Les fichiers (et les entrées-sorties)

Michel Simatic

module CSC4508/M2

Avril 2018

Plan du document

1 Primitives Unix d'entrées-sorties

1.1 Primitives de base

1.1.1 Ouverture/fermeture de fichier

- `int open(const char *path, int flags, mode_t mode)` : retour = f_id

  `flags` peut prendre l'une des valeurs suivantes :

  - `O_RDONLY` : lecture seulement
  - `O_WRONLY` : écriture seulement
  - `O_RDWR` : écriture et lecture

  associée à :

  - `O_APPEND` : écritures en fin de fichier
  - `O_TRUNC` : remise à vide du fichier
  - `O_CREAT` : création si le fichier n'existe pas. Les droits sont $(mode \,\&\, \tilde{}umask)$
  - `O_SYNC` : ouverture du fichier en écriture synchronisée
  - `O_NONBLOCK` (ou `O_NDELAY`) : open et les opérations ultérieures faites sur le descripteur ne pourront pas se bloquer.

- `int close(int desc)`

1.1.2 Lecture sur descripteur

- `ssize_t read(int fd, void *buf, size_t count)` : retour = nombre d'octets lus
  - Au retour de `read`, la zone `buf` est modifiée avec le résultat de la lecture
  - Dans le cas d'un fichier, le nombre d'octets lus peut ne pas être égal à `count` :
    - On se trouve à la fin du fichier
    - On a fait une lecture non bloquante et les données étaient verrouillées de manière exclusive

- `ssize_t readv(int fd, const struct iovec *iov, int iovcnt)`
  - Au retour de `readv`, les zones référencées par les `iovcnt` éléments du tableau `iov` sont modifiées avec le résultat de la lecture

1.1.3 Écriture sur descripteur

- `ssize_t write(int fd, const void *buf, size_t count)` : retour = nombre d'octets écrits
  - Dans le cas d'un fichier, le retour (sans erreur) de l'écriture signifie que:
    - Les octets ont été écrits dans les caches du noyau si pas de `O_SYNC` au moment du open
    - Les octets ont été écrits sur disque si `O_SYNC` au moment du open.
  - Dans le cas d'un fichier, si le nombre d'octets écrits est différent de `count`, cela signifie une erreur (disque plein, par exemple)
- `ssize_t writev(int fd, const struct iovec *iov, int iovcnt)`
  - Au retour de `writev`, les zones référencées par les `iovcnt` éléments du tableau `iov` ont été écrites sur le descripteur `fd`
- `int posix_fallocate(int fd, off_t offset, off_t len)`
  - Garantit que l'espace disque requis pour le fichier est effectivement alloué

1.2 Duplication de descripteur

- Mécanisme principalement utilisé pour réaliser des redirections des trois fichiers d'entrée-sorties standard.
- `int dup(int fdACloner)` : retour = nouveauFd

  associe le plus petit descripteur disponible du processus appelant à la même entrée dans la table des fichiers ouverts que le descripteur `fdACloner`
- `int dup2(int fdACloner, int fdClone)`

  force le descripteur `fdClone` à devenir un synonyme du descripteur `fdACloner`. Si le descripteur `fdClone` n'est pas disponible, le système réalise au préalable une opération de fermeture `close(fdClone)`

1.3 Contrôle des entrée-sorties

1.3.1 Primitive fcntl

- `int fcntl(int fd, int cmd, ...)` : retour fonction du type d'opération

  permet la réalisation d'un certain nombre d'opérations à différents niveaux dans les tables du système

- Les commandes possibles sont :
  - `F_GETFL` : Valeur des attributs utilisés au moment de l'ouverture du fichier
  - `F_SETFL` : Positionnement de certains de ces attributs (NB : ces attributs sont communs à tous les processus manipulant le fichier concerné)

    On peut ainsi positionner `O_ASYNC` ou `O_NONBLOCK` après création du descripteur de fichier.
  - Opérations liées au verrouillage

1.3.2 Verrouillage de fichier

- Les verrous sont attachés à un i-nœud. Donc, l'effet d'un verrou sur un fichier est visible au travers de tous les descripteurs (et donc tous les fichiers ouverts) correspondant à ce nœud
- Un verrou est la propriété d'un processus : ce processus est le seul habilité à le modifier ou l'enlever
- Les verrous ont une portée de $[entier1 : entier2]$ ou $[entier : \infty]$
- Les verrous ont un type :
  - partagé (*shared*)
  - exclusif (*exclusive*)

1.3.3 Conseil au noyau pour les lectures

- `int posix_fadvise(int fd, off_t offset, off_t len, int advice)` : retour = 0 si OK, numéro d'erreur sinon

  permet d'annoncer au noyau la manière dont on va accéder à un fichier, ce qui permet au noyau de faire des optimisations appropriées

1.4 Manipulation de l'offset

- `off_t lseek(int fd, off_t unOffset, int origine)` : retour = nouvel emplacement

  permet de manipuler l'*offset* du fichier

- `origine` permet de préciser la manière de prendre en compte le paramètre `offset` :
  - `SEEK_SET` : $offset = unOffset$
  - `SEEK_CUR` : $offset = offset + unOffset$
  - `SEEK_END` : $offset = finFichier + unOffset$

- `pread` (respectivement `pwrite`) fait une lecture (respectivement une écriture) comme `read` (respectivement `write`), mais permet en plus de préciser un *offset*.
  NB: L'*offset* du fichier n'est pas modifié

1.5 Gestion de la lenteur ou du blocage des entrées-sorties*

- La réception d'un signal par un processus arrête toute entrée-sortie en cours dans ce processus (`errno` est positionné à `EINTR`).
- Dans le cas où le descripteur est marqué avec l'option `O_NONBLOCK` : lorsque le programme invoque une primitive d'entrée-sortie dont le système détecte qu'elle se bloquerait, la primitive renvoie immédiatement une erreur et positionne `errno` à `EAGAIN` ou `EWOULDBLOCK` (selon que fichier ou *socket*).
- Le programme peut s'appuyer sur les primitives d'*entrées-sorties asynchrones* `aio_read` et `aio_write`. Chacune déclenche l'exécution d'une entrée-sortie de manière asynchrone :
  - Elle ne bloque pas l'exécution du programme ;
  - Quand son exécution est terminée (qu'elle se soit bien passée ou qu'il y ait eu une erreur), un signal est envoyé au programme qui, dans le gestionnaire de signal, peut récupérer avec `aio_return` et `aio_error` le résultat de l'exécution de l'entrée-sortie asynchrone.

  Les entrées-sorties asynchrones sont déconseillées sous Linux.

2 Bibliothèque C d'entrées-sorties

2.1 Introduction

- L'ensemble des fonctions vues jusqu'à présent sont des appels système spécifiques Unix : leur utilisation risque de compromettre la portabilité d'une application
- Le langage C définit des mécanismes standard d'entrées-sorties. Leur utilisation garantit donc la portabilité d'une application
- Néanmoins l'implémentation de ces mécanismes standards est une surcouche au dessus des primitives systèmes : ces mécanismes standard ont une pénalité en terme de performances

2.2 Ouverture/fermeture de fichier

- Ouverture :
  - `FILE *fopen (const char *path, const char *mode)`

    provoque l'ouverture d'un flux vers le fichier `path`
  - `FILE *fdopen (int fd, const char *mode)`

    permet d'associer un flux au descripteur de fichier `fd` obtenu, par exemple, à l'aide de `open`
  - `FILE *freopen (const char *path, const char *mode, FILE *stream)`

    Fonctionnalité similaire à `dup2`
  - `FILE *tmpfile (void)`

    crée un fichier temporaire
- Fermeture :
  - `int fclose (FILE *stream)`

2.3 Lecture de fichier

- `int fgetc (FILE *stream)`

  lit un caractère

- `int getc (FILE *stream)`

  équivalent de `fgetc`, mais implémenté sous forme de macro ($\rightarrow$ *a priori*, plus efficace)

- `int getchar (void)`

  lit un caractère sur `stdin`

- `char *fgets (char *s, int size, FILE *stream)`

  lit au plus $size - 1$ caractères jusqu'à `retour-chariot` ou `EOF`

- `char *gets (char *s)`

  lit une chaîne de caractères sur `stdin`

- `size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream)` : retour = nombre d'*éléments* lus (et non le nombre d'*octets* lus)

  lit un tableau d'objets

Lecture de fichier (suite)

- `int fscanf (FILE *stream, const char *format, ...)`

  fait une lecture formatée

- `int scanf (const char *format, ...)`

  fait une lecture formatée sur `stdin`

- Pour toutes les opérations de lecture, la lecture en fin de fichier déclenche le positionnement d'un indicateur testable par `feof()`

2.4 Écriture de fichier

- `int fputc (int c, FILE *stream)`

  écrit un caractère

- `int putc (int c, FILE *stream)`

  équivalent de `fputc`, mais implémenté sous forme de macro ($\rightarrow$ *a priori*, plus efficace)

- `int putchar (int c)`

  écrit un caractère sur `stdout`

- `int fputs (const char *s, FILE *stream)`

  écrit une chaîne de caractères

- `int puts (const char *s)`

  écrit une chaîne de caractères sur `stdout`

- `size_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream)` : retour = nombre d'*éléments* écrits (et non le nombre d'*octets* écrits)

  écrit un tableau d'objets

Écriture de fichier (suite)

- `int fprintf (FILE *stream, const char *format, ...)`

  fait une écriture formatée

- `int printf (const char *format, ...)`

  fait une écriture formatée sur `stdout`

2.5 Contrôle du tampon

- `int setvbuf (FILE *stream, char *buf, int buf, size_t size)`

  contrôle le tampon utilisé par la librairie standard pour cacher certaines opérations vis-à-vis du système de fichier

- `int fflush (FILE *flux)`

  vide le tampon d'un flux

  NB : ceci ne force pas l'écriture physique des données qui n'est assurée que par `O_SYNC` ou `fsync`

2.6 Divers

- `int fseek (FILE *stream, long offset, int whence)`

  contrôle l'offset

- `int fileno (FILE *stream)`

  renvoie le descripteur de fichier associé au flux

- `void clearerr (FILE *stream)`

  efface indicateur d'erreur et de fin de fichier

- `int feof (FILE *stream)`

  teste l'indicateur de fin de fichier (effaçable uniquement par `clearerr`)

- `int ferror (FILE *stream)`

  teste l'indicateur d'erreur (effaçable uniquement par `clearerr`)

3 Projection des fichiers en mémoire

- `void * mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)`

  permet d'établir une projection en mémoire d'un fichier (ou d'un périphérique) de manière à accéder au contenu du fichier avec des instructions de manipulation mémoire

- `int munmap(void *start, size_t length)`

  supprime une projection

- *ATTENTION* : `mmap` ne permet pas d'ajouter des octets à un fichier

4 Manipulation des i-noeuds du système de fichiers Unix

- Le système de gestion de fichiers Unix est basé sur la notion d'i-nœuds
- `int stat(const char *file_name, struct stat *buf)`

  `fstat(int fd, struct stat *buf)`

  renvoient des informations relatives à un fichier donné. Structure stat :

Type	nom	nature
dev_t	st_dev	Numéro du SGF
ino_t	st_io	Numéro i-nœud
mode_t	st_mode	Protection
nlink_t	st_nlink	Nombre de liens matériels
uid_t	st_uid	Numéro du propriétaire
gid_t	st_gid	Numéro du groupe
dev_t	st_rdev	Type périphérique
off_t	st_size	Taille en octets
time_t	st_atime	Date dernier accès
time_t	st_mtime	Date dernière modification
...	...	...

Manipulation des i-noeuds du système de fichiers Unix (suite)

- `int access(const char *pathname, int mode)`

  teste les droits d'accès d'un processus vis-à-vis du fichier `pathname`

- `int unlink(const char *pathname)`

  permet de détruire un nom dans le système de fichiers

- `int rename(const char *OLD, const char *NEW)`

  permet de renommer un fichier

5 Entrées-sorties sur répertoires

- Consultation :
  - `DIR *opendir (const char *path)` : retour = pointeur sur flux répertoire

    ouvre le répertoire `path` en « lecture »
  - `struct dirent *readdir (DIR *dir)` : retour = une entrée de répertoire ou NULL

    lit la première entrée (ou la suivante si un appel à readdir a déjà été fait) du répertoire `dir`
  - `void rewinddir (DIR *dir)`
  - `int closedir (DIR *dir)`
- Mise-à-jour :
  - `int mkdir(const char *path, mode_t mode)`
  - `int rmdir(const char *path)`
  - Rappel : `int unlink(const char *path)`

    permet de supprimer l'entrée `path` (et éventuellement de détruire l'i-nœud associé dans le système de fichier, si cette entrée était la seule pointant sur l'i-œud)

6 Limitations de NFS

- NFS (Network File System) est un protocole permettant un accès transparent pour les utilisateurs à des fichiers présents sur des machines distantes
- Ses limitations sont les suivantes :
  - Le serveur NFS est sans état et les verrous ne sont pas supportés pour les fichiers distants
  - Problème de cohérence : NFS gère un ensemble de caches pour les fichiers et leurs attributs. Ces caches ont une durée de validité de 3 secondes pour les fichiers et 30 secondes pour les répertoires et il existe des phases d'incohérence entre les différents clients
  - Problème de sécurité : l'accès aux fichiers distants est contrôlé au travers de l'identification (numérique) de l'utilisateur demandeur. Le bon fonctionnement d'ensemble du mécanisme suppose donc qu'une identification numérique donnée corresponde à la même personne physique sur le système de serveur de fichiers et sur les systèmes clients qui peuvent le solliciter via NFS. Solution classique : utiliser un NIS (Network Information Service)

Bibliographie du chapitre