Cours de Système : Gestion de Fichiers

Bertrand Le cun et Emmanuel Hyon
bertrand.le_cun@u-paris10.fr et Emmanuel.Hyon@u-paris10.fr

Université Paris X

24 janvier 2011

Qu'est ce qu'un S.G.F. ?

Définition

- Système de gestion de fichiers
- Partie du S.E. qui maintient les données sur les périphériques mémoires (disques, disquettes, clefs USB).
- Définit la structure d'un disque : fichiers, dossiers, etc...
- Fournit une interface (conviviale) à l'utilisateur.

Objectifs : Organisation Logique du disque

Comment associer les fichiers à un espace sur le disque ?

- Optimiser l'utilisation des disques,
- Placement des données,
- Minimiser les temps d'accès.

Différents SGF

Windows

- MS-DOS ou FAT 16 (file allocation table) ($\leq$ W 95)
- FAT 32 ou VFAT (W 98)
- NTFS
- WinFS
- Joliet (pour les CD-ROM)

Unix (like)

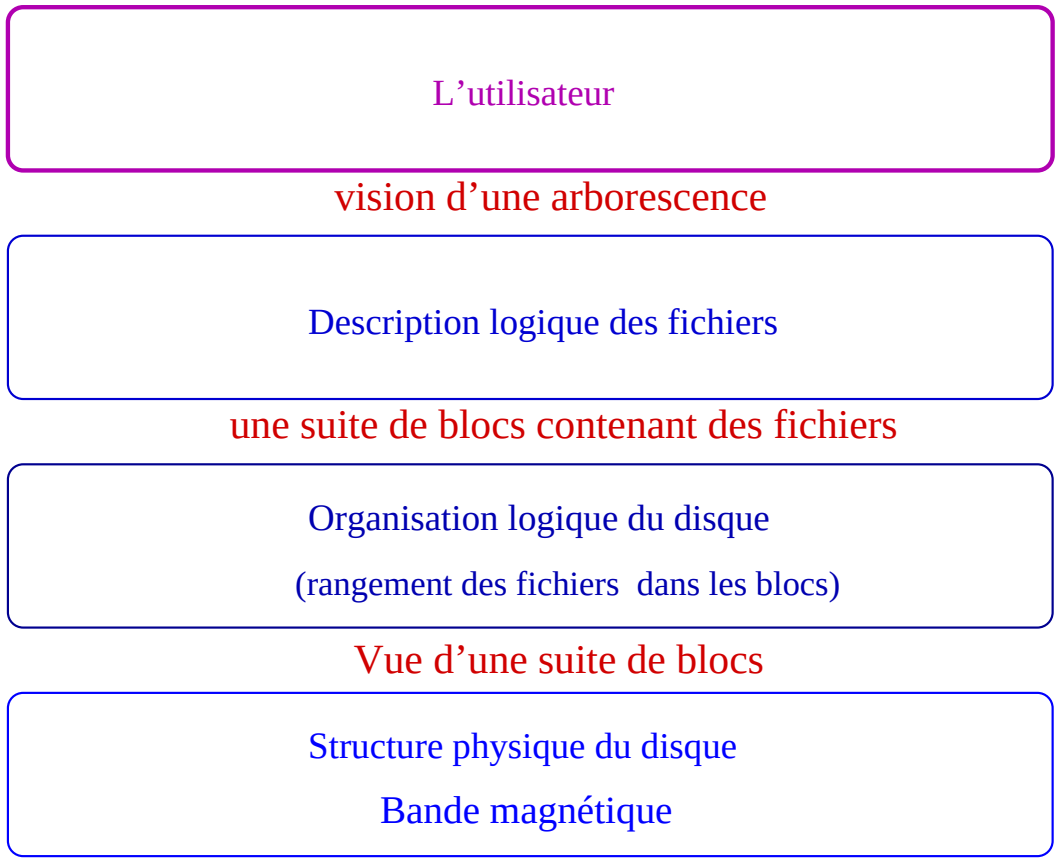
- ufs (unix BSD)
- ext2fs
- ext3fs
- reiserfs
- iso9660 (pour les CD-ROM)

Disque et S.E.

Relation

- Un disque est une unité physique Erreurs, Bad Block, etc..
- Différents niveaux d'accès
  - Disque Physique (surface, cylindre, secteur)
  - Disque Logique (numéro de block)
  - Fichier Logique (Donnée numéro # dans le fichier)
- Le S.E. cache les aspects physiques du disque pour un accès haut niveau (abstraction).
  - Bas niveau : block
  - haut niveau : fichier, dossier,

De la bande magnétique à l'arborescence



Structure physique d'un disque

Composants d'un disque

- Plateaux,
- Surfaces de plateaux
- Pistes concentriques sur les surfaces
- Secteurs composants les pistes
- Cylindres : ensemble de pistes de même diamètre sur différentes surfaces.
- Têtes de lectures au bout de bras peuvent lire en même temps plusieurs secteurs.

Structure logique d'un disque

Interface d'un disque

- Atteindre une donnée demande beaucoup d'infos
  # Cylindre, # surface, ,# secteur, taille de transfert
- Les anciens S.E. se devaient de donner toutes ces infos
  Le S.E. devait connaître toutes les "spec" du disque
- Les disques modernes sont encore plus compliqués
  secteurs de taille différentes, secteurs remappés
- Paramètres du disque cachés au S.E.
- Interfaces simplifiées
  Bloc logique $[0 \cdots n]$.

Performance d'un disque

Mesure de performance

- Seek : déplacement de la tête pour obtenir le cylindre correct
- Rotation : attente de rotation du disque pour que le secteur soit sous la tête.
- Transfert : temps de transfert des données du disque vers l'électronique, puis vers la mémoire.

Performance d'un disque

Mesure de performance

- Seek : déplacement de la tête pour obtenir le cylindre correct
- Rotation : attente de rotation du disque pour que le secteur soit sous la tête.
- Transfert : temps de transfert des données du disque vers l'électronique, puis vers la mémoire.

Implication au niveau du S.E.

- Gestion d'une mémoire tampon interne au SE, pour garder en mémoire les blocs les plus accédés
- Deux blocs accédés souvent ensemble doivent être proches Les blocs d'un même fichier doivent être proches.

Organisation logique : comment et où stocker un fichier

Fichier sur Disque

- Unité de stockage sur le disque : les blocs

Stockage de fichier

- Petits fichiers (< à la taille d'un bloc)

  Le fichier est dans un bloc.

  Le bloc ne contient que ce fichier.
- Gros fichiers :
  - Découpage du fichier en petits bouts de taille<= à un bloc disque.
  - Sauvegarde de ces petits sur les blocs

Un problème

Quels blocs choisir ?

Allocation des blocs

Au cours du temps...

- Beaucoup de modifications sur les fichiers. D'où
  - Un fichier peut : Apparaître, disparaître
  - Un fichier peut : Grandir, rétrécir
- Des blocs peuvent devenir libres, occupés...

Allocation des blocs

Au cours du temps...

- Beaucoup de modifications sur les fichiers. D'où
  - Un fichier peut : Apparaître, disparaître
  - Un fichier peut : Grandir, rétrécir
- Des blocs peuvent devenir libres, occupés...

Décisions sans informations

Lorsqu'un fichier est créé, le SE ne connaît pas sa taille future. Donc difficile de prendre une décision.

Allocation des blocs

Au cours du temps...

- Beaucoup de modifications sur les fichiers. D'où
  - Un fichier peut : Apparaître, disparaître
  - Un fichier peut : Grandir, rétrécir
- Des blocs peuvent devenir libres, occupés...

Décisions sans informations

Lorsqu'un fichier est créé, le SE ne connaît pas sa taille future. Donc difficile de prendre une décision.

Allocation des fichiers dans les blocs

A la création d'un fichier, le SGF doit :

- Attribuer de l'espace sur disque (c'est l'allocation) ;
- Mémoriser son implantation et son organisation sur le disque.
- Maintenir ces informations en cas de modifications de fichiers.

Allocation contiguë

Principe

Le système choisit parmi les ensembles de blocs libres contigus un ensemble de blocs libres contigus suffisant pour y implanter le fichier.

Allocation contiguë

Principe

Le système choisit parmi les ensembles de blocs libres contigus un ensemble de blocs libres contigus suffisant pour y implanter le fichier.

Avantages

- Accès direct à n'importe quelle partie du fichier.
- Vitesse d'accès.

Allocation contiguë

Principe

Le système choisit parmi les ensembles de blocs libres contigus un ensemble de blocs libres contigus suffisant pour y implanter le fichier.

Avantages

- Accès direct à n'importe quelle partie du fichier.
- Vitesse d'accès.

Inconvénients

- Des emplacements deviennent trop difficilement utilisables.
- Si le fichier augmente de taille après modification.

Allocation Non contiguë

Principe

Le fichier est découpé en bloc, les blocs de fichiers sont écrits sur des blocs du disque

Allocation Non contiguë

Principe

Le fichier est découpé en bloc, les blocs de fichiers sont écrits sur des blocs du disque

Avantages

- Rapidité d'enregistrement.
- Possibilité d'ajout de blocs si la taille du fichier augmente.

Allocation Non contiguë

Principe

Le fichier est découpé en bloc, les blocs de fichiers sont écrits sur des blocs du disque

Avantages

- Rapidité d'enregistrement.
- Possibilité d'ajout de blocs si la taille du fichier augmente.

Inconvénients

- Nécessité de gérer l'implantation de chacun des blocs sur le disque : le plus souvent liste chaînée (FAT).

Blocs d'index (UNIX) Une voie médiane

Principe

On utilise un bloc particulier le bloc d'index pour stocker la carte des blocs du fichier (pour chaque fichier). Un bloc étant représenté par son adresse.

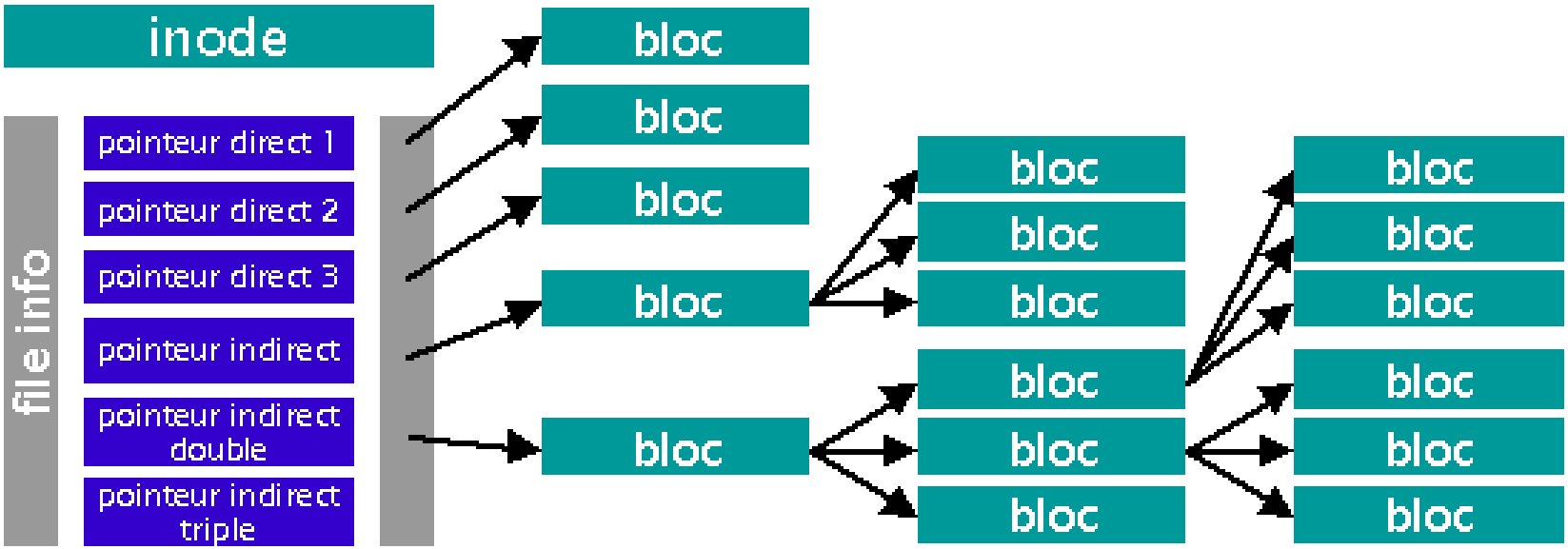
Mais pour les fichiers de grande taille

La taille de la carte des blocs peut nécessiter plusieurs blocs *i.e.* le bloc d'index n'est pas suffisant pour tout contenir. On chaîne entre eux les blocs d'index.

Blocs d'index à plusieurs niveaux

Un fichier d'index requiert plus d'un bloc ? Création d'un bloc-racine qui pointe sur la suite des blocs d'index. (réitération procédé cas échéant).
Permet d'accéder plus rapidement à la fin des fichiers de grande taille.

Blocs d'index, I-node (Sous Unix)



Description des fichiers par le système

Vous êtes bien gentil mais...

Je vais pas m'amuser à manipuler des listes chaînées !!

Structure de description d'un fichier

A partir de l'allocation des blocs comment représenter des fichiers par une structure simple.

- L'association d'un nom et d'un fichier sur disque est faite à l'aide de fichiers spéciaux, les catalogues (ou *répertoires* ou *dossiers*).
- Un catalogue est un enregistrement de taille fixe contenant des entrées qui fournissent :
  - le nom du fichier,
  - un accès à la structure de données décrivant l'espace qu'il occupe sur le disque,
  - diverses indications système (date, taille, etc.).

Catalogue : exemples

Sous MsDos

- la taille d'un catalogue est un bloc.
- Les indications système sont dans l'entrée catalogue, ainsi qu'un pointeur vers le premier bloc du fichier dans la FAT.

Sous Unix

Les catalogues reçoivent de l'espace disque = fichiers ordinaires.
Une entrée (appelée lien) contient

1. nom du fichier
2. un pointeur vers une structure de description : un noeud d'information (i-node) qui comporte
   1. des indications système :
      - ★ son propriétaire et son groupe ;
      - ★ ses droits d'accès ;
      - ★ ses dates de création, modification, accès ....
   2. les blocs qu'il utilise et l'accès aux blocs d'index.

Liens (Sous Unix)

Les liens plus de précisions

On peut avoir plusieurs liens vers un même fichier

- plusieurs entrées de catalogues pointent sur le même i-node
- La modification d'un i-node met à jour les informations dans tous les liens (car le i-node est partagé).
- L'opération d'effacement (rm) ne supprime qu'un lien. Les données ne sont perdues que lors de la suppression de tous les liens qui pointent sur le i-node.
- En cas de suppression, il y a récupération du i-node et mise à jour de la structure de l'espace libre (*i.e.* ajout des blocs occupés par le fichier à l'espace libre).

Accès aux structures de descriptions ?

Organiser son disque pour accèder aux structures de descriptions ?

Accès aux structures de descriptions ?

Organiser son disque pour accèder aux structures de descriptions ?

Début du disque

- Début de lecture : premier accès aux structures de données du disque En général, le bloc 0 est réservé au boot.
- Un bloc (sous Unix : le super-bloc) contient des informations relatives à l'organisation du disque (sa taille, l'adresse de la zone système....).

Accès aux structures de descriptions ?

Organiser son disque pour accèder aux structures de descriptions ?

Début du disque

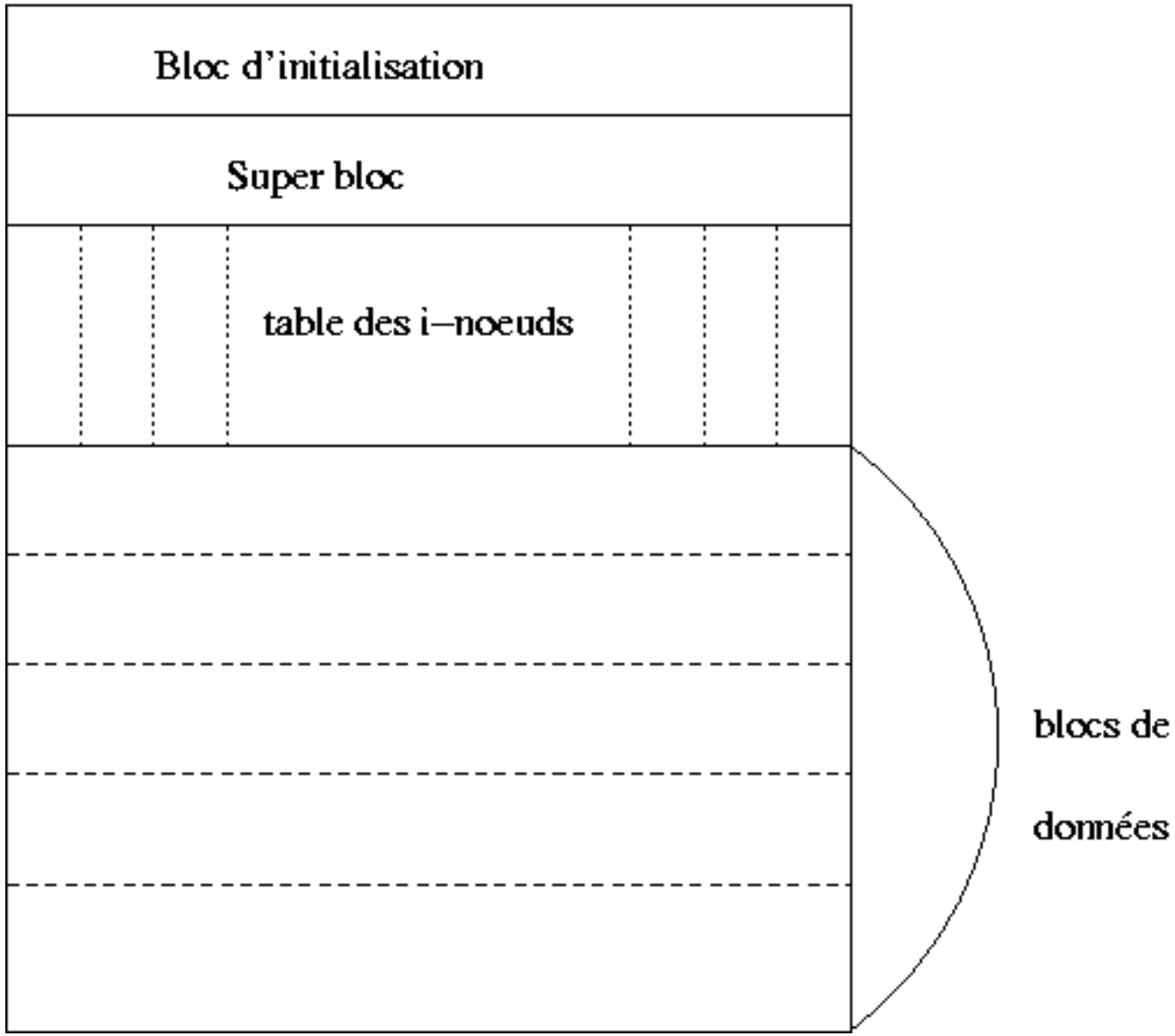
- Début de lecture : premier accès aux structures de données du disque En général, le bloc 0 est réservé au boot.
- Un bloc (sous Unix : le super-bloc) contient des informations relatives à l'organisation du disque (sa taille, l'adresse de la zone système....).

Une zone système

Les premiers noeuds sont réservés à des fichiers système (fichier des mauvais blocs, fichier de l'espace libre, catalogue racine du disque,etc.).

- Sous Unix par exemple, la zone système contient les i-nodes.
- Sous Dos, zone système est la FAT.

Description du disque (sous Unix)



Détail : Table des inodes et bloc (Linux ext2)

table des inodes

#	typ	droit	lnk	blck
123	d	700	2	3
126	d	750	2	7
140	-	644	1	8
142	-	644	2	9

Le bloc 3

152	..
123	.
126	Dossier
142	toto.txt

Le bloc 7

123	..
126	.
142	titi.txt
140	tutu.txt

Le bloc 8

Ce sont les données du fichier.
Elles sont très intéressantes.

Le bloc 9

Ce sont les données d'un fichier
Elles sont très intéressantes.
Super même !

Vue utilisateur du système de fichier

Un moyen d'accès aux fichiers pour l'utilisateur

- Les données sont sur le disque !
- Le système fournit des fonctions pour y accéder

  Convivialité comment accéder aux fichier de façon simple et transparente ?

  Efficacité ralentir le moins possible les accès aux fichiers

  Sécurité accès concurrent, autorisation d'accès

Vue utilisateur du système de fichier

Un moyen d'accès aux fichiers pour l'utilisateur

- Les données sont sur le disque !
- Le système fournit des fonctions pour y accéder

  Convivialité comment accéder aux fichier de façon simple et transparente ?

  Efficacité ralentir le moins possible les accès aux fichiers

  Sécurité accès concurrent, autorisation d'accès

Une interface standard

- Une vue utilisateur en arbre composée de dossiers (répertoires) et de fichiers.
- Chemins dans l'arborescence (suite de liens)

Vue utilisateur et montage

Construction de la vue utilisateur

- La vue utilisateur est construite par le système au démarrage, c'est le montage ou *mount*.
- La vue utilisateur peut être modifiée à n'importe quel moment (insertion cdrom par ex).
- Le montage consiste à associer :

  répertoire racine d'un disque $\leftrightarrow$ chemin dans l'arborescence
- Cette association entre tous les disques disponibles et le nom logique de leur répertoire racine est inscrite dans une table (la table des volumes montés) .
- Le disque monté sur le répertoire "/" contient donc la racine du système de fichiers.

Montage de partitions

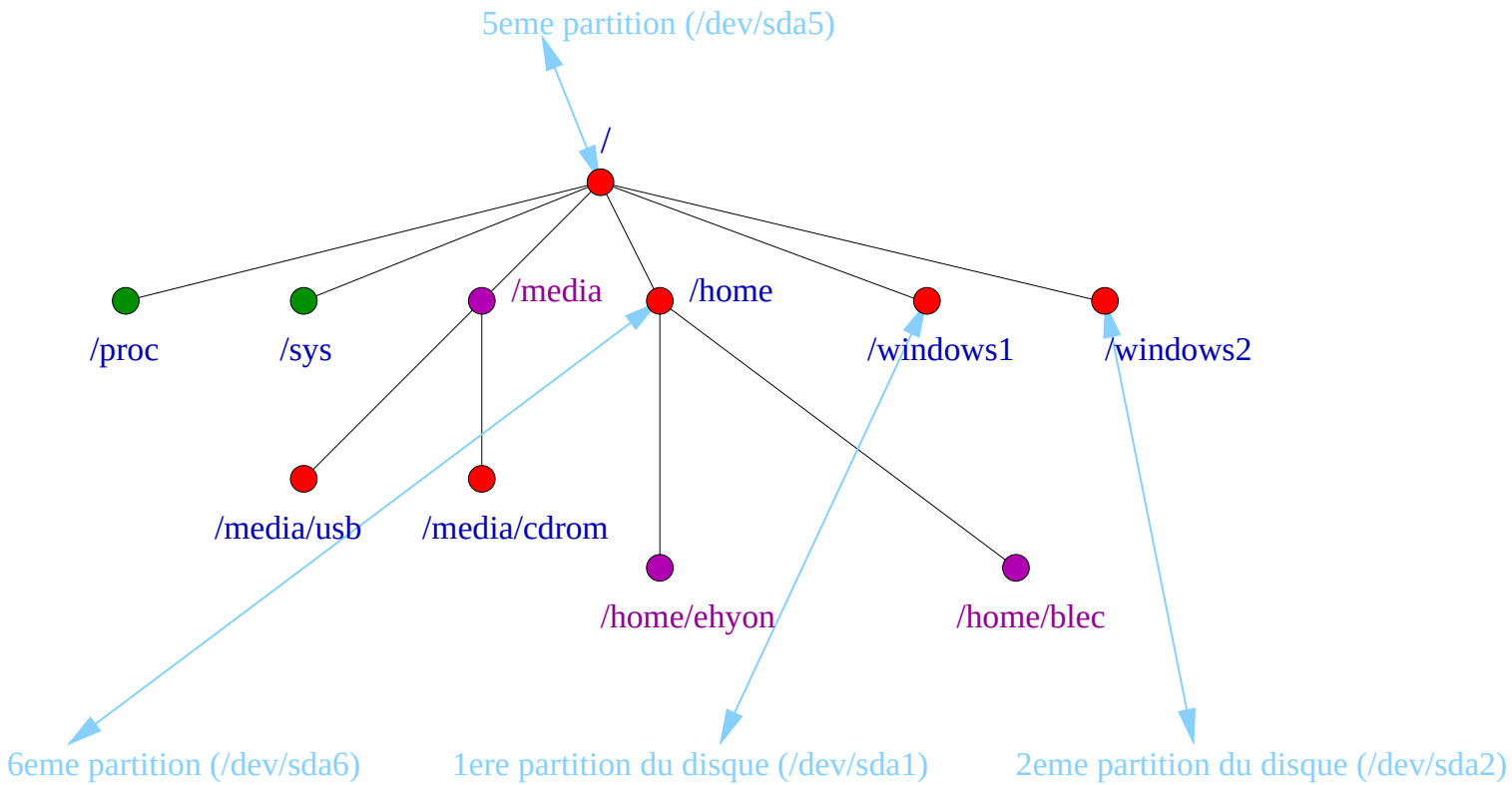
- Savoir ce qui est monté actuellement
  mount -l ou more /etc/mtab
- Pour monter un périphérique (en root)
  mount -t type fichier périphérique point_de_montage
- Savoir comment les périphériques vont être montés : le fichier fstab

Montage de partitions

- Savoir ce qui est monté actuellement
  mount -l ou more /etc/mtab
- Pour monter un périphérique (en root)
  mount -t type fichier périphérique point_de_montage
- Savoir comment les périphériques vont être montés : le fichier fstab

```
#<file system> <mount point> <type> <options> <dump> <pass>
proc /proc proc defaults 0 0
sysfs /sys sysfs defaults 0 0
/dev/sda5 / ext3 defaults,errors=remount-ro 0 1
/dev/sda6 /home ext3 defaults 0 2
/dev/sda2 /windows2 vfat defaults, iocharset=iso8859-15 0 0
/dev/sda1 /windows1 ntfs ro,auto,umask=2,gid=1001 0 0
/dev/sda3 none swap sw 0 0
/dev/scd0 /media/cdrom0 udf,iso9660 user,noauto 0 0
```

Représentation du montage associé



Mais qu'est ce qu'un fichier ?

Il y a différents types de fichiers

- Les Fichiers Réguliers
  Fichiers de données. Contenu : une suite de caractères sans organisation particulière. Caractéristique spéciale : la taille.
- Les Répertoires
  Fichiers réguliers et *fichiers spéciaux*. Contenu : une structure logique associant noms internes $\leftrightarrow$ noms externes. Fichier spécial car Appel système pour création.
- Les Fichiers Spéciaux
  Fichiers associés aux ressources du système (répertoire /dev).
- Les tubes nommés.
- Les liens symboliques.

Les fichiers spéciaux

Il y a différents types de fichiers spéciaux

- Les fichiers à usages logiques et non physiques
  - liens symboliques,
  - pseudo-terminaux,
  - sockets,
  - tubes nommés (interface entre disques)
- Les fichiers "physiques", pour communiquer avec des
  - périphériques : matériels physiques connectés à l'ordinateur (souris écran réseaux).
  - pseudo-périphériques (ou périphérique virtuel) : entrées gérées comme un périphérique mais non associées directement à un élément physique (ex écrans virtuels → écran, partitions du disque → disque).

Fichiers spéciaux II

Justification des fichiers spéciaux

- Les fichiers spéciaux ne sont manipulables que par l'intermédiaire du système.
- UNIX communique avec les périphériques (virtuels ou non) par l'intermédiaire de fichiers d'interface :
-
  - On accède à un périphérique comme on accède à un fichier.
  - Les fichiers ne contiennent pas de données, mais spécifient la manière de communiquer avec le périphérique en question.
  - Ce pilote de périphériques est une fonction du système. Il permet de manipuler les périphériques via les opérations classiques autorisées par les inodes : open, read, write, close.

Les fichiers spéciaux III

le répertoire /dev

```
brw-rw---- 1 root disk 1, 0 2007-09-20 13:51 ram0
brw-rw---- 1 root disk 8, 6 2007-09-20 11:51 sda6
crw-rw-rw- 1 root root 5, 0 2007-09-20 11:51 tty
```

Deux manières de communiquer

- Fichier spécial en mode bloc.
  Transfère les données vers le périphérique (disques durs, CD-ROM) en utilisant les buffers du système (accélération des transferts). Un périphérique qui peut être accédé avec un fichier spécial en mode bloc peut aussi être accédé par un fichier
- Fichier en mode caractère.
  Transfère les données vers le périphérique (raw devices : imprimantes, l'écran) sous forme de flux (un caractère à la fois) sans utiliser de buffer.

Liens symboliques

Définition

- Un lien symbolique est un accès indirect au fichier :
  Une référence à un autre lien, un lien secondaire sur le fichier correspondant.
- Différence avec le lien physique : il est possible d'effacer le fichier (effacer tous ses liens physiques) sans toucher aux liens symboliques (qui deviennent alors en quelque sorte orphelins : on parle de liens brisés).
- Nécessaire pour faire des liens sur des partitions différentes

Manipulation

Une commande pour la manipulation des liens :

`ln -s`

Accès aux fichiers

Comment accèder au fichier lors de programmes

Quels moyens le système met à disposition pour manipuler le fichier ?

Flux de données : Lien Disque Physique et RAM

Transmission des infos du périphérique en provenance ou vers la mémoire vive est un flux d'octets.

- Les opérations sur les flux passent par une mémoire tampon ("*buffer*").
- Une instruction d'écriture est une "écriture" dans le buffer.
- Ecriture sur disque quand le tampon est plein (ou sur instruction).
- Nécessité d'une association entre le flux de données et le fichier (spécial ou non).
- Grosso modo : programme écrit (ou lit) dans le tampon tandis que le S.E se charge de E/S directement sur le périphérique.

Accès aux fichiers (C/Unix)

2 niveaux d'accès aux fichiers.

Accès au fichier signifie manipulation accès au *buffer*.

- Manipulation haut niveau : Manipulation de fichiers
  - Utilisation de pointeurs de fichiers du type FILE*
  - Utilisation d'un buffer dans l'espace du processus et appel système de niveau 2 pour manipuler ce tampon.
  - Fonctions : *fopen*, *fclose* et autres contenues dans stdio.h.
- Manipulation bas niveau : Accès aux fichiers
  - Utilisation d'un descripteur de fichier.
  - Utilisation des buffers systèmes.
  - Fonctions *open*, *read*, *write* et autres fonctions manipulant des descripteur de fichiers.

Accès de haut niveau (1)

Structure `FILE`

- Les informations de l'association

  $$\textit{flux} \equiv \textit{programme}$$

  sont décrites dans une structure `FILE` (dans `stdio.h`).
- Dans le programme on gère un *pointeur* sur cette structure et le fichier sera identifié par ce pointeur.

Différents types de fichiers

- les fichiers *binaires* : Fichiers contiennent le codage machine (le codage en binaire) des données.

  Gain de place
- les fichiers *textes* : Fichiers formatés au format ASCII (Chaîne de caractère), *i.e.* un entier apparaîtra sous sa traduction en caractères.

  Grande Portabilité

Accès de haut niveau (2)
Ouverture d'un fichier

Création de l'association

C'est la commande `fopen`.

```
FILE *fopen(char *nom,char *name)
```

- La fonction renvoie :
  - Une variable de type `FILE*` et ouvre le fichier en paramètre.
  - Le pointeur `NULL` si l'ouverture ne se déroule pas normalement.
- Et prend comme arguments
  1. Le nom du fichier sous la forme d'une chaîne de caractères.
  2. Une chaîne de caractères qui spécifie le mode d'accès au fichier.

Accès haut niveau (3)
Mode d'accès

Mode d'accès et comportements

Pour les fichiers textes

Mode	Action	Conditions
"r"	lecture (*read*)	Le fichier doit exister
"w"	écriture (*write*)	Le fichier va être créé. S'il existe l'ancien fichier sera effacé par le nouveau fichier
"a"	écriture à la fin (*append*)	Les données vont être placées à la fin du fichier. S'il n'existe pas sera créé.
"r+"	lecture/écriture	
"w+"	lecture/écriture	
"a+"	lecture/écriture à la fin	

Accès haut niveau (4)
Exemple

Un exemple d'ouverture d'un fichier *examen.txt* à lire sur la clef USB (lecteur D sous Windows).

```
#include <stdio.h>

main(){
FILE *fichier;

  fichier=fopen(‘‘E:\\examen.txt’’,’’r’’);
  if (fichier == NULL){
   printf(‘‘j’arrive pas à lire’’);
   exit(1);
  }
}
```

Accès de haut niveau (5)
Autres opérations

Fermeture

La fonction est `fclose(flot)` où `flot` est de type `*FILE` donné par la fonction `fopen()`.

- Purge des tampons (flush).
- Ne pas fermer les fichiers peut provoquer des erreurs.

Lecture Ecriture

A l'aide des fonctions `fprintf`, `fscanf` similaires à `printf`, `scanf` et adjonction d'un flux.

- `fscanf (flot,''formatage'',expr1,...,exprn)`.
- `fprintf(flot,''formatage'',expr1,...,exprn)`.

Accès de bas niveau (1)

Les descripteurs

- Les descripteurs concernent les appels systèmes associés à l'utilisation de fichiers.
- Ils permettent les manipulations de bas niveau avec des écritures brutes (sans formatage).
- Un descripteur est une interface utilisateur présente sous la forme d'un entier.
  - Il correspond à un numéro de ligne dans une table (cf cours processus)
  - Il fait référence au fichier "dans le programme".

Les en-têtes nécessaires

Pour les tailles : `unistd.h`. Pour être cohérent avec POSIX : `types.h`.
Pour les droits `stat.h`. Pour les modes d'ouvertures : `fcntl.h`.

Accès de bas niveau (2)
Ouverture d'un accès

Primitive open

La primitive open() permet d'ouvrir un fichier à partir de sa référence.

```
int open(char *ref, int mode, ... mode_t droit);
```

Paramètres et comportement

- *ref : Référence (chemin) du fichier ou du tube nommé à ouvrir.
- mode : Mode d'ouverture du fichier.
- droit : (optionnel) droits des fichiers si création (sinon ce sont les droits par défauts de l'utilisateur)
- Cette primitive **retourne** :
  - en cas de succès : un descripteur valide (*i.e.* un entier positif).
  - en cas d'échec : -1.
- Cette primitive peut être bloquante.

Accès de bas niveau (3) *mode d'accès*

Modes d'ouverture

Le mode d'ouverture résulte d'un "ou binaire" (|) sur un ensemble d'options qui sont :

Mode	Action	Conditions
`O_RDONLY`	ouverture en lecture seule	Le fichier doit exister l'user doit avoir droit de lecture
`O_WRONLY`	ouverture en écriture seule	Le fichier doit exister
`O_RDWR`	écriture et lecture	
`O_APPEND`	Force l'écriture à la fin	
`O_CREAT`	Création du fichier si n'existe pas.	
`O_TRUNC`	le fichier est rendu vide à l'ouverture	

Accès de bas niveau (4) *Lecture Ecriture*

Les appels système read et write

Lire read écrire write des données dans un fichier préalablement ouvert (i.e. représenté par un descripteur).

int read(int desc, void *buf, size_t nbytes)
int write(int desc, void *buf, size_t nbytes)

Paramètres

- desc : descripteur valide sur le fichier dans lequel les données sont lues (écrites).
- *buf : un pointeur sur une zone mémoire dans laquelle les données (écrites) sont stockées.
  - Zone mémoire qui doit être allouée (via un tableau par ex.).
  - Type void * : données lues (écrites) octet par octet (pas interprétation ni formatage).
- nbytes : nombre d'octets que l'appel va tenter de lire. Le type `size_t` est équivalent à un type `int`.

Accès de bas niveau (5) *Lecture Ecriture*

Valeur retournée par read et write

- Succès : le nombre d'octets effectivement lus.
- Fin du fichier (pour le read()) : 0 est retourné (en effet il n'y a plus de données disponibles).
- Erreur : retourne -1.

Accès de bas niveau (6) *close*

l'appel système close()

Permet de fermer un descripteur et de libérer les ressources associées.

int close(int desc) ;

- Paramètre :
  desc : descripteur fermer.
- Retour :
  - 0 : en cas de succès
  - -1 : en cas d'échec
- Si oubli de fermeture, le S.E. détruira, en fin de programme, les ressources allouées pour ce fichier.
- Très préférable de fermer explicitement les fichiers qui ne sont plus utilisés (évite les problèmes).

Accès de bas niveau (7) Exemple

```
void main(){
  int desc;
  int data;
  int len;
  desc = open("Resultat", O_WRONLY|O_CREAT, 666);
  if(desc == -1)
        printf("Erreur pour ouverture du fichier Resultat");
  else{
        data = 3;
        len = write(desc,\&data,sizeof(data));
        if (len == sizeof(data))
            printf("La donnee a bien ete ecrite");
  }
}
```

Accès bas niveau (8) *Accès direct*

Position dans le fichier

L'*offset* d'un descripteur est la position dans le fichier ouvert de la prochaine opération.

appel système : `lseek`

Utilisation avec librairies sys/types.h et unistd.

```
lseek(int desc,off_t offset, int whence)
```

avec

- `desc` le descripteur du fichier
- `offset` le déplacement voulu (donné sous la forme d'un entier casté)
- `whence` point de référence :
  - `SEEK_SET` : offset absolu
  - `SEEK_CUR` : offset relatif à la position courante
  - `SEEK_END` : offset relatif à la fin