Fichiers

Thèmes traités

— Mise en œuvre des fichiers Unix : structures de données et fonctionnement.
— Opérations essentielles de l'API fichiers Unix, protocole d'usage.

1 Questions

1. Comment rendre le code de gestion de fichiers indépendant des caractéristiques matérielles des différents périphériques ?
2. Quel sont les avantages et les inconvénients d'une allocation contiguë pour les données d'un fichier ?

2 Ouverture d'un fichier

2.1 Les fichiers sous Unix

Les fichiers sont identifiés au sein d'un processus par un entier appelé descripteur de fichier. Celui-ci est en réalité un simple indice dans un tableau de descripteurs géré par le noyau. Par convention, les 3 premiers descripteurs de fichiers sont déjà ouverts :

— 0 : correspond à l'entrée standard (généralement le clavier) ;
— 1 : correspond à la sortie standard (généralement l'écran) ;
— 2 : correspond à la sortie standard des messages d'erreur (généralement l'écran).

On peut utiliser les constantes : `STDIN_FILENO`, `STDOUT_FILENO` et `STDERR_FILENO`.

2.2 La primitive `open`

```
int open(char *nom, int options, int mode);
```

La primitive `open` permet d'ouvrir l'accès à un fichier (et éventuellement de le créer s'il n'existe pas) dont le nom est indiqué en paramètre. Le nom référence le fichier relativement au répertoire de travail si la chaîne de caractères ne commence pas par `/`. Sinon, il s'agit d'une référence absolue à partir du répertoire racine.

La valeur de retour de la primitive est -1 si le fichier n'a pas pu être ouvert ou bien la valeur du descripteur de fichiers.

Options Les options permettent de définir comment le fichier est ouvert.

Option	Résultat
`O_RDONLY`	ouverture en lecture seule
`O_WRONLY`	ouverture en écriture seule
`O_RDWR`	ouverture en lecture et écriture
`O_APPEND`	écriture en fin de fichier si le fichier existe
`O_CREAT`	création du fichier avec les droits définis par `mode`
`O_EXCL`	avec `O_CREAT`, provoque une erreur si le fichier existe
`O_TRUNC`	remise à 0 du fichier, s'il existe

Les options peuvent être couplées en utilisant l'opérateur `|` : `O_WRONLY|O_CREAT|O_TRUNC` signifie que le fichier sera ouvert en écriture seule, et s'il existe, il sera remis à 0 ou s'il n'existe pas, il sera créé.

Modes Les modes définissent les droits d'accès au fichier lors de sa création (option `O_CREAT`). Ils sont souvent écrits en octal (base 8). Par exemple, `0644`, correspond au nombre binaire `110100100`, signifie `rw-r--r-`, c'est-à-dire droit de lecture/écriture pour l'utilisateur, lecture seule pour le groupe, lecture pour les autres. Pour éviter les erreurs potentielles, il est préférable d'utiliser les constantes :

`S_IRWXU`	l'utilisateur (propriétaire) a les droits de lecture, écriture et exécution
`S_IRUSR`	l'utilisateur a le droit lecture
`S_IWUSR`	l'utilisateur a le droit d'écriture
`S_IXUSR`	l'utilisateur a le droit d'exécution
`S_IRWXG`	le groupe a le droit de lecture, d'écriture et d'exécution
`S_IRGRP`	le groupe a le droit de lecture
`S_IWGRP`	le groupe a le droit d'écriture
`S_IXGRP`	le groupe a le droit d'exécution
`S_IRWXO`	les autres utlilisateurs ont le droit de lecture, d'écriture et d'exécution
`S_IROTH`	les autres utlilisateurs ont le droit de lecture
`S_IWOTH`	les autres utlilisateurs ont le droit d'écriture
`S_IXOTH`	les autres utlilisateurs ont le droit d'exécution

L'exemple `rw-r--r--` correspond alors à `S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH`.

2.3 Chemin d'accès à un fichier

On suppose que seule la table des i-nœuds est en mémoire et qu'un répertoire tient dans un bloc de donnée.

Question. *Combien d'accès au disque sont nécessaire pour récupérer l'i-nœud du fichier `/usr/fred/index.html` ?*

2.4 La table des fichiers ouverts

Dans son propre espace mémoire, le système garde une table contenant des informations sur tous les fichiers ouverts à un instant donné. Il s'agit bien sûr de tous les fichiers ouverts par les utilisateurs, mais aussi de ceux utilisés par le système lui-même (en particulier les devices).

Chaque élément de la table comporte un certain nombre d'informations, comme par exemple le déplacement dans le fichier (position en nombre d'octets où se fera le prochain accès, par rapport au début du fichier, position 0). Une autre information contenue dans chaque élément de ce tableau des descripteurs de fichiers est un pointeur vers la table des i-nœuds.

Si le même fichier est ouvert par plusieurs processus, il n'occupe qu'un seul élément de la table des fichiers ouverts du système. Un compteur de références représente le nombre de processus par lesquels ce fichier a été ouvert. Chaque fois qu'un processus ferme ce fichier (par la primitive `close`), le compteur est décrémenté. L'espace occupé par ce fichier n'est libéré que lorsque le dernier processus referme le fichier.

Une entrée dans la table des descripteurs de fichiers du système a un élément qui lui correspond dans la table des fichiers ouverts de chaque processus. Cet élément peut être différent d'un processus à l'autre.

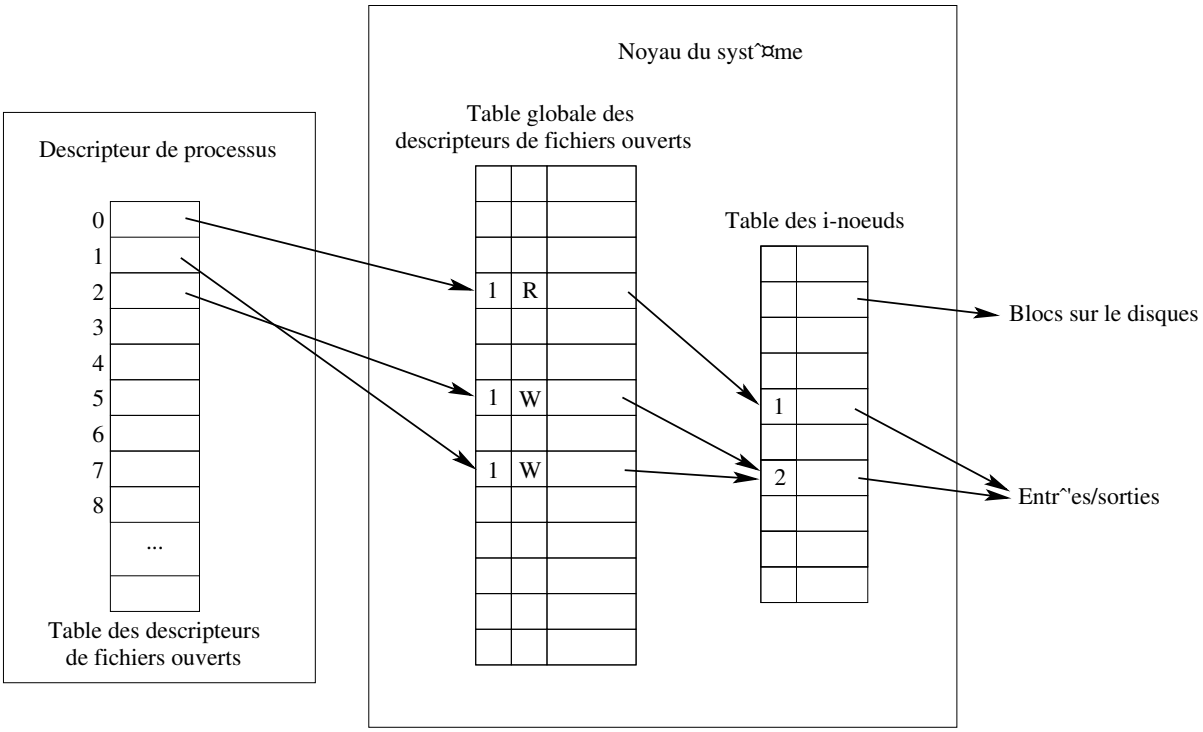

Figure 1 – Système de gestion des fichiers sous Unix

Ce n'est qu'au niveau de l'i-nœud que les différents types de fichier se distinguent. La figure 1 illustre les différentes structures du système de gestion de fichier.

Le fichier `/usr/fred/index.html` a été ouvert via la primitive `open` de la façon suivante :

```
int desc;
desc= open("/usr/fred/index.html", O_RDONLY);
```

Question. *Complétez la figure 1 en considérant que l'i-nœud du fichier `/usr/fred/index.html` a déjà été chargé en mémoire (voir section 2.3).*

Question. *Le processus exécute maintenant la primitive `fork`. Quel est le résultat pour la table des descripteurs de fichiers ouverts du système ? Quelle est la table des descripteurs de fichiers ouverts du processus créé ?*

3 Fermeture d'un fichier

```
int close(int fdesc);
```

La primitive `close` ferme l'accès au fichier par le descripteur `desc` pour le processus appelant. Le descripteur du fichier fermé peut alors être réutilisé par le processus.

4 Lecture et écriture dans un fichier

```
ssize_t read(int fdesc, char *donnees, size_t nb_octets);
ssize_t write(int fdesc, char *donnees, size_t nb_octets);
```

```
#define TAILLE_TAMPON 1024
char tampon[TAILLE_TAMPON];
int desc;
ssize_t lus;

lus= read(desc, tampon, TAILLE_TAMPON);
```

Cet appel à la primitive `read` permet la lecture de `TAILLE_TAMPON` octets du fichier accessible via le descripteur `desc`, à partir de la position courante. La suite d'octets lus est placée dans le `tampon`. La position courante progresse du nombre d'octets lus. La valeur de retour `lius` est le nombre d'octets effectivement lus ($\leq$ `TAILLE_TAMPON`) ; elle vaut 0 si la fin de fichier est atteinte dès la lecture du premier octet ; elle vaut -1 en cas de problème (fichier fermé par exemple).

```
#define TAILLE_TAMPON 1024
#define A_ECRIRE 512
char tampon[TAILLE_TAMPON];
int desc;
ssize_t ecrits;

ecrits= write(desc, tampon, A_ECRIRE);
```

Cet appel à la primitive `write` permet d'écrire, à partir de la position courante, les `A_ECRIRE` premiers octets du `tampon` dans le fichier accessible via le descripteur `desc`. La valeur de retour `ecrits` est le nombre d'octets écrits ; elle vaut -1 en cas de problème. La position courante progresse du nombre d'octets écrits.

Il est bien entendu possible de lire ou écrire des données de type différent que `char` car ils peuvent tous s'exprimer comme un tableau d'octets. Par exemple, pour lire une valeur entière :

```
int val_a_lire;
read(desc, &val_a_lire, sizeof(val_a_lire));
```

5 Exercice

1. Écrire un programme qui affiche à l'écran
   — le contenu d'une suite de fichiers passée en argument ;
   — ou ce qui est saisie sur l'entrée standard.
2. Modifier le programme de manière à écrire le résultat dans un fichier destination passé en premier argument du programme. Le programme doit retourner une erreur si le fichier destination existe.