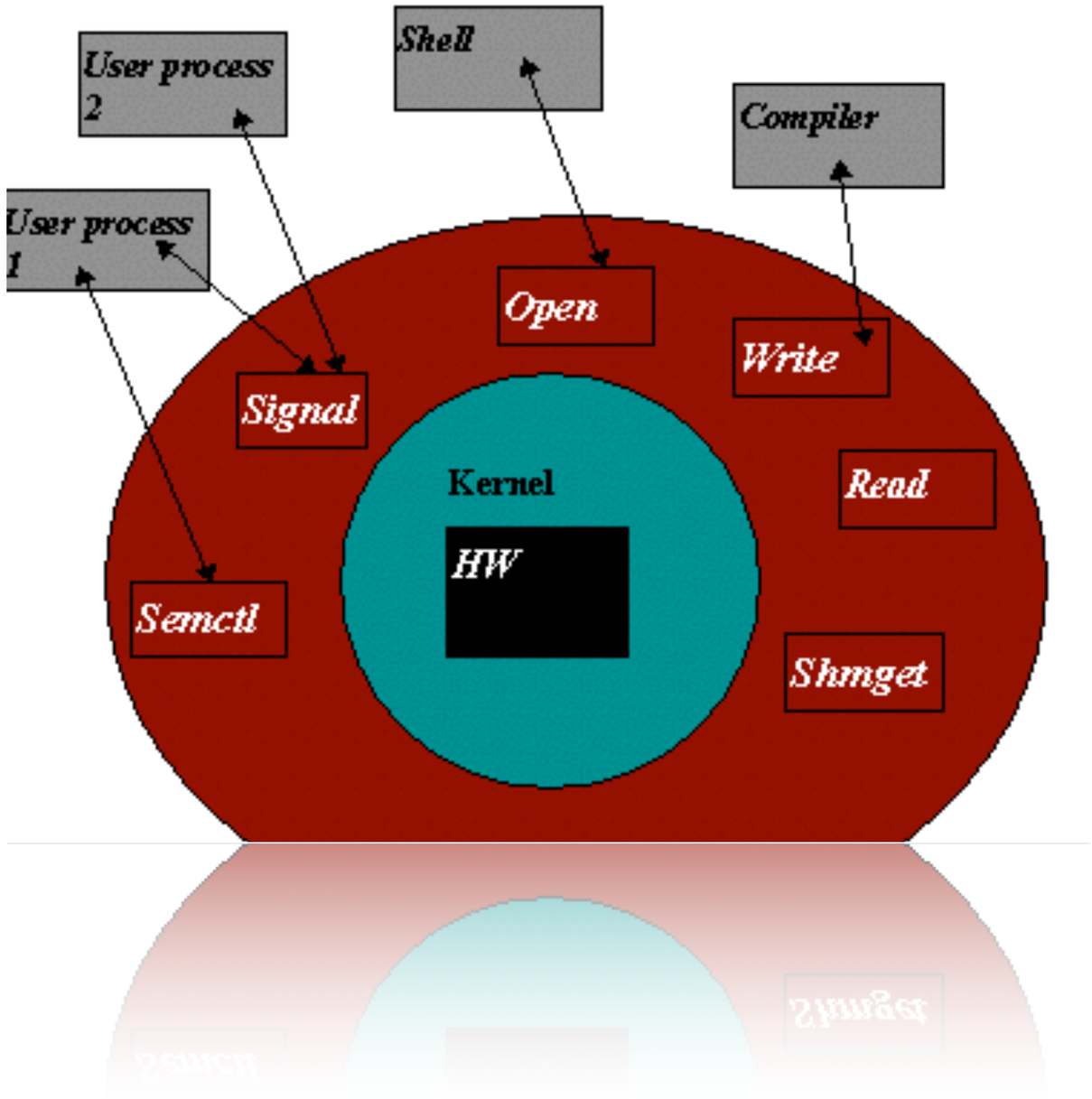
ITC313 - Informatique

Partie Programmation Système

Benoît DARTIES

Benoit.Darties@u-bourgogne.fr



Protection et appels systèmes

Accès au matériel

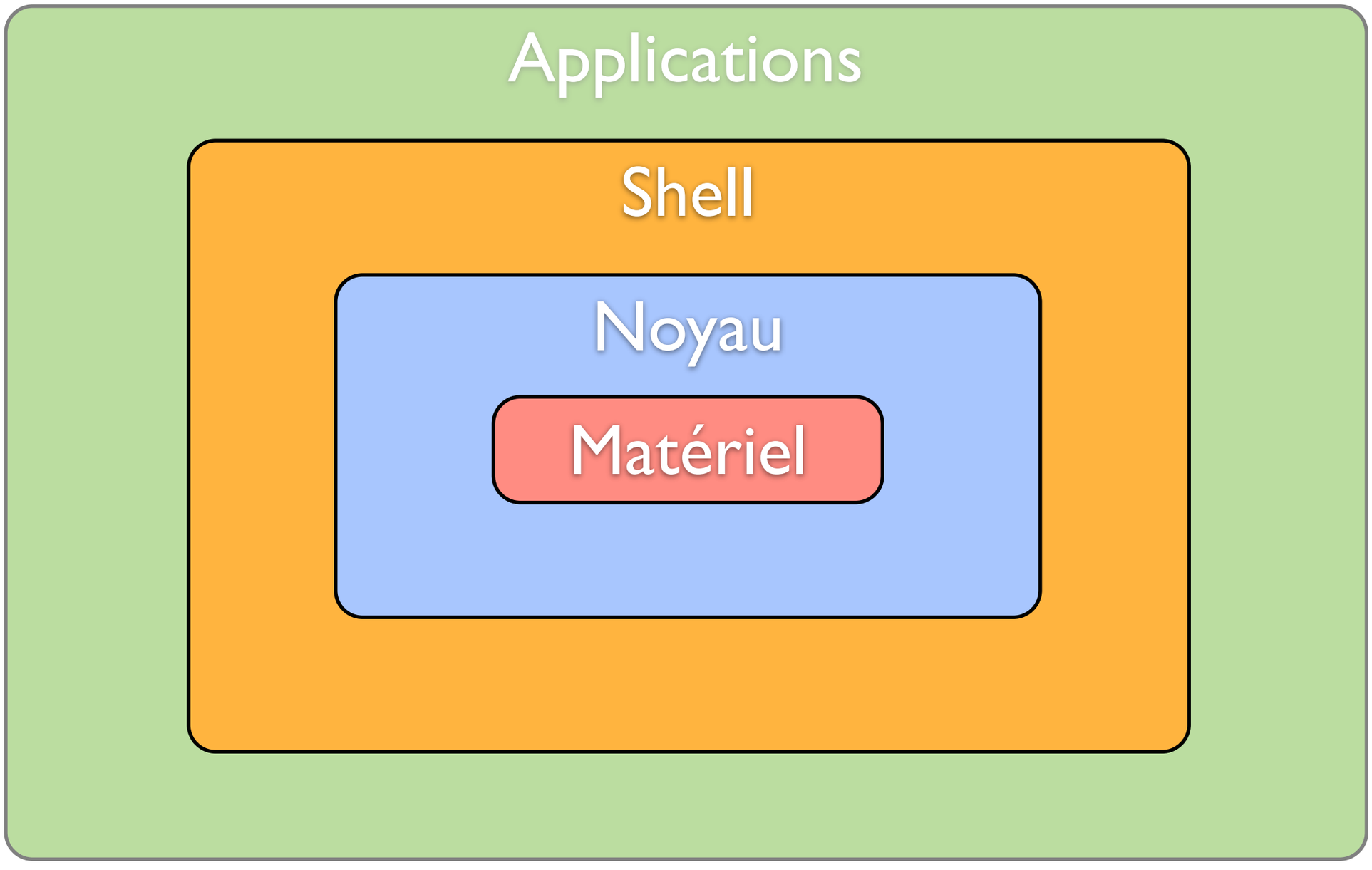
- Le processeur communique :
  - avec la mémoire principale
    - code à exécuter
    - données
  - avec le matériel
    - périphériques de stockage
    - périphériques d'entrée sorties

- Pour le programmeur :
  - Besoin de facilités d'accès au matériel

Rôle du système d'exploitation

- Rôle du système d'exploitation :
  - **Protection du matériel**
    - Empêcher le code utilisateur d'accéder directement au matériel
  - **Protection de la mémoire**
    - Empêcher le code utilisateur d'accéder / altérer les zones mémoires réservées aux programmes des autres utilisateurs.
  - **Noyau** : couche intermédiaire située entre le logiciel utilisateur et le matériel. Passage obligé pour tout processus souhaitant accéder au matériel

Rôle du système d'exploitation



Rôle du système d'exploitation

- Protection du matériel au niveau du noyau
  - deux types de codes :
  - code privilégié, ou code noyau : droit d'accès au matériel
  - code non privilégié, ou code utilisateur

- Accès d'un code utilisateur à un matériel
  - le code utilisateur invoque un code privilégié
  - Accès contrôlé par un système de protection
  - Accès autorisé : l'exécution se déroule
  - Accès non autorisé : système de protection passe la main au noyau qui tue le processus en cause

Appel systèmes

- Invocation de code privilégié par code utilisateur
  - se fait au moyen d'un **appel système**
- Appel système :
  - Fonction fournie par le noyau
  - permet de contrôler de manière sécurisée les applications exécutées dans l'espace utilisateur.
  - fait basculer le processeur en mode superviseur
  - appel système atomique
  - sur la majorités des systèmes : peuvent être utilisés comme de simples fonctions écrites en C
  - Linux : plus de 200 appels systèmes distincts

Appel systèmes

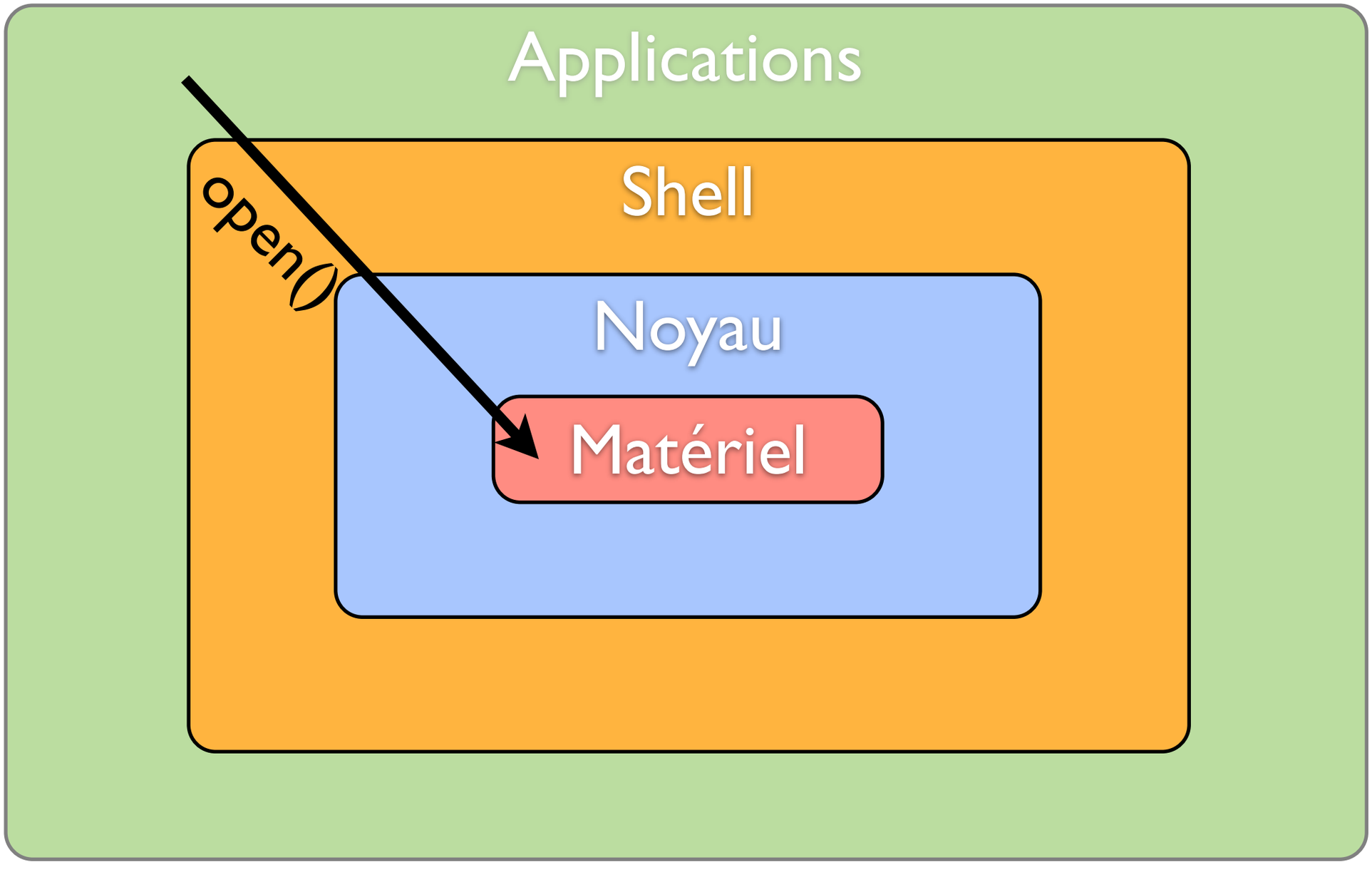
- Commandes UNIX et appels systèmes :
  - utilisent zéro, un ou plusieurs appels systèmes
  - présents dans le code du programme
  - s'utilisent comme des fonctions : requièrent des paramètres
  - parfois nom de l'appel système = nom de la commande
    - ambiguïtés sur le nom, exemple : **chmod**
  - appels systèmes : section 2 du manuel
    - sans précision, les pages du manuel affiché sont les premières trouvées : ordre selon les numéros de section
    - **man chmod** : manuel de la commande
    - **man 2 chmod** : manuel de l'appel système

Quelques appels systèmes

- Exemple : gestion des fichiers

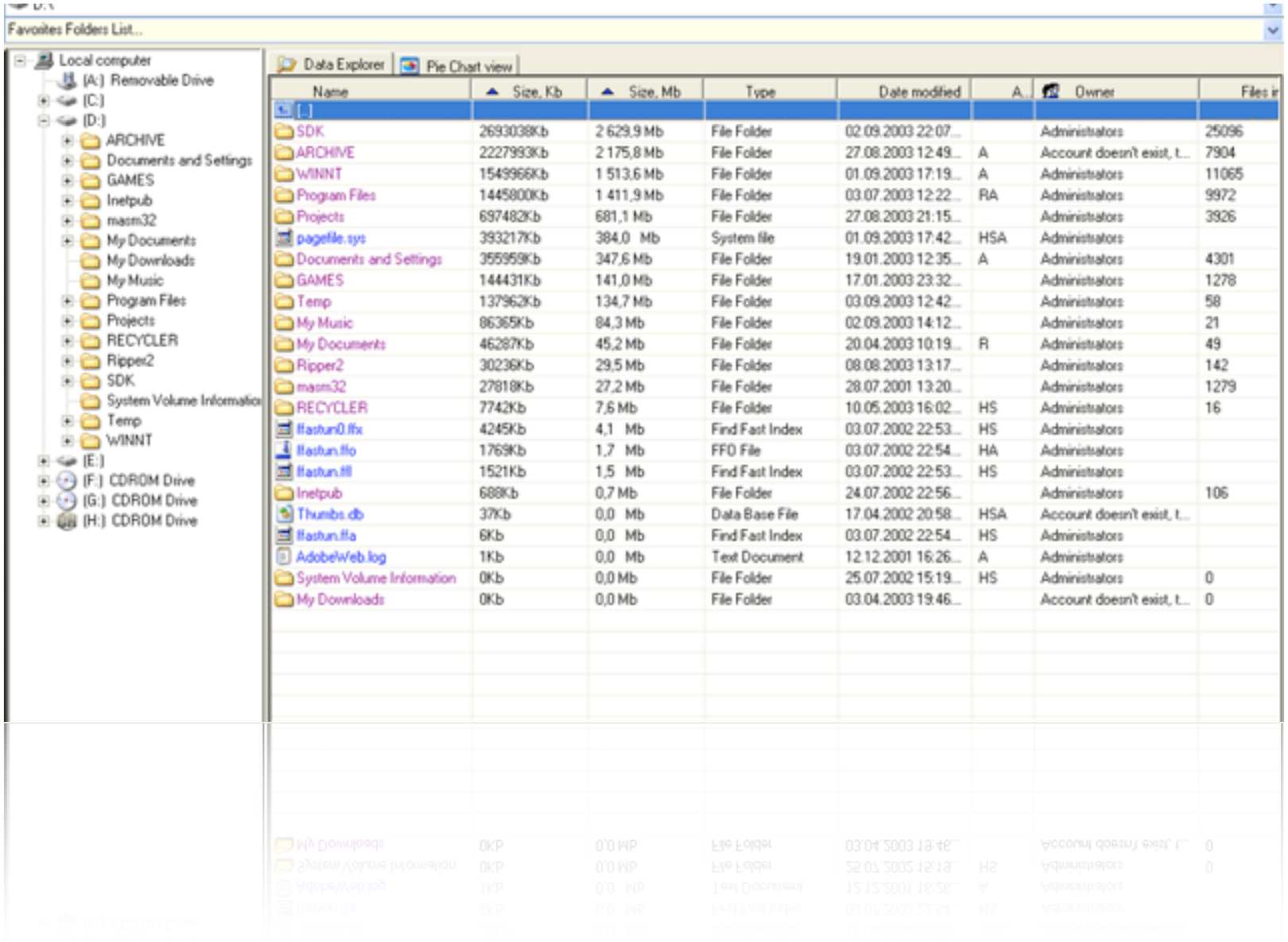
Appel :	Rôle
link	création d'un lien physique
unlink	suppression d'un lien physique (commande **rm**)
chmod	modification des droits d'un fichier
chown	modification du propriétaire d'un fichier
creat	création d'un fichier
open	ouverture d'un fichier en mode lecture
close	fermeture d'un fichier
read	lecture de données depuis un fichier
write	écriture de données vers un fichier
lseek	positionnement de la tête de lecture dans le fichier

Rôle du système d'exploitation



Appel systèmes

- Note à l'attention des programmeurs
  - Appel système : fonction-clé
  - Certaines fonctions qui ne sont PAS des appels systèmes permettent d'utiliser de manière simplifiée ces appels systèmes : interfaces d'accès
  - Contenues dans les librairies C
  - section 3 du manuel



Gestion des fichiers par un processus

Fichiers et processus

- Interaction Processus - Fichier
  - Fichier : suite d'octets
  - Principale interaction
    - lecture d'octets depuis un fichier
    - écriture d'octets vers un fichier
  - Sous UNIX : les périphériques sont vus comme des fichiers

- Interface système pour l'utilisation de fichiers
  - Un fichier est représenté par un descripteur :
    - petit entier interne au programme
    - référence une entrée de la ***table des fichiers ouverts***

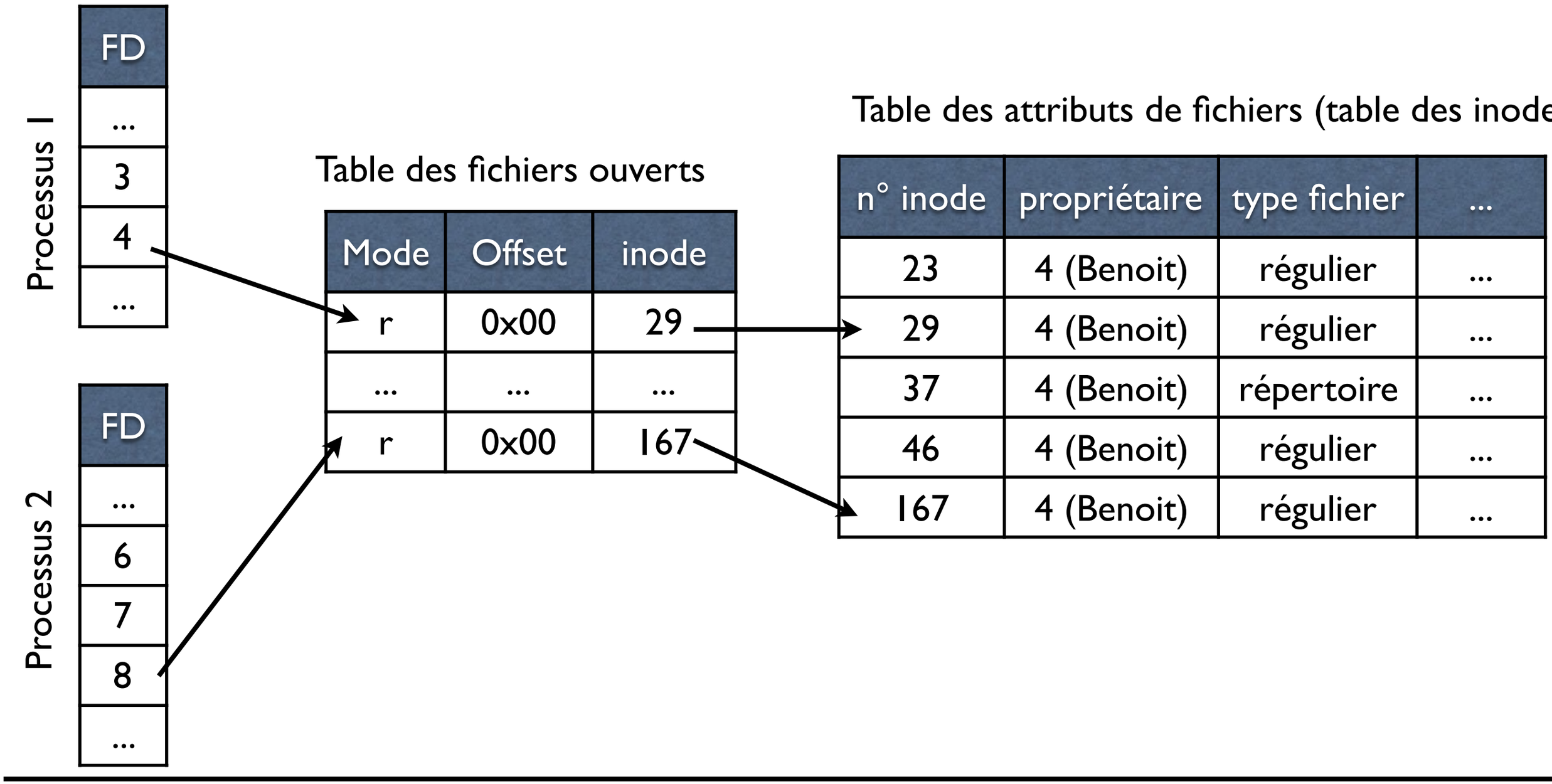
Fichiers et processus

- Table des fichiers ouverts
  - Table présente dans le noyau
  - Une entrée décrit un descripteur
    - Mode d'ouverture du fichier : écriture, lecture
    - Position de lecture
    - Pointeur vers la table des inodes

- Table des descripteurs
  - Une table par processus
  - Contient l'ensemble des descripteurs
  - Un descripteur désigne une entrée de la table des fichiers ouverts

Fichiers et processus

- Interface d'accès aux fichiers

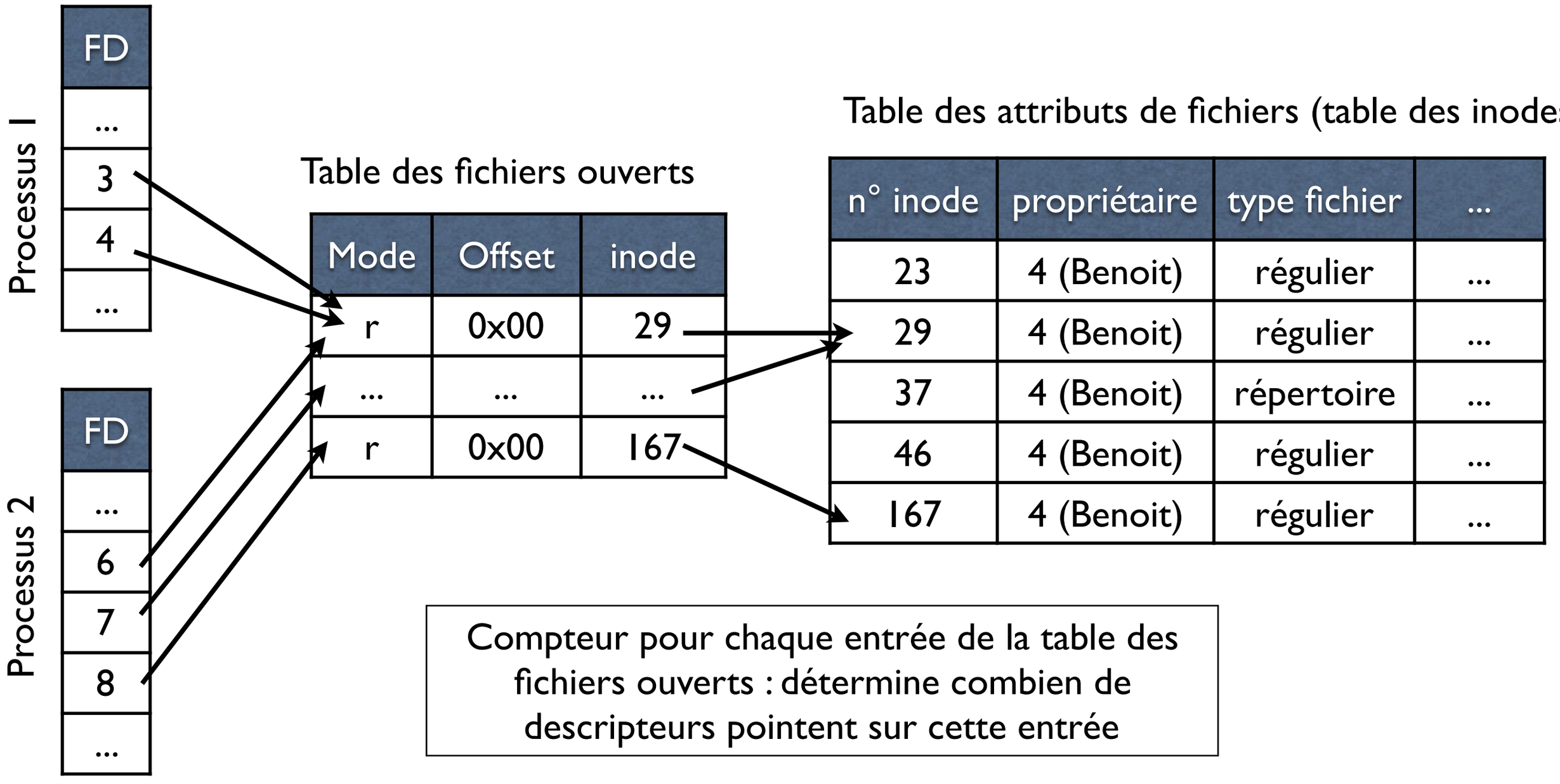


Fichiers et processus

- Interface d'accès aux fichiers
  - Un processus possède plusieurs descripteurs
  - Trois descripteurs par défaut
    - descripteur 0 : entrée standard
    - descripteur 1 : sortie standard
    - descripteur 2 : sortie erreur
- Configurations possibles
  - Plusieurs descripteurs d'un même processus peuvent pointer vers la même entrée de la table des fichiers ouverts
  - Plusieurs descripteurs de processus différents peuvent pointer vers la même entrée de la table des fichiers ouverts
  - Différentes entrées de la table des fichiers ouverts peuvent pointer vers la même entrée de la table des inodes

Fichiers et processus

- Interface d'accès aux fichiers



Ouverture d'un fichier

- Ouverture d'un fichier: création d'un descripteur
  - Appel système **open()**
    - nom de fichier à ouvrir
    - mode d'ouverture : lecture ? écriture ?
    - autres options diverses
    - retourne un descripteur de fichier
  - Crée une entrée dans la table des descripteurs
  - L'associe avec une entrée de la table des fichiers ouverts
  - Qui est elle-même associée à l'entrée inode correspondant au nom de fichier passé en paramètres

Ouverture d'un fichier

- Au sein d'un programme : appel système ***open()***
  - prototype : ***int open (char *path, int oflag, ...)***
    - path : chemin vers le fichier à ouvrir
    - oflag : options d'ouverture de fichier : macros prédéfinies
  - code retour :
    - >0 : descripteur de fichier
    - -1 : erreur (fichier inexistant, pas de droits d'accès ...)

- Options d'ouverture concaténables avec " | "
  - O_RDONLY : ouverture en lecture seulement
  - O_WRONLY : ouverture en écriture seulement
  - O_CREAT : créer le fichier s'il n'existe pas (spécifier droits)
  - ...

Ouverture d'un fichier

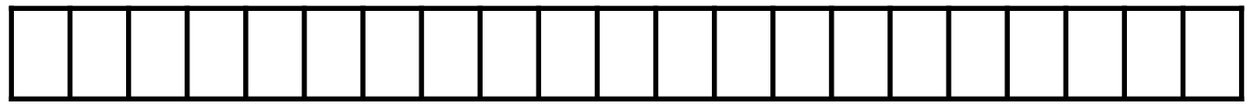
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
...

int main() {
int fd;
int fd2;

fd = open("/tmp/fichier1", O_RDONLY);
if (fd ==-1) {
printf("probleme à l'ouverture de fichier1\n");
exit(1);
}

fd2 = open("/tmp/fichier2", O_WRONLY | O_CREAT, 0666);
...

}
```

Mécanismes de lecture

- Lecture des données d'un fichier
  - Lecture à partir d'un descripteur de fichier valide
    - associé à une entrée dans la table des fichiers ouverts
    - droits de lecture autorisé pour ce fichier
  - Lecture par blocs d'octets depuis une tête de lecture
  - A chaque lecture :
    - la tête de lecture est automatiquement déplacée vers le prochain bloc d'octets à lire
    - Ecriture des données lues dans un buffer (espace tampon)
    - le buffer doit pouvoir contenir les données lues
  - En fin de fichier, un caractère spécial est lu (EOF)
    - il indique qu'il n'y a plus de données à lire
  - Appel système : **read()**

Mécanismes de lecture

- Buffer de lecture
  - espace mémoire réservé à l'écriture des octets lus
  - généralement déclaré comme un tableau de caractères de <u>taille fixe</u>.

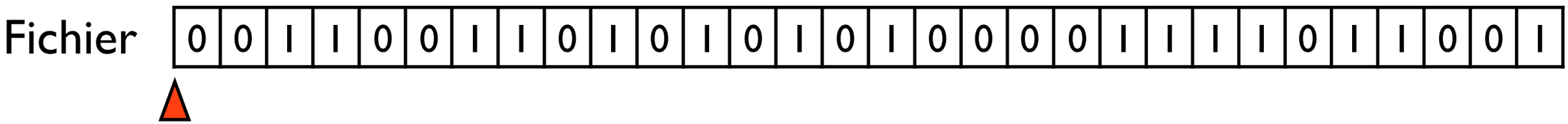
Buffer 

- Exemple

```
char monBuffer[1000];   // buffer de 1000 octets
char monBuffer2[8000];   // buffer de 8000 octets
```

Mécanismes de lecture

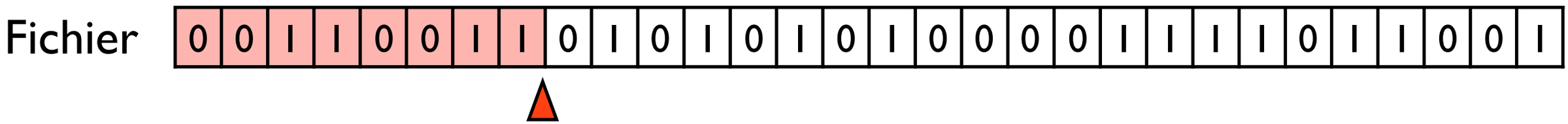
- Lecture des données d'un fichier



Lecture par blocs de 8 octets avec l'appel système read()
Déplacement automatique de la tête de lecture

Mécanismes de lecture

- Lecture des données d'un fichier

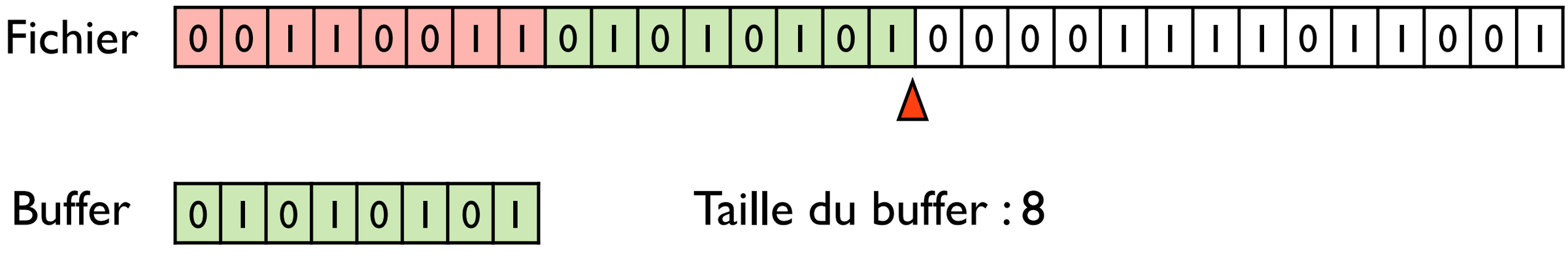


Lecture par blocs de 8 octets avec l'appel système read()
Déplacement automatique de la tête de lecture

1er appel à read()

Mécanismes de lecture

- Lecture des données d'un fichier

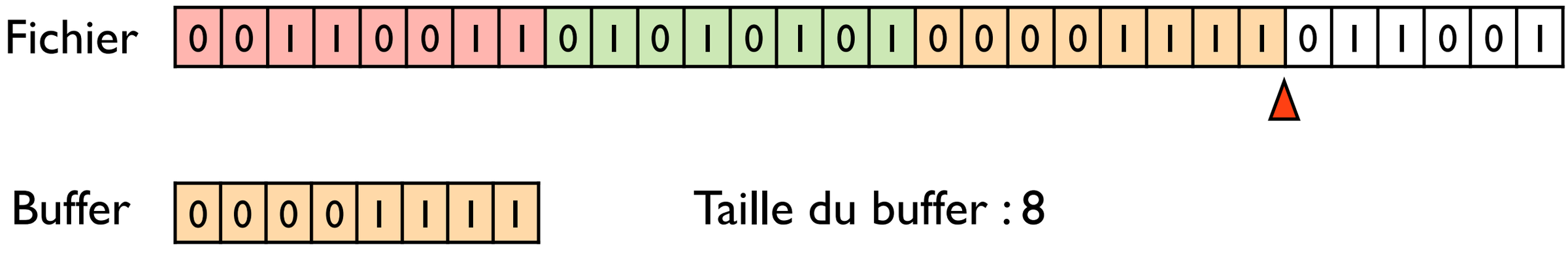


Lecture par blocs de 8 octets avec l'appel système read()
Déplacement automatique de la tête de lecture

1er appel à read()
2eme appel à read()

Mécanismes de lecture

- Lecture des données d'un fichier



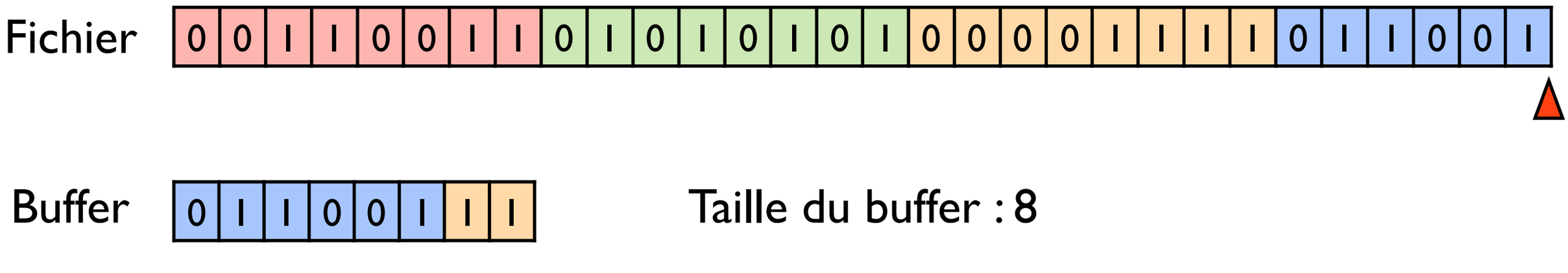
Taille du buffer : 8

Lecture par blocs de 8 octets avec l'appel système read()
Déplacement automatique de la tête de lecture

1er appel à read()
2eme appel à read()
3eme appel à read()

Mécanismes de lecture

- Lecture des données d'un fichier



Lecture par blocs de 8 octets avec l'appel système read()
Déplacement automatique de la tête de lecture

1er appel à read()
2eme appel à read()
3eme appel à read()
4eme appel à read()

Mécanismes de lecture

- Dans un programme : appel système ***read()***
  - prototype : ***size_t read (int fd, void *buffer, size_t cnt)***
  - fd : descripteur de fichier
  - buffer : buffer dans lequel écrire les octets lus
  - cnt : nombre d'octets à lire (< ou = à taille du buffer)
  - code retour :
  - >0 : nombre d'octets réellement lus
  - -1 : erreur (descripteur invalide ...)

  - note : le type void est un type générique. En pratique un buffer de type tableau de caractères (char []) est utilisé.

Mécanismes d'écriture

- Ecriture des données vers un fichier
  - Mécanismes similaires aux mécanismes de lecture
  - Ecriture vers un descripteur de fichier valide
    - associé à une entrée dans la table des fichiers ouverts
    - droits d'écriture autorisé pour ce fichier
  - Ecriture par blocs d'octets depuis une tête d'écriture
  - A chaque écriture :
    - la tête d'écriture est automatiquement déplacée après le dernier octet écrit
    - Ecriture des données depuis un buffer (espace tampon)
  - Appel système : **write()**

Mécanismes d'écriture

- Dans un programme : appel système ***write()***
  - prototype: ***size_t write (int fd, void *buffer, size_t cnt)***
  - fd : descripteur de fichier
  - buffer : buffer dans lequel lire les octets à écrire
  - cnt : nombre d'octets à écrire (< ou = à taille du buffer)
  - code retour :
  - >0 : nombre d'octets réellement écrits
  - -1 : erreur (fichier inexistant, pas de droits d'accès ...)

  - note : le type void est un type générique. En pratique un buffer de type tableau de caractères (char []) est utilisé.

Fermeture d'un fichier

- Pourquoi fermer un fichier ?
  - supprimer le descripteur de la table des descripteurs
  - mettre à jour l'entrée de la table des fichiers ouvert
    - la supprimer éventuellement
    - autrement le système peut croire qu'un programme accède encore à un fichier alors que ce n'est pas le cas

- Dans un programme : appel système ***close()***
  - prototype : ***int close (int fd)***
    - fd : descripteur de fichier à fermer
  - code retour :
    - 0 : descripteur correctement fermé
    - -1 : erreur (mauvais paramètre, erreur d'entrée sortie ...)

Exemple copie d'un fichier

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

int main() {
   int nbLus;
   int fd, fd2;
   char buffer[1024];  // déclaration du buffer de taille 1024
   fd = open("/etc/passwd", O_RDONLY);    // descripteur en lecture
   fd2 = open("/tmp/fichier", O_WRONLY|O_CREAT, 0666); // descripteur ecriture

   do {
      nbLus = read(fd, buffer, 1024);   // lecture d'un bloc de 1024 octets
      write(fd2, buffer, nbLus);        // ecriture du bloc lu vers le fichier
   } while (nbLus > 0);              // continuer tant que des octets sont lus
   close(fd);    // fermeture du descripteur fd
   close(fd2);   // fermeture du descripteur fd2
   return 0;
}
```