E/S UNIX

Plan

- Communication par flots de données
- Structuration de l'espace des fichiers
  - ◊ Disques logiques et espace physique
  - ◊ Organisation d'un disque logique
  - ◊ Opérations sur les i-nœuds
  - ◊ Organisation de l'espace de noms utilisateur
  - ◊ Opérations sur les répertoires
- Fonctionnement du sous-système d'E/S
- Interface d'E/S
  - ◊ Accès aux fichiers
  - ◊ Duplication et redirection
  - ◊ Tubes
  - ◊ Contrôle des flots de données : fcnl, select

1 – Communication par flots de données

Idée : Interface *unique* pour les échanges avec *toutes* les ressources du système

→

- « objet » unique : fichier (séquentiel), ou *flot*
- opérations « génériques »
  - ◊ ouvrir/fermer
  - ◊ lire/écrire
  - ◊ se positionner

  interprétées en fonction de la nature du périphérique/fichier source ou cible de l'échange

Catégories de fichiers

- fichiers ordinaires (*regular files*) : conteneurs de données
- répertoires (*directories*) : regroupent des noms de fichiers, au gré des utilisateurs
- lien symboliques (*soft links*) : contiennent des noms (chemins) alternatifs pour les fichiers
- tubes (*pipes*) : canaux de communication FIFO entre processus
- fichiers spéciaux (*special files*) : permettent de désigner les périphériques comme des fichiers
  - ◊ traditionnellement situés dans `/dev`
  - ◊ deux catégories : bloc et caractère
  - ◊ identification : numéro majeur (pilote), numéro mineur (adresse)
- *sockets* : prises de communication, pour l'établissement de canaux virtuels entre processus

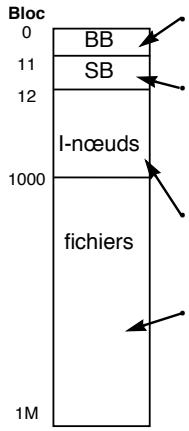
2 – Structuration de l'espace des fichiers

3 niveaux de structuration

- organisation *physique*, gérée par le périphérique, indépendamment du système
  Exemple : partitions disque (Le contrôleur gère une table des partitions)
- organisation *logique* : allocation de la ressource (espace disque), gérée par le système.
  Exemple : pour l'espace disque, UNIX distingue deux types de disques logiques (partitions)
  - les espaces de *swap*, utilisés pour la gestion de la mémoire virtuelle
  - les *systèmes de fichiers* (SGF), utilisés pour le stockage « classique « des fichiers
- organisation externe, définie par l'utilisateur : service de *répertoires*

1)Implémentation d'un système de fichiers

Organisation de base (System V, s5fs)



blocs d'amorce
UNIX : `bootblocks (BB)`

informations sur la structure du disque logique : adresse, type des ≠ zones qui le constituent, adresse du premier bloc libre
UNIX : `superbloc (SB)`

catalogue du disque : liste des *descripteurs* des fichiers présents sur le disque logique
UNIX : `table des I-nœuds`

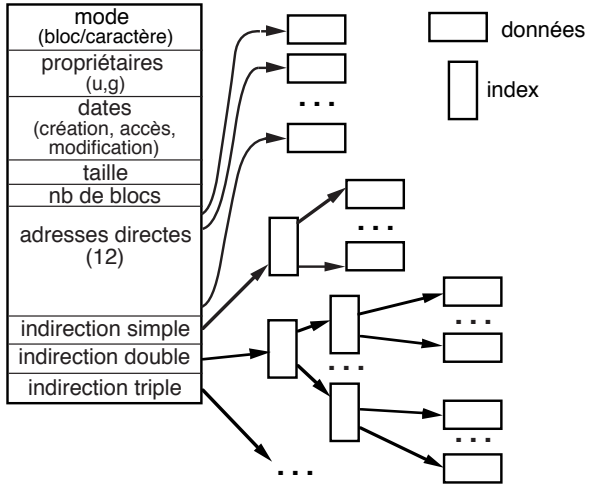
fichiers (ordinaires, répertoires, liens)

Organisation BSD (ufs)

Ajout d'une notion de cylindre (≈ partitionnement du disque logique en cylindres), pour réduire les temps d'accès disque

2) Opérations sur les i-nœuds

Structure d'un i-nœud



Avantage : les petits fichiers (les plus nombreux) sont représentés de manière efficace

Opérations (/usr/include/sys/stat.h)

- lecture :
  - ◊ `int stat(const char *, struct stat *)`;
  - ◊ `int fstat(int, struct stat *);`
- création :
  - ◊ `int mkdir(const char *, mode_t);` (répertoires),
  - ◊ `int create (const char *, mode_t)` (fichiers réguliers),
  - ◊ `int mknod(const char *,mode_t, dev_t);`(fichiers spéciaux),
  - ◊ `int mkfifo(const char *,mode_t);`(tubes nommés)
  - ◊ …
- manipulation des attributs :
  - ◊ droits
    - `int chmod(const char *, mode_t);`
    - `int fchmod(int, mode_t);`
    - `mode_t umask(mode_t);`
  - ◊ propriétaire
    - `int chown(const char *, uid_t, gid_t);`
    - `fchown(int, uid_t, gid_t)`

Représentation d'un i_nœud (/usr/include/sys/stat.h)

```
struct stat {
  dev_t     st_dev;   /* [XSI] ID of device containing file */
  ino_t     st_ino;   /* [XSI] File serial number */
  mode_t    st_mode;  /* [XSI] Mode of file (see below) */
  nlink_t   st_nlink; /* [XSI] Number of hard links */
  uid_t     st_uid;   /* [XSI] User ID of the file */
  gid_t     st_gid;   /* [XSI] Group ID of the file */
  off_t     st_size;  /* [XSI] file size, in bytes */
  blkcnt_t  st_blocks;      /* [XSI] blocks allocated for file */
  blksize_t st_blksize;     /* [XSI] optimal blocksize for I/O */
  time_t    st_atime;       /* [XSI] Time of last access */
  time_t    st_mtime;       /* [XSI] Last data modification time */
  time_t    st_ctime;       /* [XSI] Time of last status change */
  …
};
```


3) Organisation de l'espace de noms utilisateur

Principe

- Les SGF permettent à l'utilisateur de définir une structure aborescente pour l'espace de noms (répertoires)
- Un SGF est privilégié : SGF/disque système
  - ◊ les autres SGF (fixes ou amovibles) sont greffés (*montés*) sur l'arborescence du SGF système
  - ◊ mécanisme utilisé par NFS pour l'interconnexion de SGF distants

Mise en œuvre du service de répertoires

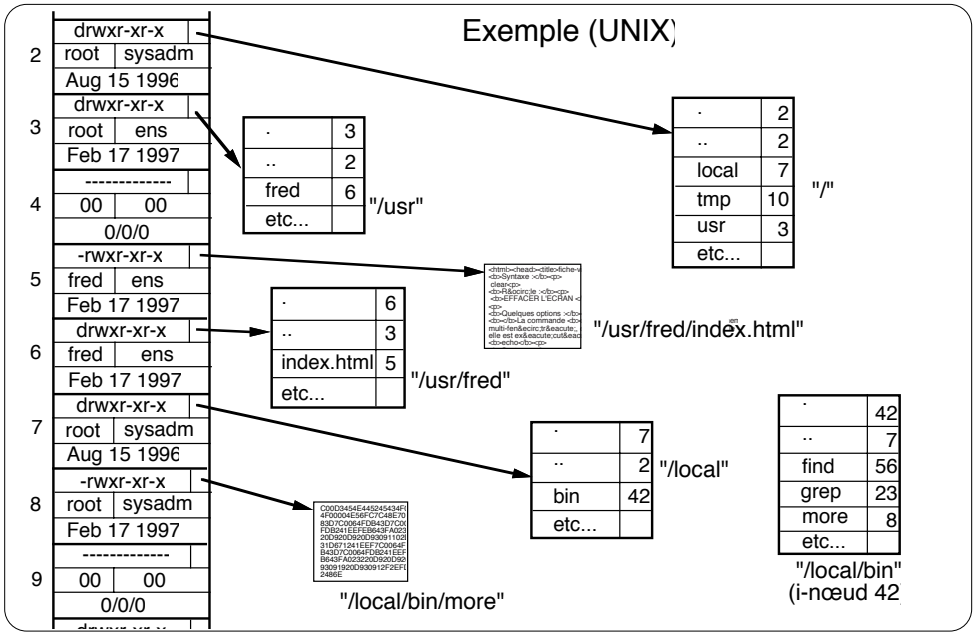
- les i-nœuds ne contiennent pas le nom des fichiers, ce qui simplifie les mises à jour
- les répertoires sont implantés par des *fichiers*, contenant une *table de correspondance :*

  nom externe des fichiers regroupés dans le répertoire ↔ nom interne (numéro de i-nœud)

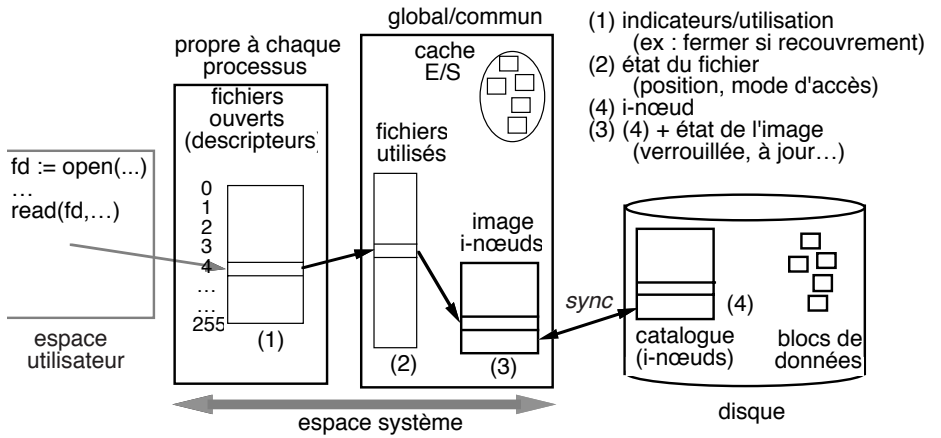
fichier contenant le répertoire "/local/bin" (i-nœud 42)

	42
..	7
find	56
grep	23
more	8
etc...	

- un chemin d'accès est interprété pas à pas à partir du répertoire racine du système de fichiers
- le répertoire racine d'un SGF est toujours associé au i-nœud numéro 2



3 – Fonctionnement du sous-système d'E/S



- Liaison (nom externe → id interne) à l'*ouverture* des fichiers
- « file descriptor » ~ indice dans la table des fichiers ouverts

4) Opérations sur les répertoires

Création/destruction de répertoires

- **`int mkdir(char *nomr, int mode);`**
- **`int rmdir(char *nomr);`**

Création/destruction d'entrées de répertoires

- Un compteur de références est géré, dans le descripteur de chaque fichier ordinaire.
- Ce compteur est incrémenté/décrémenté à chaque création/destruction de lien physique vers ce fichier
- Un fichier ordinaire est effectivement considéré comme détruit lorsque (et seulement lorsque) son compteur de références devient nul
- Opérations
  - ◊ **`int link(const char *fichier_existant, const char *nouveau_nom);`**
  - ◊ **`int unlink(const char *fichier_existant);`**

4 – Interface d'E/S

1) Accès aux fichiers

Ouverture

Avant d'utiliser un fichier, il faut l'ouvrir, ce qui lui alloue un descripteur.

`int open (const char *chemin, int mode, mode_t droits);`

descripteur — nom du fichier — (combiné avec le masque de création)

`O_RDONLY`	ouverture en lecture
`O_WRONLY`	ouverture en écriture
`O_RDWR`	ouverture en lecture et écriture
`O_APPEND`	ouverture en écriture en fin de fichier
`O_CREAT`	création du fichier avec droits d'accès définis par `droits`
`O_EXCL`	avec `O_CREAT` provoque un échec si le fichier existe déjà.
`O_TRUNC`	ramène la taille du fichier à zéro si le fichier existe déjà.

…

Note : les modes peuvent être combinés (quand cela a un sens) avec le ou (`|`)

Exemple

```
fd = open ("/home/toto/fich", O_RDONLY|O_CREAT, 0)
```

Ouvre le fichier désigné par `/home/toto/fich` en lecture seule;

Remarque

Il existe une opération creat

Positionnement

- Les opérations d'accès au fichier sont effectuées à partir d'une position courante (*offset*).
- Initialement (à l'ouverture) la position courante est 0.
- Cette position est modifiée
  - ◊ indirectement, par les opérations d'accès : lecture (`read`) et écriture (`write`).
  - ◊ directement, par l'opération `lseek`

```
long lseek (int descripteur, long offset, int origine);
```

nouvelle position

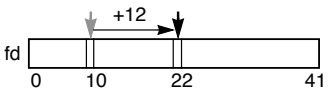
déplacement demandé

SEEK_SET	à partir du début du fichier
SEEK_CUR	à partir de la position actuelle
SEEK_END	à partir de la fin du fichier

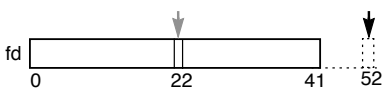
- La position courante peut être fixée après la fin actuelle du fichier.

Exemples

lseek(fd, 12, SEEK_CUR)
la position courante progresse de12 octets depuis sa valeur actuelle



lseek(fd, 52, SEEK_SET)
la position courante est fixée à 52



Lecture

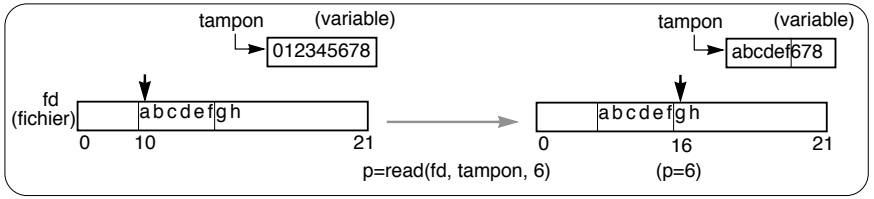
```
ssize_t read (int descripteur, void *tampon, size_t taille);
```

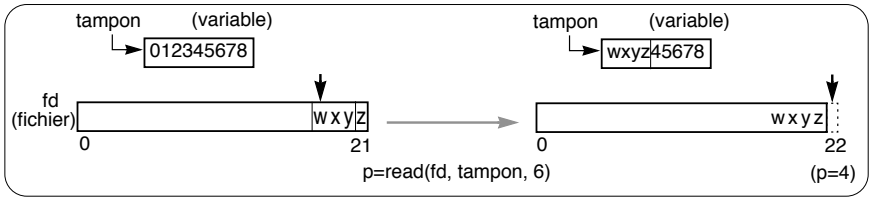
nombre d'octets effectivement lus (-1 si erreur)

adresse du résultat de la lecture

nombre d'octets à lire

Exemple





Ecriture

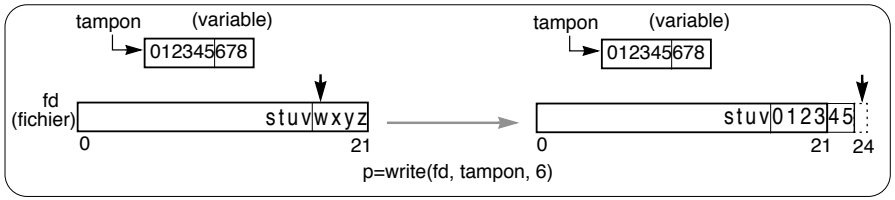
```
ssize_t write (int descripteur, void *tampon, size_t taille);
```

nombre d'octets effectivement écrits (-1 si erreur)

adresse de la zone contenant les données à écrire

nombre d'octets à écrire

Exemple



Remarque

Il existe des primitives permettant de lire ou d'écrire des vecteurs d'octets (**`readv/writev`**), ou à partir d'une position donnée (**`pread/pwrite`**)

Fermeture

Un fichier qui n'est plus utilisé peut être fermé, en fournissant son descripteur en paramètre

```
int close (int descripteur);
```


Remarque

Les primitives fournies par le noyau (open, close, lseek, read, write)

- sont de bas niveau (opérations sur des suites de caractères)
- représentent un coût d'exécution non négligeable (commutation en mode superviseur)

→ des bibliothèques sont fournies pour faciliter et optimiser leur usage

Exemple

La bibliothèque C « standard », implantée (en général) dans `/lib/libc.a`, propose des fonctions d'E/S définies dans `/usr/include/stdio.h`,

- qui permettent de définir des E/S de plus haut niveau (E/S « formatées ») : fopen, fread, fwrite, fscanf, fprintf, fflush, fseek, fclose (et fonctions analogues pour les chaînes : sprintf, sscanf
- qui sont réalisées à partir des primitives systèmes
- réduisent le nombre d'appels systèmes nécessaires en conservant les données à lire/écrire dans des caches gérés au niveau langage

  → **Attention !!**

  Lorsqu'on utilise la bibliothèque d'E/S standard, on agit, *de manière opaque*, sur l'offset

  → combiner l'utilisation des 2 familles (système/langage) d'opérations d'E/S dans un même programme est source d'erreurs

2) Duplication et redirection

Un descripteur de fichier ouvert peut être recopié/affecté à un autre descripteur

- la commande `dup` affecte le descripteur fourni en paramètre au plus petit descripteur non utilisé

```
int dup (int descripteur)
```

descripteur affecté — descripteur à copier

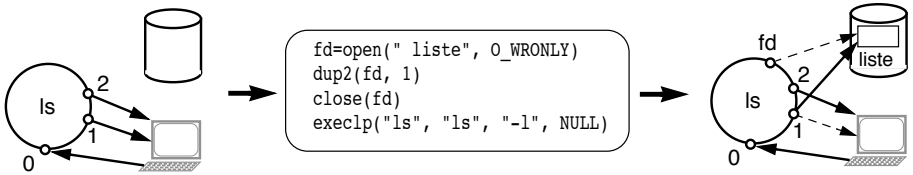
- la commande `dup2` réalise l'affectation explicite entre descripteurs

```
int dup2 (int descripteur_copié, int descripteur_affecté)
```

Si le descripteur affecté désignait déjà un fichier, celui-ci est fermé

Utilisation fréquente : réalisation des redirections

Problème : implanter la commande "`ls -l > liste`"

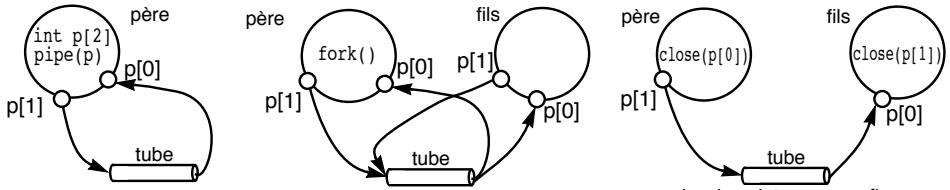


3) Tubes (commande `pipe`)

Les tubes permettent d'échanger des flots de données entre processus ayant un ancêtre commun

- un tube est un tableau (T) de 2 descripteurs (fichiers)
  - ◊ le premier élément du tableau (T[0]) est la *sortie* du tube
  - ◊ le second élément (T[1]) est l'*entrée* du tube
- le flot de données transite de l'entrée du tube à la sortie du tube, dans l'ordre FIFO : les données sont transmises en écrivant dans l'entrée (T[1]) et en lisant dans la sortie (T[0]) du tube
- un tube est un tampon de taille bornée :
  - ◊ une lecture dans un tube vide est bloquante
  - ◊ une écriture dans un tube plein est bloquante
  - ◊ une lecture dans un tube vide, sans écrivain potentiel renvoie 0
  - ◊ une écriture dans un tube sans lecteur potentiel provoque l'envoi de `SIGPIPE`

Schéma élémentaire de construction de processus communiquant par tubes



le père crée un tube — *le père crée un fils* — *les descripteurs superflus sont fermés*

4) Contrôle des flots de données : fcntl, select

La commande `fcntl` permet de consulter ou d'affecter des attributs relatifs au descripteur d'un fichier (`F_GETFD/F_SETFD`), ou à son mode d'ouverture (`F_GETFL/F_SETFL`)

```
int fcntl (int descripteur, int commande, int valeur);
```

`F_GETFD`	lecture des attributs descripteur
`F_SETFD`	affectation de la valeur fournie en 3ème paramètre aux attributs descripteur Attribut (pré)défini : **`FD_CLOEXEC`**,fermeture sur recouvrement (`exec`)
`F_GETFL`	lecture des attributs propres au mode d'ouverture
`F_SETFL`	affectation de la valeur fournie en 3ème paramètre aux attributs du mode d'ouverture Attributs (pré)définis : **`O_APPEND`**,écriture en fin de fichier **`O_SYNC`**,écriture directe, sans cache **`O_NONBLOCK`**,E/S non bloquantes (retour = -1 et errno = EAGAIN si situation de blocage)

Note : les attributs peuvent être combinés au moyen du ou (|)

Exemple

```
fcntl(desc, F_SETFL, fcntl(desc, F_GETFL) | O_NONBLOCK));
```

permet de rendre non bloquantes les E/S sur le descripteur `desc`

La commande `select` permet de gérer la scrutation sur un ensemble de descripteurs.

```
int select (int nbdesc,
            fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
            struct timeval *timeout)
```

0 : délai écoulé
> 0 : nombre de descripteurs prêts

majorant du plus grand numéro de descripteur scruté

Délai de garde pour la scrutation (NULL : attente infinie)

Entrée : ensembles des descripteurs scrutés pour la disponibilité de caractères en lecture, en écriture, ou « hors bande » (sockets TCP)
Retour : ensembles des descripteurs trouvés prêts

Notes

- Les ensembles sont des tableaux de bits, de même dimension que les tables de descripteurs
- Des macros (définies dans <types.h> permettent de manipuler ces ensembles (`fd_set`) :
  - ◊ **`FD_ZERO`** `(&fds)` initialise l'ensemble de descripteurs `fds` à 0 (ensemble vide)
  - ◊ **`FD_SET`**`(fd,&fds)` ajoute le descripteur `fd` à l'ensemble `fds`
  - ◊ **`FD_CLR`** `(fd,&fds)` supprime le descripteur `fd` de `fds`
  - ◊ **`FD_ISSET`**`(fd,&fds)` teste si `fd` appartient à `fds` (non nul si `fd` appartient à `fds`)