Module SY5 – Systèmes d'Exploitation

Dominique Poulalhon
dominique.poulalhon@irif.fr

Université Paris Cité
L3 Informatique & DL Bio-Info, Jap-Info, Math-Info
Année universitaire 2023-2024

Organisation du système de fichiers (fin)

Modification des répertoires

création d'une entrée de répertoire :

- avec création d'un nouvel i-nœud :

```
int open(const char *pathname, int flags, mode_t mode); /* en O_CREAT *
int creat(const char *pathname, mode_t mode);
int mkdir(const char *pathname, mode_t mode);
int mkfifo(const char *pathname, mode_t mode);
int symlink(const char *target, const char *linkpath);
```

Modification des répertoires

création d'une entrée de répertoire :

- avec création d'un nouvel i-nœud :

```
int open(const char *pathname, int flags, mode_t mode); /* en O_CREAT *
int creat(const char *pathname, mode_t mode);
int mkdir(const char *pathname, mode_t mode);
int mkfifo(const char *pathname, mode_t mode);
int symlink(const char *target, const char *linkpath);
```

- sans création d'i-nœud :

```
int link(const char *oldpath, const char *newpath);
```

Modification des répertoires

création d'une entrée de répertoire :

- avec création d'un nouvel i-nœud :

```
int open(const char *pathname, int flags, mode_t mode); /* en O_CREAT *
int creat(const char *pathname, mode_t mode);
int mkdir(const char *pathname, mode_t mode);
int mkfifo(const char *pathname, mode_t mode);
int symlink(const char *target, const char *linkpath);
```

- sans création d'i-nœud :

```
int link(const char *oldpath, const char *newpath);
```

suppression d'une entrée de répertoire

```
int unlink(const char *pathname);
int rmdir(const char *pathname);
```

modification d'une entrée de répertoire

```
int rename(const char *oldpath, const char *newpath);
```

Modification des répertoires

```
int link(const char *oldpath, const char *newpath);
```

- `oldpath` est une référence valide de fichier autre qu'un répertoire (sauf si utilisateur privilégié)
- `newpath` ne correspond à aucun lien existant,
- `dirname(newpath)` désigne un répertoire sur le même disque que `oldpath`

- crée un nouveau lien physique `basename(newpath)` dans le répertoire `dirname(newpath)` vers l'i-nœud désigné par `oldpath`,
- incrémente le compteur de liens de l'i-nœud,
- retourne 0, ou -1 en cas d'échec.

Modification des répertoires

```
int unlink(const char *pathname);
```

où `pathname` est une référence valide de fichier *autre que répertoire*,

- supprime le lien correspondant dans `dirname(pathname)`,
- décrémente le compteur de liens de l'i-nœud correspondant,
- si ce compteur est nul (et si le nombre d'ouvertures du fichier est nul), le fichier est supprimé,
- retourne 0, ou -1 en cas d'échec.

Modification des répertoires

```
int unlink(const char *pathname);
```

où `pathname` est une référence valide de fichier *autre que répertoire*,

- supprime le lien correspondant dans `dirname(pathname)`,
- décrémente le compteur de liens de l'i-nœud correspondant,
- si ce compteur est nul (et si le nombre d'ouvertures du fichier est nul), le fichier est supprimé,
- retourne 0, ou -1 en cas d'échec.

Pour la suppression des répertoires *vides* :

```
int rmdir(const char *pathname);
```

Modification des répertoires

```
int rename(const char *oldpath, const char *newpath);
```

- `oldpath` est une référence valide de fichier autre que `.` et `..`
- si `newpath` correspond à un lien existant, il doit être de même type que `oldpath`

- remplace, *de manière atomique*, le lien (déduit de) `oldpath` par le lien (déduit de) `newpath`,
- si ce lien existait déjà, il est supprimé (cf `unlink` et `rmdir`)
- retourne 0, ou -1 en cas d'échec.

Modification des répertoires

analogues avec précision d'un (descripteur de) répertoire de référence :

```
int openat(int dirfd, const char *pathname, int flags, mode_t mode);
int mkdirat(int dirfd, const char *pathname, mode_t mode);
int mkfifoat(int dirfd, const char *pathname, mode_t mode);
int symlinkat(const char *target, int newdirfd, const char *linkpath);

int linkat(int olddirfd, const char *oldpath,
                int newdirfd, const char *newpath, int flags);

int unlinkat(int dirfd, const char *pathname, int flags);

int renameat(int olddirfd, const char *oldpath,
                 int newdirfd, const char *newpath);
```

Processus

Définition

processus = objet dynamique correspondant à une exécution d'un programme

Définition

processus = objet dynamique correspondant à une exécution d'un programme

au cours du temps, un processus passe en boucle par les états suivants :

- état prêt
- états actifs (actif noyau ou actif utilisateur)
- état endormi ou en attente

Définition

processus = objet dynamique correspondant à une exécution d'un programme

au cours du temps, un processus passe en boucle par les états suivants :

- état prêt
- états actifs (actif noyau ou actif utilisateur)
- état endormi ou en attente

autres états possibles :

- état transitoire initial
- état suspendu
- état zombie

un processus a deux constituants :

- son espace d'adressage : la zone mémoire où il travaille, divisée en segment de texte (le code à exécuter) et segment de données – statiques et dynamiques (pile et tas)

un processus a deux constituants :

- son espace d'adressage : la zone mémoire où il travaille, divisée en segment de texte (le code à exécuter) et segment de données – statiques et dynamiques (pile et tas)

- son bloc de contrôle : toutes les informations dont le système a besoin pour assurer la bonne gestion des processus : entre autres, identifiant, état, compteur ordinal, pointeur de pile, répertoire de travail, descripteurs...
  les informations utiles même lorsque le processus est inactif sont stockées dans la table des processus ; les autres (descripteurs par exemple) peuvent être stockées dans l'espace d'adressage.

Quelques attributs des processus

son identifiant, celui de son père

```
pid_t getpid(void);
pid_t getppid(void);
```

ses propriétaires (réel et effectif)

```
uid_t getuid(void);
uid_t geteuid(void);
int setuid(uid_t uid);
int seteuid(uid_t euid);
```

son répertoire de travail courant

```
char *getcwd(char *buf, size_t size);
int chdir(const char *path);
int fchdir(int fd);
```

Création de processus

sous Unix, la création de processus est scindée en deux étapes :

Création de processus

sous Unix, la création de processus est scindée en deux étapes :

- le clonage = création d'un processus (presque) identique : même état de la mémoire (code, pile, tas), même compteur ordinal, même pointeur de pile, mêmes fichiers ouverts... $\Longrightarrow$ `fork()`

  (le nouveau processus dispose de son propre espace d'adressage, indépendant de celui de son père, et naturellement de son propre bloc de contrôle)

Création de processus

sous Unix, la création de processus est scindée en deux étapes :

- le clonage = création d'un processus (presque) identique : même état de la mémoire (code, pile, tas), même compteur ordinal, même pointeur de pile, mêmes fichiers ouverts... $\Longrightarrow$ `fork()`

  (le nouveau processus dispose de son propre espace d'adressage, indépendant de celui de son père, et naturellement de son propre bloc de contrôle)

- le recouvrement = remplacement de toute la mémoire par un nouveau segment de code, réinitialisation de la pile et du tas, réinitialisation des registres $\Longrightarrow$ (famille) `exec*()`

Création de processus

sous Unix, la création de processus est scindée en deux étapes :

- le clonage = création d'un processus (presque) identique : même état de la mémoire (code, pile, tas), même compteur ordinal, même pointeur de pile, mêmes fichiers ouverts... $\Longrightarrow$ `fork()`

  (le nouveau processus dispose de son propre espace d'adressage, indépendant de celui de son père, et naturellement de son propre bloc de contrôle)

- le recouvrement = remplacement de toute la mémoire par un nouveau segment de code, réinitialisation de la pile et du tas, réinitialisation des registres $\Longrightarrow$ (famille) `exec*()`

hiérarchie de processus : un processus et ses descendants forment un groupe de processus, auquel on peut envoyer collectivement un signal ; par ailleurs le père est d'une certaine manière « responsable » de ses fils

Création de processus

```
pid_t fork(void);
```

- retourne -1 en cas d'erreur (et `errno` est positionnée)

sinon :

- crée un nouveau processus (*fils*) par clonage du processus courant (*père*)
- retourne 0 dans le processus fils
- retourne le pid du fils dans le processus père

Création de processus

```
pid_t fork(void);
```

- retourne -1 en cas d'erreur (et `errno` est positionnée)

sinon :

- crée un nouveau processus (*fils*) par clonage du processus courant (*père*)
- retourne 0 dans le processus fils
- retourne le pid du fils dans le processus père

autrement dit, ***un appel*** à cette fonction entraîne ***deux retours***

le nouveau processus ne diffère de l'ancien essentiellement que par son identifiant (et celui de son père)

le fils poursuit l'exécution au point où en était son père

Création de processus

```
pid_t fork(void);
```

Comment différencier le père du fils ? par la valeur de retour de `fork`

```
switch(r = fork()) {
  case -1:
    perror("fork");
    exit(1);
  case 0: /* code pour le fils */
    break;
  default: /* code pour le père */
}
```

Création de processus

```
pid_t fork(void);
```

l'*espace d'adressage* du fils est (initialement) une copie de celui du père

Création de processus

```
pid_t fork(void);
```

l'*espace d'adressage* du fils est (initialement) une copie de celui du père

la *table des descripteurs* du fils est (initialement) une copie de celle du père

Création de processus

```
pid_t fork(void);
```

l'*espace d'adressage* du fils est (initialement) une copie de celui du père

la *table des descripteurs* du fils est (initialement) une copie de celle du père

chaque descripteur du fils pointe sur la *même entrée* de la table des ouvertures de fichiers que le descripteur correspondant du père

Création de processus

```
pid_t fork(void);
```

l'*espace d'adressage* du fils est (initialement) une copie de celui du père

la *table des descripteurs* du fils est (initialement) une copie de celle du père

chaque descripteur du fils pointe sur la *même entrée* de la table des ouvertures de fichiers que le descripteur correspondant du père

ils partagent donc la *même position courante* dans le fichier ouvert