

Architecture et Systèmes

Stefan Schwoon

Cours L3, 2018/19, ENS Cachan

Tableau des fichiers ouverts

Le noyau détient un tableau des fichiers couramment utilisés (*ouverts*) par tous les processus.

Une entrée dans ce tableau représente un accès vers un fichier et contient des informations telles que : inode référencé, mode d'accès (lire/écrire/les deux), position dans le fichier, ...

Un même fichier peut être référencé par plusieurs lignes dans ce tableau (p.ex., avec des accès différents).

Gestion des fichiers dans les processus

Chaque processus possède un ensemble de *descripteurs* (qui évolue au fil de son exécution).

Au sein du processus, un descripteur est représenté comme un entier (0,1,2,...). Au sein du système, le tuple $\langle \text{pid}, \text{fd} \rangle$ indique un fichier ouvert.

Dans un même processus, plusieurs descripteurs peuvent référencer le même fichier ouvert.

Plusieurs processus peuvent partager un même fichier ouvert.

Une entrée dans le tableau du système existe tant qu'il existe un processus qui détient une référence vers ce fichier.

Quelques fonctions pour gérer les descripteurs

Fonctions pour obtenir un descripteur (ouvrir un fichier) : `creat`, `open`, `pipe`,
...

`open` ouvre un fichier existant (pour lire ou écrire).

`dup` duplique un descripteur au sein d'un processus.

`close` supprime le descripteur dans le processus actuel.

`fork` duplique le processus avec tous ses descripteurs.

Descripteurs standard

Par défaut les trois premiers descripteurs d'un processus sont utilisés ainsi :

0 est l'**entrée standard** (stdin) (p.ex., `getch` ou `scanf` s'en servent)

1 est la **sortie standard** (p.ex., `printf` s'en sert)

2 est la **sortie erreur** (on est censé y envoyer des messages d'erreur)

Dans le terminal, l'entrée standard est typiquement alimentée par le clavier, et les sorties standard/erreur sont affichés sur l'écran (représentés par des fichiers spéciaux).

On peut changer ces descripteurs avec `dup` (utilisé par le shell lors des redirections).

Créer un descripteur

`open`: ouvrir un fichier existant. Exemples :

`open("myfile", O_RDONLY)` : ouvrir fichier en lecture (alternatives:
`O_WRONLY, O_RDWR`)

`open("myfile", O_WRONLY | O_CREAT)` : ouvrir pour écrire, créer le fichier
s'il n'existe pas

`open("myfile", O_WRONLY | O_CREAT | O_TRUNC, 0666)` : Comme
avant, mais détruire ancien contenu s'il en existe ; en plus, spécifier droits
d'accès.

`creat`: raccourci pour `open` avec `O_WRONLY, O_CREAT` et `O_TRUNC`

Lecture et écriture

`read/write(fd, p, n)` : lire/écrire n octets à partir de l'adresse p dans fichier fd

`read` et `write` renvoient les nombres d'octets qui ont été réellement lus/écrits (ce nombre peut être inférieur à n). Un renvoi de -1 veut dire erreur.

Il est conseillé de vérifier les valeurs renvoyées en cas d'erreur.

`read` un renvoi de 0 veut dire "fin de fichier".

`read` bloque si aucune donnée est disponible actuellement, mais d'autres processus pourraient en créer dans le futur.

Manipuler des descripteurs

`dup` and `dup2`: recopier un descripteur vers un autre

`g = dup(f)` : créer un nouveau descripteur `g` avec le même comportement que `f`

`dup2(f, g)` : recopier `f` vers `g`, même si `g` est déjà utilisé

`pipe`: créer un conduit unidirectionnel (tube)

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

Les données écrites dans `p[1]` apparaissent dans `p[0]`.

Les tubes

Un tube permet à deux processus d'échanger des données.

La lecture sur un tube soit renvoie toute de suite les données qui sont dedans, soit elle bloque jusqu'à ce que des données arrivent (ou que tous accès en écriture ont été fermés).

Écrire sur un tube qui n'a plus d'accès en lecture donne un signal `SIGPIPE`.

Utilisation dans le shell : `cmd1 | cmd2`

Le shell crée un tube, plus fork deux fois, ferme ses accès au tube, et attend les deux fils.

Le premier fils ferme l'accès en lecture, redirige la sortie standard vers le descripteur écriture, et fait un exec sur `cmd1`.

Le deuxième fils ferme l'accès en écriture, redirige l'entrée standard vers le descripteur lecture, puis fait exec sur `cmd2`.

Modifier position de lecture/écriture

`lseek` modifie la position dans un fichier où la prochaine opération de lecture/écriture sera effectuée.

Pas disponible sur tous les types de fichier (p.ex. pas sur les tubes).

Syntaxe: `lseek (f, p, m)`, où `m` est l'une des valeurs suivantes :

`SEEK_SET`: se positionner à l'octet numéro `p` (premier octet à 0)

`SEEK_CUR`: avancer position par `p` octets

`SEEK_END`: position relative à la fin du fichier (`p` peut être négatif)

Renvoie la position obtenue (peut être utilisé pour déterminer la taille du fichier).

I/O: Descripteurs et streams

Sous C il existe deux familles de fonctions pour les entrées/sorties :

`open`, `write`, `read`, ...

Appels système défini par POSIX

travaillent sur les *descripteurs* (0, 1, 2, ...)

`fopen`, `printf`, `scanf`, ...

Fonctions utilisateurs défini par le standard ANSI-C

travaillent sur des *stream* (`stdin`, `stdout`, `stderr`, ...)

plusieurs modes de comportement en sortie : sans tamponnage,
tamponnage par bloc, tamponnage par ligne

Ne pas mélanger les appels des deux familles !

Streams en sortie avec tamponnage

Un stream est associé avec un descripteur.

Dans un stream en mode tamponné, les opérations d'écriture ne sont pas directement transmises au descripteur mais on stocke les données dans une zone tampon.

Transmission entre zone tampon et descripteur :

- sur appel de `fflush` ;

- sur fermeture du stream ;

- en mode tamponnage par ligne : sur apparition d'une nouvelle ligne (`\n`) ;

- en mode tamponnage par bloc : quand le tampon est rempli.

Opérations

Le mode de tamponnage peut être modifié par `setvbuf`.

Il existe quelques raccourcis, voir la page man.

Important lorsqu'on :

- fait passer des données à un autre processus (par un pipe)

- `printf` sans nouvelle ligne

`fdopen` crée un stream au-dessus d'un descripteur.