `NULL` — fin de la liste des paramètres

Ne retourne pas en cas du succès, renvoie -1 en cas d'échec

Le code et les données sont remplacés par le nouveau programme

Préserve les fichiers ouverts (dont les fichiers standards, les tubes, etc.)

Exemple : `ls -1 | wc -l`

```
1  int main() {
2    int pd[2];
3    pipe(pd);
4    if (fork() == 0) { // PARTIE GAUCHE
5      close(pd[0]);    // fermer l'extrémité de lecture non-utilisée
6      dup2(pd[1], 1);  // dupliquer l'extrémité d'écriture en stdout
7      close(pd[1]);    // fermer la première extrémité d'écriture
8      execl("/bin/ls", "ls", "-1", NULL);  // exécuter "ls -1"
9      return 1;        // exec() ne retourne qu'en cas d'échec
10   }
11   if (fork() == 0) { // PARTIE DROITE
12     close(pd[1]);    // fermer l'extrémité d'écriture non-utilisée
13     dup2(pd[0], 0);  // dupliquer l'extrémité de lecture en stdin
14     close(pd[0]);    // fermer la première extrémité de lecture
15     execl("/usr/bin/wc", "wc", "-l", NULL);  // exécuter "wc -l"
16     return 1;        // exec() ne retourne qu'en cas d'échec
17   }
18   close(pd[0]); close(pd[1]);  // fermer les deux extrémités
19   wait(NULL); wait(NULL);      // attendre la fin des fils
20   return 0; }
```