340 Traitement des fichiers ouverts

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Descripteurs de fichiers

Descripteur de fichier

- Dans un processus
- Désigne un fichier ouvert
- Sert à la manipulation
- C'est un entier tout simple (`int`)

Trois descripteurs standard

- 0: entrée standard
- 1: sortie standard
- 2: sortie standard pour les messages d'erreur

$\rightarrow$ C'est des conventions de l'espace utilisateur : le noyau s'en fiche

Utilisation des descripteurs

- Ouverture: `creat(2)`, `open(2)`, etc.
- Manipulation: `read(2)`, `write(2)`, etc.

```
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
int main(void) {
  char msg[] = "Hello, World!\n";
  int fd = creat("hello", 0666);
  write(fd, msg, strlen(msg));
  close(fd);
}
```

Questions

- Pourquoi `strlen` et pas `sizeof` ?
- Quels sont les cas d'erreurs possibles pour tous ces appels système ?

Organisation interne : 3 niveaux

TD. Tables des descripteurs (une par processus)

- Une entrée par descripteur
- Pointe sur un fichier ouvert ($\rightarrow$TFO*)

TFO. Table de fichiers ouverts (globale)

- Une entrée par demande d'ouverture d'un fichier
- Chaque `open` ou `create` ou autre
- Pointe sur un inode en mémoire ($\rightarrow$TIM*)

TIM. Table des inodes en mémoire (globale)

- Une entrée par fichier (inode) distinct manipulé (ou en cache)
- Synchronisée avec les inodes sur disque

*Qui contient un compteur.

Info sur les fichiers ouverts?

- `/proc/sys/fs/inode-nr` nombre d'inodes en mémoire
- `/proc/sys/fs/file-nr` nombre d'inodes ouverts distincts

Commandes

- `lsof(1)` et `fuser(1)` permet de « voir » ou « chercher » les fichier ouverts
- Cherchent/voient aussi les communications réseau et les tubes

/proc/PID

- `/proc/PID/fd` le fichier (ou autre) associé au descripteur
- `/proc/PID/fdinfo` des informations sur le fichier ouvert

TIM: Caches des fichiers

- Table des inodes en mémoire $\rightarrow$ cache par fichier

Le SE minimise les accès disque : asynchronisme

- En lecture (*readahead*) et en écriture (*flush*)
- Demandes des utilisateurs $\neq$
  Lectures et écritures effectives sur disque
- `sync(1)`, `sync(2)`, `fsync(2)`: forcer les écritures
- `/proc/sys/vm/drop_caches`: libérer l'espace des caches

Cohérence entre accès concurrents

- Processus peuvent lire et écrire sur le même fichier
- Chacun a la même vision du contenu (le cache noyau)

Attention : ne pas confondre

- Caches noyau
- Caches applicatifs (programmes et bibliothèques)

Table des fichiers ouverts

- Une entrée par `open(2)` (ou autre) effectué
- Contient le mode d'ouverture (lecture, écriture, etc.)
- Contient le curseur lecture-écriture (éventuel) dans le fichier

→ Un même fichier peut être manipulé indépendamment par des processus

- `lseek(2)` permet de déplacer le curseur lecture-écriture

hello_wr

```
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
int main(void) {
  char msgout[] = "Hello, World!\n", msgin[50];
  int fdout = creat("hello", 0666);
  int fdin = open("hello", O_RDONLY);
  write(fdout, msgout, strlen(msgout));
  ssize_t len = read(fdin, msgin, sizeof(msgout));
  write(1, msgin, len);
  close(fdout); close(fdin); return 0;
}
```

- Un seul fichier `hello`
- Deux ouvertures (distinctes)
- Deux descripteurs

Questions

```
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
int main(void) {
  char msgout[] = "Hello, World!\n", msgin[50];
  int fdout = creat("hello", 0666);
  int fdin = open("hello", O_RDONLY);
  write(fdout, msgout, strlen(msgout));
  ssize_t len = read(fdin, msgin, sizeof(msgout));
  write(1, msgin, len);
  close(fdout); close(fdin); return 0;
}
```

- Et si on inverse le `read` et le `write`?
- Comment s'assurer de la cohérence d'un fichier si plusieurs processus peuvent écrire en même temps ?
- Pourrait-on faire communiquer des processus via un fichier commun ouvert ?

Threads, fork, exec

Pthreads partagent

- Les descripteurs sont associés au processus

$\rightarrow$ Donc partagés par les threads

Fork duplique (et partage)

- La table des descripteurs est **dupliquée**
- Les entrées dans la table des fichiers ouverts sont **partagées**
- En particulier le curseur de position (lecture/écriture)
- Les compteurs de la table des fichiers ouverts sont incrémentés
- **Question** pourquoi ne pas incrémenter les compteurs de la TIM?

Exec préserve

- La table des descripteurs et **préservée**

$\rightarrow$ Permet de préserver 0, 1 ou 2

hello_fork.c

```c
#include<unistd.h>
#include<fcntl.h>
#include<wait.h>
int main(void) {
  char buf[50];   size_t len = 0;
  int fd = open("bonjour.txt", O_RDONLY);
  pid_t p = fork();
  if (p==0) {
    len += read(fd, buf, 5);
    sleep(2);
    len += read(fd, buf+len, 5);
  } else {
    sleep(1);
    len += read(fd, buf, 5);
    wait(NULL);
  }
  write(1, buf, len); return 0;
}
```

```
$ cat bonjour.txt
Bonjonde!
ur mouldiou!
```

Spécificités Linux

Flag O_CLOEXEC

- `O_CLOEXEC` flag de `open(2)` (et autres appels système)
- Le descripteur sera automatiquement fermé lors d'un `execve(2)`

→ Évite la fuite de descripteurs ou gaspillage de ressources

→ , Mais pas portable

Tâches Linux

`clone(2)` permet de décider quoi partager ou cloner

- `CLONE_FILES` la table des descripteurs
- `CLONE_FS` des informations liées au système de fichiers, dont `chdir` et `umask`

Duplication de descripteurs

Descripteurs synonymes

- Deux descripteurs d'un même processus peuvent pointer une même entrée dans la table des fichiers ouverts
- Appels système `dup2(2)` (et `dup(2)`)

Quel est l'intérêt ?

- Redéfinir les entrées et sorties standard
- Redirection de fichiers
- Communication par tube (pour plus tard)

Question

- Quelle est la différence entre dupliquer un descripteur et ouvrir deux fois un fichier ?

Redirection de la sortie standard

```c
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>

int main(void) {
  int fd = creat("sortie", 0666);
  dup2(fd, 1);
  printf("Hello World!\n");
  return 0;
}
```

Redirection de l'entrée standard

```c
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>

int main(void) {
  int fd = open("hello", O_RDONLY);
  dup2(fd, 0);
  close(fd);
  execlp("lolcat", "lolcal", NULL);
  perror("lolcal");
  return 1;
}
```

Autre partage de descripteurs ou fichiers

- Un descripteur est un entier, partager `4` n'avance à rien

$\rightarrow$ le noyau doit être impliqué

Via sockets Unix

- `unix(7)`
- Partage des **fichiers ouverts**
- Via messages auxiliaires (`SCM_RIGHTS`)

Via /proc (Linux)

- `/proc/PID/fd` : `proc(5)`
- Partage des **inodes en mémoire**
- Même des fichiers supprimés
- Même de communication interprocessus (tubes, sockets, etc.)