Fortier
Amandine
17-18

SYS

SYS

SYS

S1

File Alocation Table = FAT : c'est un système de fichier, il est propriétaire.

Diff dos et windows :

- Dos : monoprogrammation : 1 progrmme qui s'execute à la fois.
- Windows : multiprogrammation : plusieurs programme qui s'exécute en même temps.

Un système informatique :

- Un CPU
- Des programmes
- Des supports de stockage (disques)
- Une mémoire principale (RAM)
- Des imprimantes
- Clavier, écran, souris ...

OS

Système d'exploitation : logiciel, c'est du code. Exemple : linux, Windows, …
Noyau : linux.
Couche : distribution. Exemple : Ubuntu.

Il fournit aux programmes une interface simplifiée au matériel et à certains services.

Sert à exécuter des logiciel, cad, il prend l'adresse de l'exécutable et le met dans la RAM. Lorsque l'on exécute, c'est toujours quelque chose en mémoire JAMAIS directement apd disque.

Ajd on travail en mémoire virtuel : une partie du programme que l'on exécute mais qu'on n'utilise pas directement.

Un OS est une machine étendue car ce n'est plus simplement le CPU, il y a le disque etc. Les périphériques (disque, clavier, …).

Bash est un interpréteur de commande = shell. Lorsque l'on tape « ls », bash charge « ls ».
L'OS est un gestionnaire de ressources : il gère la RAM, le disque, etc ça s'appelle l'ordonnancement. L'ordonnancement à des tables (process pages)

SYS

L'utilisateur n'interagit pas directement avec l'OS, il exécute certains programmes :

- Via une interface graphique GUI
- Via un interpréteur de commande en mode console

Ce sont les programmes qui interagissent avec l'OS via des Appels Système. En exécutant un programme que vous avez écrit ou même juste la commande ls dans un terminal, vous lancez un exécutable qui interface l'OS, vous faites de même en lançant une application graphique.
Appel système : service que l'on demande au système. Exemple : read : lire quelque chose dans un fichier, write : écrire quelque chose dans le fichier. (Service read, service write)

Traitement d'interruption, mécanisme d'interruption : lorsque je bouge la souris, le CPU va interrompre ce qu'il fait pour que visuellement on voit la souris bouger. Ça vient de l'extérieur (souris, …).

Les restrictions, il faut cloisonner, lorsque 2 logiciel sont executé, le 1er ne peut pas écrire qqchose dans la mémoire du 2ème.
On parle de ring, ring0 , ring3 c'est le ring de l'utilisateur.

- Le code d'un utilisateur, s'exécute en mode non privilégié, (ring 3)
- Le code de l'OS s'exécute en mode privilégié, il a tous les droits (noyau, superviseur, ring 0 ...)
- Il a accès à la totalité des instructions et à la totalité de la RAM

Quand le CPU est en mode "superviseur" il exécute le code de l'OS. Le code qui s'exécute peut accéder l'entièreté de la RAM et peut exécuter n'importe quelle instruction. Les instructions de bas niveau qui permettent l'écriture sur un périphérique de type disque sont des instructions réservées à l'OS, on dit qu'elles sont "privilégiées".

Toutes les applications et programmes doivent passer par le code de l'OS.
Comprendre le fonctionnement de celui-ci permet d'en comprendre les limites qui se répercutent sur les programmes également.

L'OS est matérialisé notamment par

- Du code
  - Les appels système
  - Les traitements d'interruption
  - L'ordonnanceur
- Des données
  - Des tableaux et données structurées complètent ces bouts de code en RAM. Elles décrivent les ressources de l'OS (processus, fichiers ouverts ...)

La présence d'un ordonnanceur est requise en multiprogrammation. Il assigne le CPU aux programmes en les alternant. On ajoute à cela quelques démons (programmes qui réalisent un service) qui tournent en permanence comme le démon d'impression, les services réseau, …

SYS

L'OS s'occupe de :

- Organiser les disques en systèmes de fichiers
- Gérer les processus - création, mort, état
- Gérer l'accès au CPU - ordonnancement (partage du CPU)
- Permettre la communication entre processus
- Gérer la mémoire - mémoire virtuelle
- Gérer l'accès aux ressources non partageables (imprimantes)
- Gérer les périphériques souris, clavier ...
- Gérer des services au travers de démons ...

.13

Multiprogrammation : but : rentabilisé les machines qui coûte cher.
DMA : Direct Memory Access, ont lui accorde un processeur.
Ordonnanceur dit qui sera le process suivant.
Problématique des interblocage : car ils ont besoin des mêmes ressources.

Révision

1 Kib	1024b	2^{10}b
1 Mib	1024 Kib	$2^{10}.2^{10}b=2^{20}b$
1 Gib	1024.1024.Kib	2^{30}b

$2^{12}b = 2^2.2^{10}b = 4096b$

Pourquoi utiliser des multiples de 2 allège énormément les calculs.
Car il suffit de bouger à gauche ou à droite. A droite c'est MOD et a gauche c'est DIV.

S2

SYSTEME DE FICHIERS

Système de fichiers = organisation d'une partition d'un disque.

Machine étendue

Toutes les applications doivent passer par un OS.
Ring3 est le moins privilégié, niveau application.
Ring 0 niveaux OS, il a accès à la totalité de la Ram et aussi à toutes les instructions du CPU, seul l'OS peux accéder.

Gestionnaire de ressources

Qu'est-ce que l'OS gère :

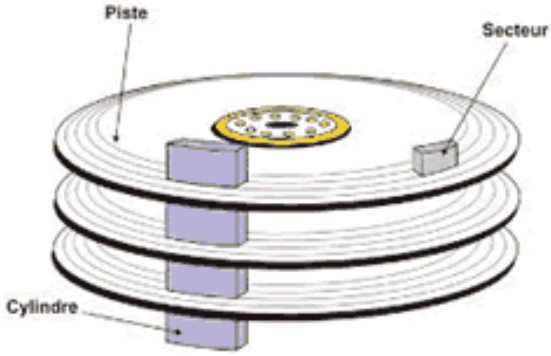
- Attribue la RAM au processeur
- Attribue les processus/le temps CPU = ordonnancement.

SYS

Disque dur et leurs organisations

Disque :

- Est une mémoire secondaire (mémoire de masse)
- Contient des secteurs (formatage de bas niveau)



Secteurs :

- Unité physique de stockage et d'échange
- Un secteur = 512=2^9 (2048) octets de données utiles plus quelques octets d'information redondante permettant de valider son contenu. Ne change jamais de taille. C'est la plus petite taille utilisable.

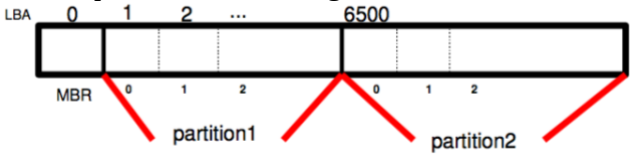
Disque à plateaux :

- Piste/trak : c'est la circonférence parcourue par la tête de lecture pendant une rotation de disque, elle est identifiée par le couple (Cylinder,Head). Sur une piste se trouvent plusieurs secteurs numérotés de 1 à 63.
- Cylindre : est l'ensemble de pistes parcourues sans déplacement de têtes (Heads). Pour changer de cylindre sur un disque : opération mécanique(lent).

Numérotation/addressage :

- CHS cylinder head sector : ancienne notation lié au disque mécanique, permet d'identifier un secteur sur le disque, le cylindre, la tête et le secteur tiennent en 3bytes(24bits) donc 224 = 16Misecteurs (C=10octets H=8octets S=6octets). N'est plus adapté aux tailles de disque actuelles : < (2^{24} secteurs).
- LBA logical block address : un disque est une suite de secteurs n° apd de 0. Le MBR a comme LBA 0. Codé sur 4bytes, (8bytes GPT), LBA c'est un nombre entier qui indique un numéro d'ordre, pour ordonner les secteurs, cylindre, … Taille maximum de disque avec 4bytes = 2Tib.

MBR : participe à la séquence de démarrage. Contient les infos du disque.



Partitionnement et formatage :

Partitionnement : Un disque est subdivisé en partitions (C:, D:, /dev/sda1, /dev/hda1, ...) Commandes de partitionnement (fdisk, parted, ...), le partitionnement découpe, ça ne créé que des boite. Il prend plusieurs secteurs.

Formatage : Chaque partition peut être organisée selon un format différent (système de fichiers) en linux mkfs(make file system).C'est ici qu'on attribue un système de fichier.

Une partition sera donc organisée en système de fichiers (FAT, NTFS, …) Chaque partition a son propre système de fichier et son propre secteur 0.
Une deuxième numérotation des secteurs au sein d'une partition coexiste avec la précédente et commence également à 0.

SYS

Organiser une partition (formater) sert a :

- Stocker des grandes quantités d'informations de manière permanente.
- Nommer ces informations (fichiers)
- Partager ces informations
- Retrouver (localiser) les informations grâce à leur nom/chemin et non via des numéros de secteur.

Il y a 2 vue d'un système de fichiers :

- Vue utilisateur du service, quel service ? (Créer un fichier, y écrire, changer les droits, ...)
- Vue système du service, comment faire cela ?(Mise en œuvre - code des Appels Système read, write, ...)
-

Rôle de système de fichiers :

- Définir implantation des fichiers et répertoires
- Localiser les fichiers/répertoires et leurs données (métadonnées et données)
- Allouer de l'espace aux fichiers/répertoires
- Gérer l'espace libre, les quotas, …
- Offrir l'encryptions, les droits, …

Localiser sur un disque : numéro de secteur et position dans le secteur.
Localiser les métadonnées d'un fichier = localiser sa description.
Localiser un byte du fichier = localiser ses données.
L'allocation (contiguë, par blocs) de l'espace aux fichiers est un point sur lequel on va s'attardes dans la suite.

Qualité d'un système de fichier :

- Fiabilité
- Performance
- Portabilité (devices externes)
- Tailles maximums des partitions et fichiers

Les systèmes propriétaires ne sont pas portables. Même EXT et FAT ne sont pas censé.

Vocabulaire :
Données des fichiers et répertoires
Méta-données des fichiers et répertoires (droits, taille, propriétaire, date de création, …)
Méta-données du système de fichiers (taille des secteurs, localisation de la racine, taille des blocs, blocs libres, blocs défectueux, fichiers remarquables, …)

SYS

Allocation contiguë ou par blocs

taille secteur = 512b

Allocation de l'espace disque :

- Allocation contiguë : l'un derrière l'autre. Chaque secteur se suit. Tous les octets de données d'un fichier sont consécutifs.
- Allocation par blocs : Les blocs de secteurs sont disséminer dans le disque.

Contiguë :

Où est le byte N :
N<longueur du fichier.
On cherche :

x = le n°de secteur qui contient le byte N
position = la position du byte N dans x (0,..)

x = n°secteur début du fichier + N DIV TailleSecteur
position = N MOD TailleSecteur

Métadonnées nécessaires à la localisation d'un byte d'un fichier :

- Métadonnées du système de fichiers :
  - La taille du secteur
- Métadonnées du fichier :
  - N°de secteur de début
  - Longueur du fichier en bytes

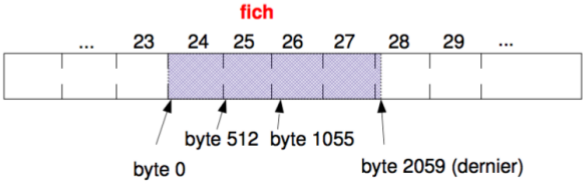
Exemples :
F35,2100 (début, longueur) N = 1000 ?
secteur 35+(N DIV TS(taille secteur = 512b)) = 35+1=36
décalage 1000 MOD 512 position dans le secteur : N MOD taille secteur
N = 512 ? Oui

un fichier démarre au secteur 24 et contient 2060 bytes.

le byte n° 1055 du fichier se trouve :
secteur : 24 + (1055 DIV 512) = 26
byte dans le secteur : (1055 MOD 512) = 31

le byte n° 512 du fichier se trouve :
secteur : 24 + (512 DIV 512) = 25
byte dans le secteur : (512 MOD 512) = 0

ce fichier occupe :
(2060 + 511) DIV 512 = 5 secteurs



Performance :
Fragmentation Externe

- L'espace libre du disque s'est fragmenté
- Il n'est plus possible de créer de nouveau fichiers sans devoir déplacer d'autres fichiers.
- Les déplacements sont coûteux en écritures

Fragmentation externe = fragmentation de l'espace libre

- Rapidité d'accès en lecture
- Lenteur inadmissible réécritures dues à la fragmentation externe suite à des modifications -> convient aux systèmes de fichiers en lecture uniquement

Par blocs/cluster :

Blocs par 2N secteur N est un multiple de 2. Pour 1 byte on utilise 64Kib au minimum.
Bloc = unité d'allocation Bloc = unité d'accès logique

Un fichier occupe toujours un nombre entier de blocs L'espace alloué aux fichiers est non nécessairement contigu.
Des métadonnées supplémentaires sont nécessaires pour décrire le chaînage des blocs d'un fichier.

Les blocs commencent à la frontière d'un secteur
Les blocs sont numérotés
Le n°de bloc de la partition (P), permet de calculer le numéro de son premier secteur (ce n°de secteur est l'adresse du bloc P)

- Adresse de P = P*"nb Secteurs par bloc"

Les métadonnées du système de fichiers occupent également de l'espace disque souvent en dehors des blocs De ce fait, le bloc 0 n'est généralement pas aligné avec le secteur 0.
Donc : adresse P = "adresse du bloc 0" + P x "nb secteurs par bloc"

C'est dans les métadonnées que FAT, NTFS, … se différencie.

Exemple : Soit une partition divisée en blocs : bloc = 4 secteurs ici. Soit le bloc 0 aligné sur le secteur 8 de la partition

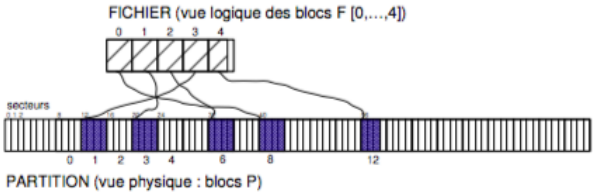
Le bloc 0 commence au secteur : 8 + (0 x 4) = 8
Le bloc 3 commence au secteur : 8 + (3 x 4) = 20

Deux numérotations pour le même bloc (F, P)

- F = numéro d'ordre du bloc au sein du fichier (un fichier de 2000 bytes contient deux blocs (F=[0,1]) de 1024 bytes). Il s'agit d'une vue logique.
- P = numéro d'ordre du bloc au sein de la partition (vue physique)

Exemple :

▸ F=4 donne P=12 dans l'exemple



- Ajout/suppression de données sont plus aisés car la taille des blocs est fixe et les fichiers non contigus
- Les blocs des fichiers ont tendance à s'éparpiller (fragmentation des fichiers) - les déplacements de têtes deviennent pénalisants à la lecture/écriture.
- On voit apparaître une perte de place importante au sein du dernier bloc du fichier pas toujours rempli à 100% (fragmentation interne)

SYS

Espace perdu par fragmentation interne. Car obliger de prendre min 64Kib.
Principe : disséminer des blocs de fichier dans des blocs du disque, de même taille. = fragmentation des fichiers.

- fragmentation interne

espace perdu dans le dernier bloc

- fragmentation des fichiers

dispersion des blocs d'un fichier

Allocation par blocs : + souple pour les mises à jour de fichiers, trop sujet à la fragmentation de fichiers nuisant la rapidité d'accès, trop sujet à la fragmentation interne et donc à une utilisation partielle de l'espace disque.

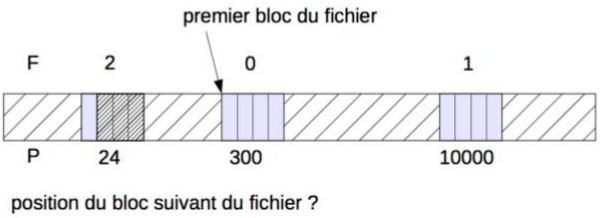
Le début et la longueur ne suffisent plus à localiser un byte N si celui-ci est plus loin que le premier bloc du fichier (F>0).

Comment établir la correspondance entre F et P dans ce cas ?
Des métadonnées supplémentaires doivent nous renseigner sur le chaînage des blocs au sein des fichiers. Ce sont des métadonnées des fichiers. Mais FAT en fait des métadonnées du système de fichiers via la table d'index.

Quand je suis en train de parcourir un fichier, j'ai besoin tout le temps de parcourir la table d'index (celle-ci est au tout début du disque pour ça que perte de vitesse des tête lecture écriture)

Le byte N d'un fichier se trouve dans le bloc P correspondant à son bloc n°F
F = (N DIV TailleBloc)
Mais à quel bloc P de la partition correspond F ?
F = n°du bloc dans le fichier
P = n°du bloc de la partition

Comment établir la correspondance entre F (numéro d'ordre du bloc dans le fichier) et P (sa position dans la partition) ?



Différentes approches permettant de localiser chaque bloc du fichier :

- Blocs chaînés
- Un index par fichier (EXT, NTFS)
- Table d'index globale pour les blocs (FAT)

Chainage par table d'index

Tableau d'index : c'est une partie des métadonnées. C'est celle-ci qui donne le n° des blocs dans l'ordre.

Cas de la table d'index :

- Une zone de la partition est réservée pour y accueillir un tableau de description des blocs de la partition.
- Ce tableau aura nécessairement autant d'entrées qu'il y a de blocs dans la partition
- Chaque bloc peut être libre, défectueux ou chainé au sein d'un seul fichier.

Dans ce dernier cas, le tableau contient le numéro du bloc suivant celui-ci dans le chaînage ou une marque de fin de fichier s'il est dernier d'un fichier.

Si dans le tableau à l'indice 4 je trouve la valeur 9 cela veut dire que le bloc 4 contient des données du fichier et la suite des données se trouve dans le bloc 9.

Implantation des fichiers : table d'index

Table d'index : une entrée par bloc de la partition

- La table décrit les blocs
- Le tableau a autant d'éléments (entrées) qu'il y a de blocs sur la partition
- Le nombre à l'indice i de l'index indique le numéro du bloc suivant i s'il y en a un.
- Un bloc n'appartenant qu'à au plus un fichier, chaque bloc peut indiquer son suivant (au sein du fichier)

Un bloc n'a pas toujours de suivant, il peut être le dernier d'un fichier.
Un bloc libre n'a pas de suivant non plus mais on pourrait chaîner tous les blocs libres pour les trouver

- L'index décrit ainsi le chaînage des blocs dans les fichiers
- Si un bloc indique son suivant au moyen de la table, chaque fichier peut être reconstitué à condition de connaître la position P de son bloc 0 (premier bloc)
- P est le premier bloc du fichier et le premier indice auquel on regardera.

L'emplacement du premier bloc d'un fichier est une métadonnée du fichier et sera renseigné dans le répertoire parent.

L'entrée de l'index a une taille fixe, elle contient un nombre entier (sur 1, 2, 4 bytes, ...)
Le plus grand nombre entier qu'on peut représenter dans l'index dépendra du nombre de bits dédiés à la représentation de cet entier.
Par exemple sur un short (16 bits) on peut représenter 2exp16 valeurs différentes (0 -> 2 exp 16 -1)
Cette limite est donc aussi la limite au nombre de blocs du système de fichiers !

Il est intéressant de remarquer que le choix de la taille des entrées de l'index contribue à limiter la taille du système de fichiers pour une taille de bloc donnée.
Nous y reviendrons en parlant de la FAT.

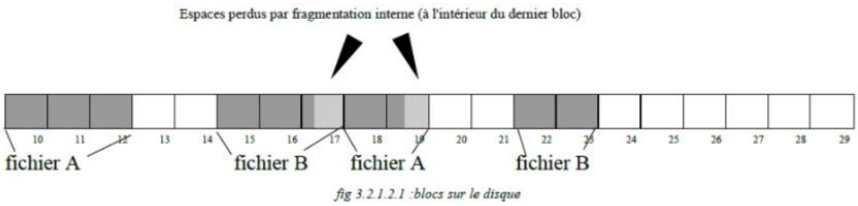
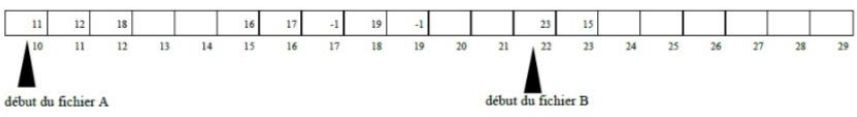


fig 3.2.1.2.1 :blocs sur le disque



SYS

Reconstituons le fichier A : 10-11-12-18-19
A démarre au bloc 10 (métadonnée du fichier) l'entrée 10 de l'index donne le bloc suivant de A : 11. . . la valeur -1 dans l'index marque le dernier bloc du fichier

Performance

Constat :

- Les blocs d'un même fichier sont éparpillés
- Chaque accès à un nouveau bloc, demande un double accès disque (table d'index et bloc de données)

-> pour améliorer les performances la table d'index est chargée en RAM au démarrage !
Désavantage : Sa grande taille cause un ralentissement du système au démarrage (faiblesse de la FAT)

Localiser

Qu'en est-il de la problématique de la localisation d'un byte du fichier dans ce cas ?
Quelles métadonnées sont nécessaires ? :
Métadonnées du système de fichiers :

- Taille du secteur
- Nombre de secteurs par bloc | taille du bloc
- Début du bloc 0
- Table d'index du système de fichiers

Métadonnées du fichier :

- Bloc physique associé au premier bloc du fichier
- Longueur du fichier (occupation du dernier bloc)

Localiser le byte N ?

Traduire "byte N" en "position dans le secteur x"

1) F : position du byte N dans le fichier (n°bloc) : F = N DIV tailleBloc
2) Bloc correspondant à F sur la partition : P = (*)
3) Adresse du bloc P = début du bloc 0 + (P x nbSecteursParBloc)
4) x = (3.) + ((N MOD tailleBloc) DIV tailleSecteur)
5) Position du byte N dans x = N MOD tailleSecteur

(*) P est trouvé grâce à l'index (chaînage)

Exemple

Si le bloc 0 est aligné sur le secteur 0 et un bloc mesure 1Kib (1024b)

1) Pour le fichier B de l'exemple on cherche N=3600 :
2) F = 3600b DIV 1Kib = 3
3) P = bloc 16 (22-23-15-16-...)
4) Adresse (secteur) du bloc 16 ? 16 x 2 = 32
5) x = (secteur contenant N sur la partition)
6) x = 32+(3600b MOD 1024b) DIV 512b = 33
7) Position de N dans le secteur 33 = 3600 MOD 512 = 16

Le byte 3600 de B est dans le secteur 33 de la partition à la position 16.

Exercices :

Étant donné la table d'index suivante, des secteurs de 512 bytes et des blocs de 1Kib :
| | | | |8| | | |10| |-1| | …
Localisez le byte 1400 (n°secteur-position) dans le fichier qui démarre au bloc 4
1400DIV1024 donc se trouve au 2ème bloc (le n°1) 1400MOD1024 = 376 = décalage
Le bloc 4, 8 et 10 compose le fichier. Comme on veut le secteur et non le bloc, faut faire x2 Secteur : x=32+(8x2) = 48 ➔ on a imaginé que le bloc 0 démarre au bloc 32
8 car on prend le bloc suivant.

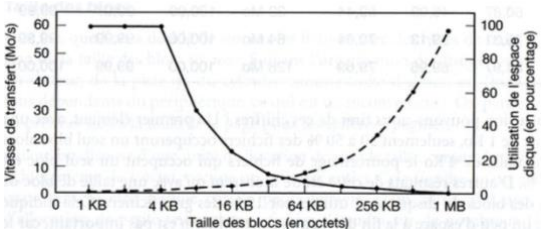
SYS

Taille des blocs

Quand je fixe une taille je fixe aussi une limite aux nombres de blocs
Fragmentation interne
Le dernier bloc n'est pas toujours entièrement rempli
Fragmentation interne = espace perdu à l'intérieur des derniers blocs des fichiers
Choisir la taille à donner aux blocs : un choix stratégique

- Grands blocs -> perte de place
- Petits blocs -> perte de temps
- Taille d'1 bloc = n secteurs partition (n >= 1)
- Choix de n fait au formatage

64KB taille de bloc meilleur car entre les 2



2 à 4 Kib = taille moyenne des fichiers sur un système Unix [TNB]
Avec des blocs de 8Kib 50% de l'espace disque est perdu ! avec des blocs de 64Kib 90%!

Gain de place ou performance ?

« Avec des partitions > 1Tib il peut être préférable d'augmenter la taille des blocs à 64Kib et d'accepter un gaspillage de l'espace disque » [TNB]
Dans ce cas, 1Tib -> +/-100 Gib utiles !

Performance

Fragmentation des fichiers

Allocation par blocs -> fichiers fragmentés (blocs éparpillés)
-> beaucoup de déplacements des têtes -> LENT ->
Solutions :

- Réorganisation systématique (adopté par linux)
- Outils de défragmentation fournis (xp, seven, vista)

Questions

- En allocation de l'espace par blocs, une plus grande taille de bloc augmente la fragmentation externe[V-F] F c'est l'interne
- Quelle différence y a-t-il entre fragmentation externe - interne - des fichiers ?
- Soit un fichier de 10360 bytes et une taille de bloc de 4Kib en allocation par blocs. Combien de blocs sont utilisés par ce fichier ? Quel pourcentage du dernier bloc est occupé ? 3 et 50%

Répertoire

Implantation des fichiers : répertoires
Répertoire : (fichier structuré) fichier particulier qui contient quelques métadonnées des fichiers ces métadonnées permettent notamment la localisation des fichiers.

SYS

Localisé d'un fichier métadonnée

Différentes solutions :

- ext - le répertoire parent contient les numéros d'inode des fichiers
- FAT - le répertoire parent contient le premier bloc des fichiers dans la partition
- NTFS - le répertoire parent contient le numéro d'entrée du fichier MFT décrivant chaque fichier du système

il ne faudra pas confondre localisation d'un byte d'un fichier dans la partition (secteur x, position) et localisation du premier bloc d'un fichier dans la partition (bloc P).
Dans ce dernier cas on dira on dira qu'on va "localiser un fichier"

Répertoire

Un répertoire est un fichier particulier dans le sens qu'il décrit d'autres fichiers. Ses données sont les métadonnées des fichiers qu'il contient (FAT) ou permettent de retrouver ces métadonnées (EXT, NTFS)

Métadonnées des fichiers

Exemple de métadonnées d'un fichier :

- Nom
- Premier bloc (valeur de P pour F=0)
- Longueur du fichier
- Attributs divers
- N°de l'entrée de la MFT (NTFS)
- N°d'inode(EXT)

Localiser un fichier :

- Localiser un fichier => lire le répertoire parent
- Un répertoire est un type de fichier
- Et pour lire le répertoire ? -> lire son parent !

Où cela s'arrête ? Qui est le parent du répertoire racine ?

Localiser un fichier : Avec son chemin apd la racine. Si je dois lire le fichier, je dois lire le répertoire parent (celui du dessus, donc on doit tous les lire, l'un après l'autre, joue un rôle important) du coup, où se trouve la racine ? chaque système de fichier choisis.

Commencer par la racine

La racine doit avoir un emplacement fixe connu ou calculable.EXT, FAT, NTFS utilisent des conventions

- Position calculable ou renseignée à un endroit calculable (FAT, NTFS)
- Renseigné à une position fixe (EXT : inode 2)

Lire le fichier /home/mba/test demande de localiser, et lire 3 répertoires :

- /
- /home
- /home/mba

Ce dernier donnera les informations qui permettent de localiser et finalement lire le fichier test.

Systèmes de Fichiers

Appels système

Un Système d'exploitation mettra à disposition des applications les services pour utiliser les systèmes de fichiers. Un système Différentes approches permettant edlinux prévoit notamment :

- open localisation d'un fichier pour une session de lecture/écriture, il retourne un entier qui permet de localiser.
- close cloture de la session de lecture/écriture

La localisation d'un fichier est une opération longue (plusieurs lectures de répertoires).
On travaille plutôt par sessions. Une seule localisation pour plusieurs lectures/écritures.
Open renvoie un entier.Sa met en ram l'association entre le les métadonnées du fichier.

Quelques Appels Système pour les fichiers

- read - lire des bytes (localiser les bytes)
- write - écrire des bytes (localiser les bytes)
- lseek - modifier la position de lecture/écriture

Une librairie de fonctions C permet d'utiliser chaque appel système.
Les pages de manuel man pages de niveau 2 décrivent cette librairie.
bash>man 2 read
Niveau 1 : commande ; niveau 2 : appels système ; niveau 3 : libC.
man 2 open on doit spécifier le niveau.

Open/Close

La localisation du fichier est mémorisée dans la TDFO par l'appel système open.
TDFO = Table de Descripteurs de Fichiers Ouverts
Lire un fichier se fait généralement par plusieurs appels consécutifs à l'Appel Système read
La position courante (prochain byte à lire ou écrire) est mémorisée dans la même table et sera mise à jour par les appels système (read, write, lseek, . . .)
L'appel système close, libère l'entrée de la table

Open, read, write, lseek, close

que dit **man 2 open** pour linux ?

```
int open(const char *pathname, int flags);

L'appel système open() sert à convertir un chemin d'accès en descripteur de fichier (un petit entier non négatif utilisable pour les opérations d'entrée-sorties ultérieures telles read, write, etc.).

ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);

int close(int fd);
```

remarquez le paramètre entier **fd** qui sert à identifier le fichier ouvert.

char * \0 = adresse de chaine de caractère.
void * = aussi adresse mais non typé c'est une zone de mémoire.

O_CREAT O_WRONGLY: si n'existe pas ont créé le fichier.
O_TRUNC : tronque

vi usr/include/bits
Le 0 avant c'est du C pour dire que c'est de l'octal.

IMG = CREE FICHIER QUI CONTIENT UN TEXTE.
Appel système Write : d'abord écraser rax car c'est la qu'on met appel système
Mov rdx,8 c'est la taille de bonjour.
Dans EXT on n'utilise pas table d'index mais inode.

Questions

- L'appel système open (/home/mba/test) lit trois fichiers [V-F] V, mba, home, racine
- L'appel système read (fd,&char_lu,1)) lit plusieurs blocs du fichier [V-F] F, 1 byte ne lis pas plusieur byte.
- Localiser le répertoire / demande de lire son parent [V-F] F

FAT

FAT = File Allocation Table avec les premier PC personnel. Fat c'est un format public.
Hardware White Paper dis exactement comment est construit une FAT.
Système de fichiers de DOS (dérivé de CP/M unix), reconnu par d'autres OS : Windows et linux …
Reverse engineering : faire de l'ingénierie a postériori, exemple : faire un dump(faire une copie binaire du HDD, ajouter un fichier et refaire un dump pour voir ce qui a changer, voir comment est fait NTFS).

Allocation

Allocation de l'espace par blocs avec index

- CLUSTERS (blocs) clusters pour Windows.
- FAT File Allocation Table (index)
  - FAT12 (12 bits)
  - FAT16/vfat (16 bits)
  - FAT32 (28/32 bits) 28 car en réalité ➔ 28 au lieu de 32

12-16-28 est la taille en bits des entrées de l'index, la taille doit être enregistré dans la RAM, peut être contraignant si prend bcp de place.
Cela correspond à 1.5, 2 et 4 bytes respectivement

SYS

Structure d'une partition FAT16

Métadonnées et données du système de fichiers, on commence au cluster 2, il n'y a pas de 0 et le 1 est réservé.

- Boot sector : métadonnées du système de fichiers, connaît la taille de tout.
- FAT : index (souvent en deux exemplaires)
- Répertoire racine contigu et de taille fixe, se situe entre table et répertoire fichier, ce n'est pas un fichier chainé, il a 512 entrées dont 511 utilisable et 1 qui reprend le nom du volume. En 32bits, la convention c'est que le n° du 1er bloc est renseigné dans boot, souvent 2 sauf si défectueux, cad il renseigne direct l'espace fichiers.
- Clusters : blocs des fichiers et répertoires, les clusters ne sont pas alignés sur le début de la partition

En FAT 32, le répertoire racine est chaîné comme les autres fichiers.

Taille des clusters

Taille d'un cluster : définie au formatage
1 cluster = n secteurs (512 bytes), n=puissance de 2 :

Table : Tailles des clusters

exposant	0	1	2	...	6	7
nb secteurs	1	2	4	...	64	128
taille cluster	512b	1Kib	2Kib	...	32Kib	64Kib

Limites

Certaines versions de Windows limitent les clusters à 32Kib.
Il est important de distinguer

- Les limites dues au format
- Les limites dues à l'utilisation

-> on parlera plutôt des limites dues au format (théoriques)

Taille maximum

Nous savons déjà que le nombre de clusters d'une partition est limité et que cette limite dépend de la taille de l'entrée de l'index
Connaissant la taille maximum d'un cluster, nous pouvons en déduire la taille maximum de la partition.
Nb max clusters x taille max cluster

Exercices

Taille max partition FAT16 : cmb de bloc max sur 16bits : 2^{16}
taille du + grand bloc = 64Kib = $2^{16} * 2^{16}b = 2^{32}b$ = 4Gib

Taille de la + grande table d'index FAT16 : maximum 2^{16} entrées, 16bits pour codée un nombre, donc 16 différents.
Taille de la + grande table = 2^{16}*2bytes(2bytes = 16 bits) = 2^{17} = 128Kib

FAT32 la + grande partition : bloc max : 2^{28}
Taille + grand bloc = 64Kib(ne change pas) = $2^{28}*2^{16} = 2^{44}$ = 16Tib mais avec MBR on ne pouvait pas dépasser 2Tib.

SYS

FAT32 taille + grande table d'index : maximum 2^{28}entrées,
Taille + grande table: 2^{28} * 4bytes (4bytes = 32bits) 1Gib

Table FAT

FAT = index

- Contient une entrée par cluster de la partition
- Décrit le chaînage des clusters des fichiers
- Chaque entrée contient le numéro du cluster suivant au sein du fichier

FAT : valeurs réservées (FAT12, FAT16) :

- 0 - cluster disponible
- -1 - dernier cluster du fichier
- Cluster réservé MS-DOS
- Cluster défectueux ➔ -2

Choisir 16 ou 32 taille partition

Taille maximum théorique d'une partition FAT

- La taille maximum théorique d'une FAT32 est 16Tib
- La taille maximum théorique d'une FAT16 est 4GiB

Quel choix pour une partition de 4 Gib ? FAT16.
Que dire de la taille des clusters ? Si on a des petits fichiers on perd bcp de place.

Exercice

J'ai une partition de 4Gib et je voudrais la formater en FAT32 avec des blocs de 4Kib :
2^{28}*4Kib(2^{12}) = 2^{40} = 1Tib➔ ok
Quel est la taille réel : taille partition/taille des blocs = 4Gib/4Kib =2^{20} blocs nécessaire.
J'ai donc 2^{20} entrées.
La taille de la table = 2^{20}*4bytes(car FAT32) = 4Mib

Si on a une grosse table, au démarrage ➔ + lent.

Index FAT - quelle taille table ?

Sachant que :

- Nombre entrées FAT = taille partition / taille clusters
- Taille table FAT16 = nombre entrées FAT * 2 bytes
- Taille table FAT32 = nombre entrées FAT * 4 bytes
- La taille maximum d'une table FAT16 est 128Kib
- La taille maximum d'une table FAT32 est 1Gib

Index FAT – taille / Exercices

La taille de la table FAT (index) pour une partition de 500Gib formatée en FAT32 avec des clusters de taille 4Kib est 500 Mib !
Avec des clusters de 16Kib on réduit à 125Mib

Une partition de 500 Gib partitionnée avec des clusters de 4Kib a besoin de 500Gib/4Kib entrées de table pour décrire chacun de ses clusters s'agissant d'une FAT 32, les entrées de table font 4 bytes donc la taille de la table d'index peut être calculée par :
500Gib/4Kib x 4 bytes = 500Mib

Une table FAT32 peut devenir très grosse

SYS

Critique

Une table FAT32 peut devenir très grosse et est chargée en RAM au démarrage du système ! !
-> un grand volume FAT ralentit le démarrage du système

Valeurs défaut

taille partition	type de FAT
< 2 Gib	FAT16
2 Gib - 32 Gib	FAT32
> 32 Gib	NTFS

Secteur de boot

Le Boot sector et Bios Parameter Block métadonnées du système de fichiers :

- (3b) instruction de saut au chargeur du système (plus loin)
- (2b) taille d'un secteur (512)
- (1b) nombre de secteurs par cluster
- (2b) taille zone réservée (1-32)
- (1b) nombre de FAT (2)
- (2b) nombre d'entrées de 32 bytes dans le répertoire racine en FAT16 (512, 0 pour FAT32 et 4 bytes ailleurs)
- (2b) nombre de secteurs dans le volume FAT (4b pour FAT32)
- Code du chargeur du système
- Nombre d'entrées de la FAT

Nombre d'entrées répertoire racine

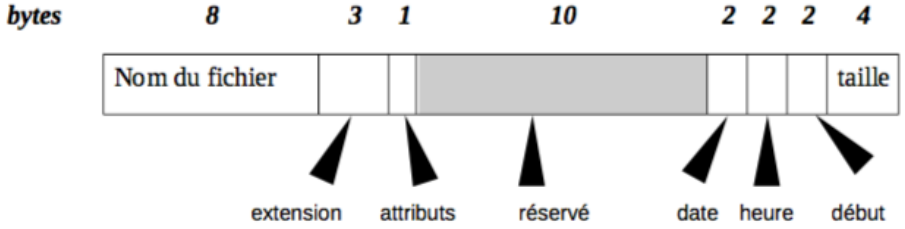
- Pour une FAT 16 le nombre d'entrées du répertoire racine est limité : 512 - 1 (utilisé pour le Volume-id)
- VFAT introduit la possibilité de noms longs en utilisant plusieurs entrées de répertoire ! Avant, les noms de fichier étaient limités à 2 caractères ;

Répertoire

- Fichier particulier
- Contient les métadonnées des fichiers
- Descripteurs de 32 bytes (un par fichier/sous-répertoire)
- Fichier d'enregistrements

Entrée de répertoire

Tout fichier/répertoire est décrit dans son répertoire parent (mais est-ce bien vrai ?)



Il n'y a que 1bytes pour indiquer les droits
Les 32 bytes d'une entrée de répertoire en FAT16
Si je stocke la taille sur 4bytes, je peux aller jusque 2^{32} donc 4Gib de taille max.

SYS

Attributs : un par bit

- ARCHIVE
- READ_ONLY
- HIDDEN
- SYSTEM
- DIRECTORY
- VOLUME_ID
- 2 bits à 0 (inutilisés)

Nom et extension : 11 bytes (8 modifiable + 3 extension) nom DOS
Premier byte du nom : valeurs particulières

- 0x00 : première entrée libre
- 0xF5 : fichier supprimé
- 0x05 : remplace la valeur 0xF5 car maintenant F est caractère.

Noms longs depuis vfat.

Long nom

- En cas de nom long on occupe plusieurs entrées répertoire
- Chaque nouvelle entrée permet d'étendre le nom de 11 caractères
- L'entrée est marquée grâce à l'attribut ATTR_LONG_NAME

ATTR_LONG_NAME est ignoré par DOS ce choix assurait la compatibilité ascendante pour DOS.

L'attribut ATTR_LONG_NAME correspond à une combinaison inconnue par DOS et donc ignorée
ATTR_LONG_NAME = ATTR_LONG_NAME = READ_ONLY + HIDDEN + SYSTEM + VOLUME_ID

- Ceci permet de résoudre la compatibilité ascendante pour DOS
- DOS n'est pas perturbé par les entrées de répertoire qui correspondent à des noms longs et ne verra que la première partie du nom des fichiers
- Surprise ! Pour ceux qui croyaient faire des backups et ne faisaient qu'écraser des fichiers
- Surprise ! Pour ceux qui travaillaient avec une FAT16 sans classer leurs fichiers en répertoires

Ne comprend que les 8premiers bits.

SYS

Heure-date en FAT16 :

Dernier accès en écriture : heure et date (2+2 bytes)

- Le choix est porté sur une représentation structurée (année, mois, ...)
- Si on représentait la date par un nombre de secondes sur 32 bits on pourrait représenter sur 4 bytes 9 années de plus avec une précision supérieure.

2bytes HMS
H 24 ➔ 5bits
M 60 ➔ 6bits
S 30➔ ne reste que 5 bits donc on représente 1 seconde sur 2.

2bytes JMA
J 31 ➔5bits
M 12 ➔ 4bits (2^4)
A 128 ➔ 7bits 2^7 = 128.

Donc on a 128 ans avec 2 secondes de précision.

UNIX :
(2^{32} sec)/(1 jour =86400 sec) / 365 = 136 ans avec précision de 1 seconde(1980+136 = 2116)
FAT32 : Du coup on a repris 1byte, 200 valeurs ➔ précision 1/100secondes.

Heure-date en FAT32

Un byte dans la partie réservée permet d'obtenir une précision plus fine :
Une valeur comprise dans [0-199] centièmes de secondes (= 2 secondes)
Gère également la date du dernier accès (lecture ou écriture)

Numéro du premier bloc :
2 bytes en FAT16 le premier cluster du fichier
Servira à localiser tous les clusters suivants pour un fichier vide ce premier cluster vaut 0
FAT32 utilise 4 bytes pour ce champ (deux bytes supplémentaires dans la partie réservée)
Taille du fichier : 4 bytes (taille logique) -> le plus grand fichier sur une partition FAT : 4Gib
Cette limitation est encore présente en FAT32.

Racine

Répertoire racine

Particularités du répertoire racine :
Nom ? premier cluster ? longueur ?
Sa position est calculable
Son contenu, comme celui des autres répertoires, est une suite de descripteurs de fichiers/répertoires de 32 bytes
Sa longueur : une métadonnée du système de fichiers

En FAT12-16

Le répertoire racine a un emplacement fixe calculable. Sa taille est limitée !
Son allocation contiguë

En FAT32

Racine = chaîne de clusters, le premier est renseigné dans la zone boot (souvent le 2)

SYS

Localiser

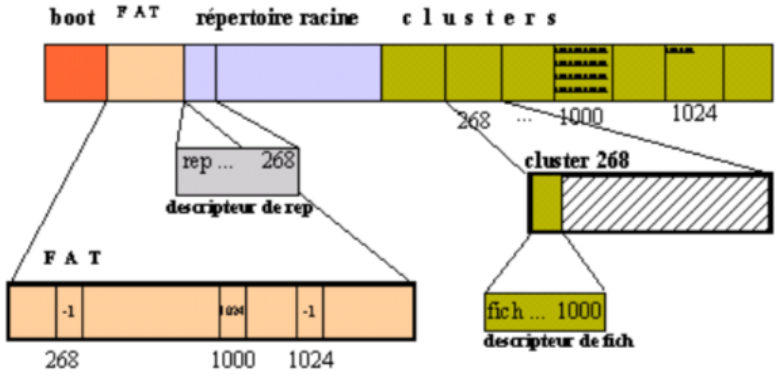
Les clusters sont numérotés depuis le n°2

- Pour une FAT16 le cluster 2 suit le répertoire racine
- Pour une FAT32 le cluster 2 suit les tables FAT

La position d'un cluster est calculée sur base des métadonnées du système de fichiers

Exercice

Soit une partition FAT16 avec le fichier et le répertoire : /rep/fich Le fichier contient 1030 caractères 'a'
La taille des clusters est 1Kib (1024 bytes)
Le répertoire rep est décrit dans le cluster 268 Le fichier fich occupe les clusters 1000 et 1024. Dessinons la structure de ce système de fichiers.



Fich est définit dans 2 cluster.
Le bloc 1024 n'est pas rempli ➔ fragmentation interne.

± Résumé

Répertoire : c'est un tableau de structure et chaque entrée de ce tableau c'est la description d'un fichier.
Répertoire = tableau et chaque tableau fais 32bytes

Où est la taille ? Dans le descripteur du fichier void qui se trouve dans le répertoire parent.
1byte pour indiquer les attributs : READONLY etc
EN FAT 32 pas de descripteur de fichier car dis directement ou commence les fichiers.

Comment savoir ou se termine le fichier ?
C'est dans le descripteur de fichier, il y a un -1 mais indique juste la taille/fin du BLOC.
-1 car le fichier rentre totalement dans le byte.

Clusters 1024bytes
/→fich 1 15 000 bytes
↳→ rep
Besoin de ± 15blocs car 15000/1024 pour cluster et pour rep 1blocs car 3x32bytes

Descripteur de fichier de fich est dans racine répertoire car apd de racine on a 2 répertoire tandis que dans le slides on a /→ rep→fich ➔ /rep/fich

TDFO : mémorise la manière d'accéder directement à la métadonnées des fichiers ouvert, ne doit pas faire tout le travail de open à chaque fois.

open - localiser le fichier

Open - Comment fait le système pour lire les données du fichier /rep/fich ?

- Localiser /
- Lire les enregistrements successifs du fichier jusqu'au descripteur de "rep" et obtenir sa position
- Lire les enregistrements successifs du fichier rep jusqu'au descripteur de "fich" (en parcourant les clusters chainés)
- Obtenir la position de fich ajouter l'entrée à la TDFO read

Lire les données du fichier fich (en parcourant les clusters chaînés)

Questions

Que doit faire le système quand la taille d'un fichier augmente et que le dernier cluster du fichier est plein ? = Que doit faire le système de fichier lorsque l'on veut ajouter un bloc à la fin ? On fait un chainage vers un bloc libre.

Que doit faire le système pour créer un nouveau fichier vide ? On n'alloue pas de bloc, mais on doit faire une nouvelle entrée dans le descripteur de fichier. Qui fera descripteur : f 0 0

- les clusters d'une FAT ont tous la même taille [**V**-F] et dans ext aussi.
- en FAT, il y a perte d'espace disque par fragmentation interne [**V**-F]
- un répertoire quelconque en FAT tient sur maximum un cluster [V-**F**]
- la File Allocation Table sert à trouver le premier cluster d'un fichier[V-**F**] c'est dans le parent.

Est-ce possible et si oui comment, de saturer un système de fichiers FAT16 de 2Gib :

- Avec +/- 2 exp 30 fichiers de 1 byte ? Non car chaque fichier prendra 1 blocs.
- Avec +/- 30 fichiers de 1 byte ? Oui avec des noms TRES long

Intégrité

Incohérence

Des états incohérents d'un système de fichier peuvent survenir par exemple suite à une panne de courant. Notamment si celle-ci survient pendant un état instable du système. En cours de modifications, par exemple lors d'ajout ou suppression d'un cluster d'un fichier ou bien si on retire la clé USB alors qu'elle est occupée.

L'ajout d'un cluster demande deux mises à jour :

- Le cluster libre pointe vers le cluster suivant du fichier ou la fin du fichier
- Le cluster qui précède le nouveau pointe vers le nouveau cluster

(étant donné qu'on ajoute un cluster, celui qui se situe avant doit pointer le nvx et le nvx doit pointer le prochain cluster)

Même chose lors de la suppression d'un cluster :

- Le cluster qui précède celui à supprimer pointe vers le cluster suivant
- Le cluster à supprimer est marqué libre

(le cluster qui le pointait va pointer celui qui était pointer par celui à supprimer.)

SYS

Dans les deux cas, si une panne de courant survient entre les deux mises à jour, la FAT sur disque reste dans un état incohérent.
Exemple
On aura peut-être 2 fois une même redirection vers un même cluster ou même un cluster qui nous dirige vers un bloc libre, etc …

Comment se rendre compte qu'il y ai eu une coupure de courant. Car il y a un flag mis à 1 lorsque l'ordinateur est allumé donc lors d'une coupure il sera toujours à 1 sinon il aurait été mis à 0.

La redondance permet de "réparer" l'incohérence.

Mais lequel est incohérent ?

La commande chkdsk utilise cette redondance :

1. Parcourt la FAT à la recherche des clusters chaînés : ceux qui appartiennent à un fichier
2. Cherche les clusters libres : ceux qui ont un 0 en FAT

En théorie un cluster "normal" doit soit être libre soit être chaîné.

La comparaison entre ces deux résultats permet d'isoler des "chaînes de clusters perdues"
Ces dernières sont récupérées sous forme de fichiers nommés FILE0000.CHK, FILE0001.CHK dans le répertoire racine.

Cluster défectueux ?

Un secteur peut devenir défectueux aussi dans la FAT. Du coup on perd tout le chainage donc GROS problème c'est pour ça qu'il y a une copie FAT ou bien même 2* la FAT donc 2 mises à jour sur disque.

Cluster défectueux dans un fichier -> perte d'une partie de données
Cluster défectueux dans la FAT -> le système de fichiers risque d'être fortement compromis
➔ la copie de la FAT peut être maintenue pour garantir une plus grande fiabilité
si le cluster défectueux est au niveau d'un fichier utilisateur c'est moins grave.

Conclusion

(++)Est portable, connu, ouvert, ont pu facilement créer des driver.
(-)Taille de fichier limitée à 4Gib
(-)Pas de propriétaire, tout le monde est propriétaire de tout.
(-)Pas grande partitions car trop grande table (grandes partitions et clusters petit) donc démarrage TRES lent. (jusque 16TiB)
(-)Comme tous les système souffre de fragmentation fichier à cause par bloc. Celui qui souffre moins = EXT car avant d'allouer réfléchit + à ou le mettre tandis que les autres prennent le 1er bloc libre.

Table FAT chargée en RAM et mise à jour en même temps en RAM et sur le disque

FAT16/32

(+) FAT32 utilise l'espace plus efficacement que FAT16
(-) FAT32 + lent au chargement car grosse table FAT

SYS

Question

L'appel système open fait des lectures sur le disque [V-F] on doit lire racine puis tous les sous-répertoires jusqu'au parent, c'est lui qui dit les droits, … (fd se trouve dans le répertoire parent)
En fait, que fait l'appel système open pour un fichier d'une FAT ?

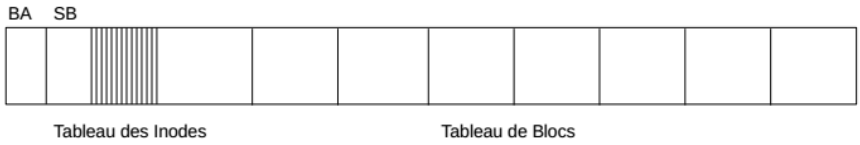
EXT

Un mini-disque (équivalent partition) formaté en ext est composé de 4 zones :

- Boot area: participle au boot.
- Superbloc
- Tableau d'inodes : contient taille, structure. Est une métadonnée du système. inode 2 c'est le slash, la racine (index commence à 0). Répertoire parent contient l'inode de chaque fichier et leur nom. (Le répertoire au-dessus)
- Tableau de blocs : les donnée fichier & répertoire.

La commande linux mkfs permet de formater une partition. mkfs.ext2 formater en ext2

Structure interne



Tableaux de blocs

- Un bloc est un ensemble de bytes.
- Un bloc contient uniquement les données du fichier.
- La taille du bloc est exprimée en nombre de secteurs(n*512 bytes).
- La taille et le nombre de blocs sont fixés à la création du F.S.
- /home est un minidisque sur Linux2, les blocs ont une taille de 8 secteurs (4Kib)

Tableau d'inodes

Un inode est un ensemble structuré. L'inode contient uniquement les métadonnées du fichier.

- A chaque inode correspond un et un seul fichier. (Quand créé lien hardware on donne un nouveaux nom à un même fichier, donc 1 inode 1 fichiers mais plusieurs noms différents. Lien software est un petit fichier qui a aussi sont inode et contient le chemin du fichier.)
- Chaque inode est repéré par son numéro dans le tableau d'inode.
- L'inode contient toutes les informations concernant son fichier sauf les données et le nom.
- Quelles sont les informations que vous vous attendez à voir dans un inode ?

SYS

Contenu d'inode

- Le numéro du propriétaire (uid,gid).
- Les droits d'accès (rwxrwxrwx). sur 24bits, masque de 9bits mis ou non à 1, il y aura + que 9 bits car indication si fichier régulier, type(normalisé, …)
- Des dates (dernier accès, dernière modif des données, de l'inode, effacement). en système UNIX
- Le type (fichier normal, spécial, répertoire, lien, ...).
- Le nombre de liens.
- La taille du fichier. + le nbre de bloc réellement utilisé.
- La liste continue des blocs de ce fichier. on va pouvoir trouvé le 1er bloc, le 2ème, le 3ème, ...

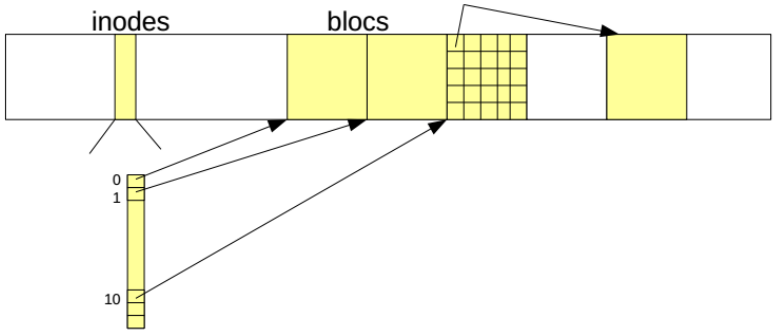
Liste des blocs

13 (15 en ext2) numéros de blocs sur 4 bytes (= des pointeurs vers des blocs)

- Les 10 premiers n° de blocs : Dans l'ordre, je trouve les données de fichiers à la suite des un des autre.
- Les bloc 10, 11 et 12 : sont différent, pointe vers un n° de bloc nommé bloc (pointeur) d'indirection, contient des n° de bloc supplémentaire qui pointe quelque part dans le tableau de bloc.
- Bloc 10 : pointeur d'idirection simple, on peut couvrir 40Kib avec les 10 premier + maintenant c'est 3 dernier blocs donc comme j'ai des blocs de 4 Kib je peux déterminer le nombre de pointeur que j'ai en +, ici il y a 1024 pointeur dans un blocs. Donc 40Kib + 1024*4Kib
- Bloc 11 : pointeur d'indirection double : j'ai toujours un pointeur vers le tableau de bloc on a de nouveaux 1024 pointeur mais chacun de ces 1024 pointe vers un autre. Donc 40Kib + 1024^2*4Kib
- Bloc 12 : pointeur d'indirection triple : donc il pointe 3 fois avant d'arriver aux donnée. 40Kib+1024^3*4Kib

(Bloc 12 : est le numéro du bloc contenant des numéros de blocs contenant des numéros de blocs contenant les données du fichier)

Structure interne



Combien de blocs peut-on avoir pour un fichier si les blocs ont une taille de 2 unités ?
Les blocs on taille de 1024 donc 2 secteurs

nb de bloc : 10+256+(256^2)+(256^3)
Taille max : 10Kib + 1024Kib+1024^2Kib+1024^3Kib

bloc de 1024bytes : 10 +1024+1024^2+1024^3
Taille max : 40Kib+4096Kib+4*1024^2Kib + 4*(1024^3)

SYS

Exemple listes des blocs

- Dessinez comment est mémorisé un fichier de 10 nombres ? de 2000 nombres ? de 200000 nombres ?

  200000nb = 800 000bytes ➔ 782blocs.
  10 pointeurs ➔ 772 encore à caser
  pointeur 10 (256 bloc pris) ➔ 516 encore à caser
  pointeur 11 (utilise 2 pointeurs, 256 et le 2ème pointeur 256 puis on utilise un autre pointeur encore. Donc on part du n°11, là on utilise 2 pointeur, sur 1 pointeur on pointe sur un autre qui nous fais les 2*256, puis on retourne en arrière et la sur le 2ème pointeur on ne prend que 4 blocs)
- *Quelle est la taille d'un fichier contenant les 26 lettres de l'alphabet ? Quelle est son occupation sur le disque ?*
- *Quelle est la taille maximale d'un fichier si les blocs sont de 4 unités et les numéros de blocs sur 4 bytes ?*
- Comment est représenté un fichier de 160000 bytes avec une taille de bloc de 512 bytes ? Si taille de bloc de 512 ==> 128 pointeurs dans chaque blocs.
  Les 10 premier 10 + 128 pour le n°11 + 128 + 47 = 313

(yes blabla va répéter blabla à l'infini, on peut aussi rediriger vers fichier, on aura direct un fichier énorme.
stat fichier ➔blocks = avec indirection.)

Vérif 145/167 italique ➔ pas fais

- Que vous faut-il pour accéder à un fichier : son n° de bloc ou son n° d'inode ? *Son n° d'inode car ce dernier nous donne le n° de bloc.*
- Connaissant le numéro d'inode d'un fichier, quelles informations sur ce fichier connaissez-vous ? Précisez où vous trouvez ces informations. *On connaît les inodes de tout les fichiers qu'il contient ainsi que le nom de ces fichiers, on voit ça dans la table d'inodes.*
- Connaissant le numéro d'inode d'un fichier, connaissez-vous son nom ? Le nom de son répertoire ? *Oui*

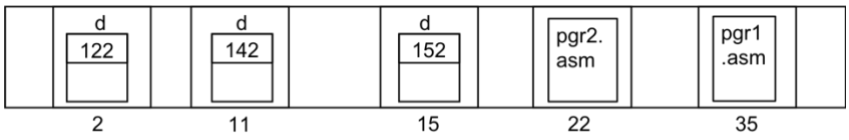
Répertoires

Un répertoire est un fichier. (inodes, informations, blocs, ...), est structuré en enregistrements, il y en a un par fichier que contient ce répertoire.
Chaque enregistrement contient 2 champs : Le nom du fichier et son numéro d'inode.
Le point de départ : le root correspond au numéro d'inode 2. (1 gère les blocs défectueux).

Au moment de la création d'un fichier ou répertoire, le type de celui-ci est dans la table d'inode. Cad que dans la table d'inode on sait si c'est un répertoire ou un fichier.

Exercice

Dessinez comment est mémorisé le répertoire /home/g12345 contenant pgr1.asm et pgr2.asm.



Ici, n° inode choisit au hasard.

/

.	2
..	2
home	11

Bloc 122

home

.	11
..	2
g12345	15

Bloc 142

g12345

.	15
..	11
pgr1.asm	35
pgr2.asm	22

Bloc 152

SYS

Super bloc

Contient des informations permettant de savoir

- où commencent et où se terminent les tableaux d'inodes et de blocs.
- le nombre d'inodes et de blocs sur ce F.S.
- la taille d'un bloc.
- si le F.S. a été correctement démonté.
- où sont les blocs libres.

Liens

Permet plusieurs accès au même fichier.
Un fichier porte plusieurs noms.
Deux types différents : liens (hard) et liens soft (ext2).

Liens hard (ln brol lien)

- Manière d'avoir 2 fichiers/répertoire avec le même inode et un nom différents.
- Si on supprime un des 2, il va dans la table d'inode et supprime ce qui est dans la table d'inode. MAIS comme on a fait un lien, dans la table d'inode au même inode, on a un compteur de lien donc on sait qu'il y a 2 fichier pour cette inode, si on veut supprimer il va juste décrémenter, s'il arrive a 0 le S.E. peut détruire.
- Lorsque je créé un lien, le propriétaire de lien est celui qui est proprio du fichier duquel on a fait un lien.

Liens soft (ln-s brol lien)

Dans ext2 on a créé autre notion de lien, le lien software.

- Ici on ne fais pas référence à l'inode mais au chemin.
- Donc le fichier lien à son propre inode, les données de ce fichier lien est le texte qu'on a rentré dans la commande, ici c'est brol.
- Le lien aura le type lien soft.
- Comment le système sait que lorsque l'on fais un nano brol ou un nano lien c'est pareil, il fait un déférencement, il le sait. Si je fais rm lien il va juste supprimer le lien. Et si je fais un rm brol, il supprime brol et laisse le lien mais lorsque l'on essaiera de le modifier, il recréera brol. (lien irrésolu : lorsque l'on a supprimer le fichier mais qu'on a gardé le liens)

Compteur d'inode : pour les répertoire, il compte aussi le nbre de sous répertoire car dans un sous répertoire il le référence.

S3

MULTIPROGRAMMATION

Définition

- MULTIPROGRAMMATION : plusieurs programmes sont chargés en mémoire et s'exécutent de manière entrelacée.
- TIME SLICING : ajout de contrainte sur le temps maximum pendant lequel un processus peut utiliser le CPU sans interruption.
- POLLING aller chercher une information de manière récurrente. CAD : T'as fini ?T'as fini ?T'as fini ?T'as fini ?T'as fini ? jusqu'a qu'il dise oui.
- MONOPROGRAMMATION : le fait que dans notre système il ne tourne uniquement qu'un programme à la fois. Dans la RAM, qu'un programme chargé en mémoire.

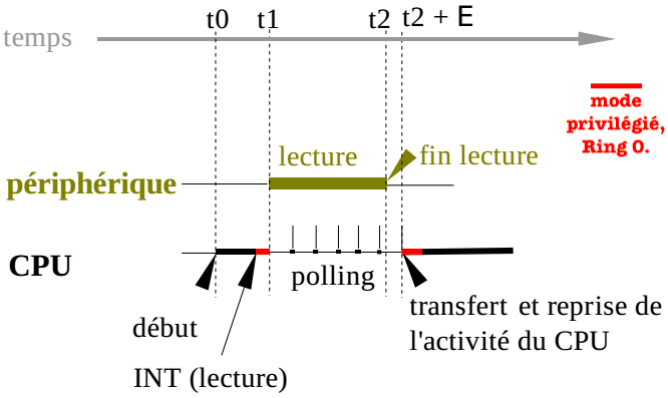
En multiprogrammation chaque appel système se termine en rendant la main à l'ordonnanceur ou ret.
Multiprogrammation, né pour rentabilisé les proco de l'époque.

Monoprogrammation

En monoprogrammation, on fait du polling pour savoir quand le programmes est fini.

Activité CPU-périphérique

Ici on demande à lire sur un périphérique.
Le CPU va exécuter cela, donc il fait la lecture sur le périphérique et fais un polling pour savoir quand la lecture est finie.
On utilise un appel système, donc ça passe en mode privilégie, le ring 0.



➔ utilisation inefficace du CPU car lors de la lecture il ne fais que du polling et ne peut rien faire d'autre.

Processeur canal & multiprogrammation

Processeur canal : va s'occuper du polling et du transfert en RAM. Pendant ce temps-là le cpu va pouvoir faire autre chose.
C'est l'ancêtre de DMA(direct memory access).

Il y a une interruptions (appels système provoqué par le matériel)du canal qui signifie la fin de la lecture. Comme ça le cpu pourra être VRAIMENT tranquille.

Processeur canal

Le processeur Canal est relié au bus et accède à la RAM.
Le canal interfère avec le fonctionnement du CPU.
Une synchronisation CPU/Canal est nécessaire (régler les accès à la RAM via les bus)

SYS

Multiprogrammation

Le CPU ne gère plus polling et transfert de données : il attend du coup on peut charger un 2ème programme en RAM.

point fort - CPU bien rentabilisé
point faible - canal non rentabilisé (processus de calcul)

Un programme qui demande une lecture, est bloqué au profit d'un autre qui récupère le CPU.

Processus

Processus = programme en mémoire.
Un processus s'exécute en mode normal et en mode privilégié lorsqu'on exécute du code système pour son compte.

Problèmes d'avoir plusieurs processus en mémoire

- mémorisation de l'état des registres du processeur (RIP, RAX, ...) (table des process)
- protection d'accès à la mémoire (segmentation sur intel)
- attribution du CPU : à qui le tour ?
  - ➔ ordonnanceur
- partage de ressources
  - ➔ problématique des interblocages

Pour chacun des processus en mémoire :

- une valeur de RIP
- une valeur de RSP pour sa pile
- un état de registres du CPU (RAX, RBX, ...)
- un mode de fonctionnement du CPU (privilégié ou non)
- un état (élu, prêt, bloqué)

Contexte à mémoriser dans la table des processus. La table des processus contient l'état, la prochaine valeur de RIP, …

L'état d'un processus

- Bloqué : dans l'attente d'une fin de lecture.
- Prêt : pourrait s'exécuter si il avait le processeur.
- Élu : s'exécute (un seul).

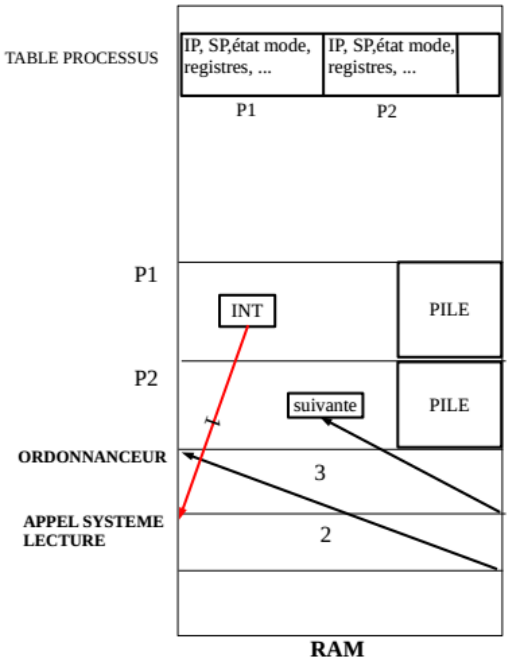
Ordonnancement

L'ordonnanceur choisit un processus prête dans table des process et lui attribue le CPU.

SYS

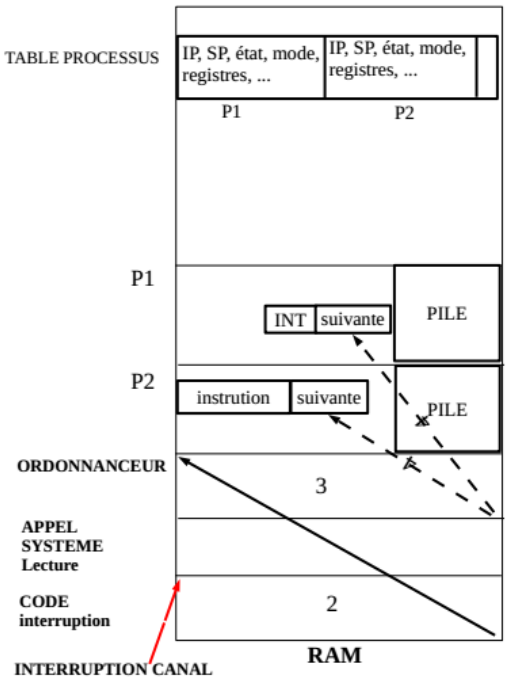
Lecture en multiprogrammation

Demande de lecture



- P1 demande une lecture (SYSCALL)
- Appel Système
  - état P1 <– bloqué
  - sauvegarde le contexte de P1 dans la table
  - commande périphérique et canal
  - RIP < adresse de l'ordonnanceur
- Ordonnanceur
  - état P2 dans la table <– élu (seul prêt)
  - charge les registres et le mode de P2 (depuis la table)
  - RIP <- RIP de P2 depuis la table
- P2 a la main, il continue à s'exécuter

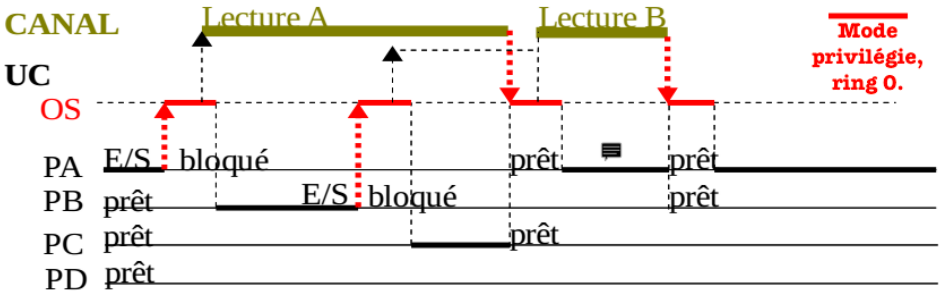
Fin de lecture



- Le CANAL interrompt le CPU (P2)
  - pile<-RIP et mode, mode<- privilégié, RIP<-adresse gestion d'interruption
- la routine de gestion de la fin de lecture
  - état P2 <– prêt
  - sauvegarde le contexte de P2 dans la table
  - état P1 dans la table <– prêt car fin de lecture
  - RIP <- adresse de l'ordonnanceur
- Ordonnanceur
  - état P1 ou P2 dans la table <– élu (il choisis l'un des 2 au hasard.)
  - charge les registres et le mode de l'élu
  - RIP <- RIP de l'élu depuis la table
- l'élu a la main

Dis au cpu : j'ai tout transféré, on y go.
juste avant d'exécuter suivante il va se demander s'il n'y a pas une interruption si c'est le cas elle ne sera pas exécuter, du coup le cpu va basculer pour aller exécuter le code de l'interruption canal.
Va sauvegarder le contexte, p2 devient prêt, plu interrupt.
Comme on aura remis la main à l'ordonnanceur, il va reparcourir la table et devra choisir entre les programmes prêt.

SYS



Basculement OS pour après utiliser canal. **en gras : moment où on exécute du code.** Donc on commence à exécuter le code du processus A. **la fin de lecture A interrompt C.** un bon ordonnanceur doit être équitable, par expl ici D n'a jamais eu la main.

Un processus rend la main lorsque il fait une lecture.
les processus d'E/S rendent vite la main (sont dit : gentils)
les processus de calcul ne rendent jamais la main, monopolise le CPU (sont dit : méchant)

Time Slicing

time slice = tranche de temps. Rentabilise le canal.

Avant un processus pouvait arder le cpu pendant très longtemps, ici on a mis un temps maximum.

Un processus garde le CPU pendant une tranche de temps de durée maximum t après on le lui retire au profit d'un autre, il devitn alors prêt

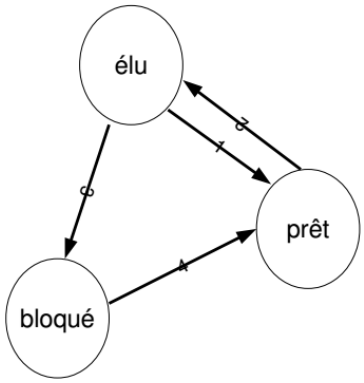
- permet un contrôle plus fin de l'enchaînement
- permet de rentabiliser le Processeur Canal

On utilise une interruption horloge (mécanisme de préemption)

- l'ordonnanceur est appelé par l'interruption horloge
- permettra une exploitation intéractive (time sharing) car minimise les temps de réponse

inconvénients : gérer les changements de contexte prend du temps CPU

Transitions d'état d'un processus



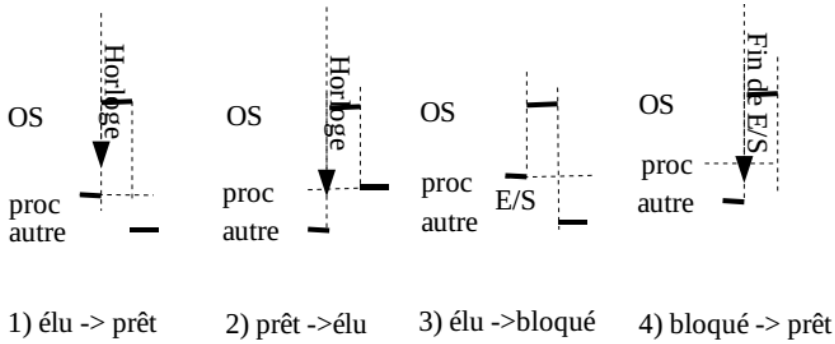
- Élu → prêt : à cause du time slicing(interruption horloge) ou interruption canal. Lorsqu'un process s'exécute depuis trop longtemps.
- Prêt → élu : l'ordonnanceur a choisi ce process. Lorsqu'un autre process est bloqué ou s'exécute depuis trop longtemps.
- Bloqué → prêt : l'interruption canal. Lorsqu'un process qui attendait une ressource la reçoit.
- Élu → bloqué : process élu demande une ressource ou communique avec périphérique.

On en peut pas :

- Prêt → bloqué non car il faut être élu pour être bloqué.
- Bloqué → élu : On doit obligatoirement être prêt pour être élu.

Un processus passe d'un état à un autre lorsqu'il reçoit un signal.

SYS



1. time slicing car l'horloge interrompt proc, il était élu et devient prêt.
2. proc n'avait pas la main, il l'a reçois. car horloge interrompt autre.
3. il y a une entrée
4. interruption canal pour signaler fin de lecture. Les 2 process sont prêt.

Quantum de temps = le temps après lequel l'horloge gère les changement de contexte.
Si quantum court : le cpu ne va faire que de la gestion donc cpu inéficace. mais meilleur temps de réponse, perte d'efficacité.

Préemption

- système coopératif : un processus rend la main spontanément (premiers windows) (chaque programme devait coopéré et dire à un moment "je rends la main")
- système non préemptif : un processus se bloque (E/S, ...) ou rend la main spontanément
- système préemptif : un processus se bloque (E/S, ...) ou rend la main spontanément, ou son temps est écoulé (Windows NT, 2K, XP, linux depuis noyau 2.6,...)

Time slicing

Un ordonnanceur peut être non préemptif même en présence d'interruptions horloge
Un ordonnanceur préemptif ne peut pas exister sans interruptions horloge

Quand a lieu l'ordonnancement ?

- demande d'E/S
- fin d'E/S
- interruption horloge (préemption)
- nouveau processus
- fin d'un processus

à la fin de chaque interruption / appel système (en simplifiant)

Questions

- si P est l'élu, alors RIP pointe vers une des instructions de P [V-F]
- la multiprogrammation rentabilise le CPU [V-F]
- combien de processus en mémoire? tant qu'il y a de la place on peut en mettre
- combien de prêts en même temps? minimum 1
- combien de bloqués en même temps? n-1, tous sauf 1 car il y en a toujours 1 qui tourne.

le cpu ne s'arrête JAMAIS

- un processus peut passer de l'état prêt à bloqué [V-F]
- un processus peut passer de l'état bloqué à élu [V-F]
- quels avantage(s)/inconvénient(s) a le Time Slicing par rapport à la Multiprogrammation ? Time Slicing rentabilise le canal mais pert de l'efficacité avec de la gestion tandis que la multiprogrammation rentabilise le cpu mais pas le canal, si il y a un process de calcul, il va monopolisé le cpu.
- donnez des cas où l'état d'un processus bascule
    - de élu -> prêt : interruption horloge, interruption canal
    - de élu -> bloqué : appel système

ORDONNANCEMENT

L'ordonnanceur (scheduler) doit élire un nouveau processus qui est dans la table des process avec un état prêt.

L'algorithme du sheduler (scheduling algorithme) doit être conçu pour prendre des décisions en fonction de différents points de vue :

- équité (votre voisin compile 10 fois plus vite ...)
- temps de réponse (le calcul de pi vous empêche de taper)
- temps d'exécution (chaque E/S vous pénalise) depuis le moment où on démarre un programme et qu'on en sort. Entre le début et la fin.
- rendement/efficacité (90% du processeur est pour le scheduler) time slicing si on interrompt trop souvent le cpu ne va faire que de la gestion donc cpu inefficace.
- équilibre (toutes les parties du système sont utilisées)

L'ordonnanceur a un rôle très important dans un S.E. multitâche.
Privilégier un point de vue se fait au détriment d'un autre.
Pour les systèmes interactifs on se souciera plus d'améliorer les temps de réponse

Compromis à trouver en tâchant que

- chaque processus ait sa part
- le processeur soit utilisé à 100%
- le temps de réponse (en interactif) soit minimum
- le nombre de travaux achevés/unité de temps soit maximum

Il n'existe pas de solution idéale pour toutes les situations : nous allons étudier certaines techniques.

SYS

Tourniquet

Chaque processus se voit assigner un intervalle de temps, appelé quantum, pendant lequel il est autorisé à s'exécuter. Place tous les processus sur le même niveau.
Un processus est interrompu soit :

- parce qu'il a terminé
- parce qu'il a besoin d'une E/S (élu -> bloqué)
- parce qu'il a épuisé son quantum de temps.

Le sheduleur élit alors le suivant de la liste etc
algorithme simple à écrire. Comment fixer le quantum ?

Si sheduler utilise 5ms pour échanger un processus(échange de contexte, mise à jour de tables, ...) et si 10 processus prêts en même temps :

quantum	temps max attente	efficacité CPU
5 ms	90 ms	50% = 5/10
100 ms	+/- 1 sec	95% = 100/105
1 sec	+/- 10 sec	99,5% = 1000/1005

Pour le quantum de 5ms, le temps max d'attente est de 95 ms car 5ms pour changer de processus + 5ms de quantum, le tout * 10 - 1 changement donc 95ms.

Temps max attente = cmb de temps il va falloir pour que je récupère la main.
Efficacité = temp utile(quantum)/temp total (util + tps pour échanger un processus.)
Pour quantum 100ms, on perd 5% en gestion(tâches d'administration).

Si le quantum est trop court, il y'a trop de changement, s'il est trop long, le temps de réponse aux requêtes est trop important. On le fixe alors aux alentours de 20-50ms.

facile à mettre en œuvre, peu couteux. Mais pas satisfaisant car on peut imaginer que dans un système on ait envie de donner priorité à certain process.

Priorité

Certains processus sont plus importants que d'autres donc on leur associe une priorité.
A chaque processus, on associe une priorité.
Le sheduler élit les processus par priorité décroissante ➔ risque de famine pour les processus de petite priorité
Priorité statique : on donne une priorité une fois pour toute au process.
Famine : exemple avec mail qui a un process pas prioritaire, il risque qu'à chaque fois un autre process prenne la main car il a + de priorité, mail ne prendra jamais la main.

Dynamique

Solution : priorité dynamique (-1 à chaque accès au processeur par exemple à la fin ils seront tous a 0. Pas très intelligent.)
Désavantage : on oublie sa priorité de départ
Idée suivante : on favorise les processus qui demandent beaucoup d'E/S en leur attribuant tout de suite le processeur.

Un processus qui demande beaucoup d' E/S doit être favorisé (car les E/S se font en // avec le processeur et interactivité)
priorité = quantum / temps du quantum utilisé
Idée : si un process n'a pas utilisé tout son quantum, on voudrait qu'il passe + vite. De faire en sorte que toutes les ressources du système tourne.

Exemple :

- un processus a un quantum de 100 ms et est bloqué par demande d'E/S après 5 ms : recevra une priorité de 100/5 = 20.
- Un processus utilise tout son quantum : recevra une priorité de 1.

SYS

Files multiples ou classes

- Processus qui utilisent beaucoup le CPU et peu les E/S ont une performance qui augmente quand le quantum est grand. (process calcul)
- Processus qui utilisent peu le CPU et beaucoup les E/S ont un temps de réponse meilleur quand le quantum est petit.

Il faudrait donner un grand quantum à un processus qui demande beaucoup de calcul et une petite priorité.
Il faudrait donner une priorité plus grande et un petit quantum à un processus qui demande beaucoup d'E/S.

Classe	Quantum	priorité	remarque
1	40 ms	5	pour un processus d'E/S
2	80 ms	4	...
3	160 ms	3	intermédiaire
4	320 ms	2	...
5	640 ms	1	pour un processus de calcul

Si plusieurs process dans la même classe on va les prendre à tour de rôle(tourniquet).
Quand le quantum du process diminue, il diminue de classe.
Si un process fais bcp bcp de calcul on va le basculer en classe 5 alors que si après il fait des E/S il sera désavantager.

Comment connaître la classe d'un processus ? En la fixant au départ.
Désavantage : tous vont se mettre en classe 1 !
En faisant évoluer : au départ : classe 1 et chaque fois que le processus consomme son quantum, il descend de classe.

On a, dans cet exemple, 5 tourniquets, un par classe. Tous les processus prêts de chacune des classes reçoivent un quantum. Un processus de classe inférieure est élu quand aucun processus d'une classe supérieure n'est prêt.

Un processus qui demande beaucoup d'E/S après un long calcul n'est pas à sa place
Pour remédier à cet inconvénient : à chaque E/S, le processus remonte en classe 1 mais, à cause de la nature humaine, cela ne fonctionne pas : il suffit de pousser régulièrement sur enter pour remonter en classe 1.

Exercices

- L'ordonnancement du tourniquet utilise le plus souvent un quantum fixe. Donnez un argument en faveur d'un petit quantum et un argument en faveur d'un grand quantum. Un petit quantum passe bcp de temps en gestion(perte d'efficacité) et maximise le temps de réponse. Un grand quantum permet a un process d'aller + loin dans ce qu'il a à faire mais monopolise le cpu.
- Soit un quantum de 92ms et un temps d'ordonnancement de 8ms calculez l'efficacité du processeur. 92% ➔ 92 utile sur 100(92+8)
- Donner une haute priorité à un processus favorise son temps de réponse [V-F]
- Avec un ordonnancement par tourniquet un processus s'interrompt pour une seule raison [V-F] Raison pour lesquels un process s'interrompt : son travail est terminé, fin du temp(time slicing)
- un ordonnanceur à priorités statiques améliore le temps de réponse de tous les processus [V-F] problème de famine

INTERBLOCAGE

Gestion ressources

Pour utiliser une ressource :

1. Sollicitation de la ressource
2. Utilisation de la ressource
3. Libération de la ressource

plusieurs processus → partage de la Mémoire, du CPU et aussi de ressources exemples de ressources :
mémoires de masse, imprimante, graveur, informations (enregistrement d'une BD), entrées de la table des processus...

Nous classons les ressources en

- Partageables / non partageables : si la ressources peut être utilisé par 2 processus en même temps→ partageable.
- Préemptibles / non préemptibles : si non préemptible, je ne peux pas faire quitter l'autre pour la prendre pour moi. Si la ressource peut être retirée à un processus sans dégâts → préemptible.

Ces propriétés sont liées à la nature des ressources, mais aussi à la manière dont ces ressources sont gérées.
Exemple : une imprimante est par sa nature non partageable, mais elle devient partageable par l'utilisation d'un spooler d'impression.

Une demande de ressource non partageable et non préemptible attribuée à un autre, bloque un processus jusque à "disponibilité de la ressource"

Interblocages

Dans certaines situations plusieurs processus se bloquent mutuellement indéfiniment ce que l'on nomme étreinte mortelle ou interblocage ou deadlock. ➔ aucune issues.

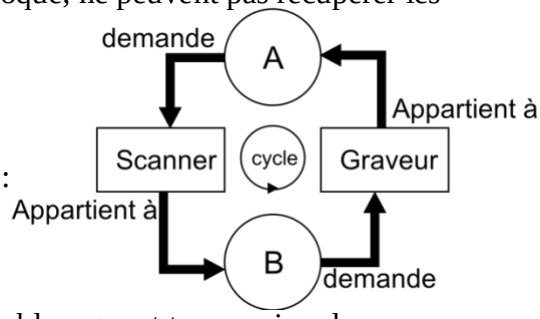
Soit l'exemple d'une utilisation simultanée de deux ressources non partageables et non préemptibles comme le scanneur et le graveur de CD, par deuxprocessus :

processus A :
obtenir scanneur
obtenir graveur
scanner et graver
rendre graveur
rendre scanneur

processus B :
obtenir graveur
obtenir scanneur
scanner et graver
rendre scanneur
rendre graveur

A obtient le scanneur : le garde tant qu'il ne l'a pas utilisé.
B obtient le graveur.
Maintenant ils sont interbloqué, ne peuvent pas récupérer les prochains.
Seul issue => la mort

On peut illustrer cela avec graphe cyclique :



Dans le cas d'utilisation d'une ressource unique le blocage est temporaire : le processus qui a la ressource se termine et cède la ressource à l'autre.
Une telle situation ne peut se présenter que dans le cas suivant :

- ressources non partageables
- ressources non préemptibles
- plusieurs de ces mêmes ressources sont nécessaires simultanément à plus d'un processus.

Il s'agit de conditions nécessaires non suffisantes

SYS

Solutions

- ignorer (politique de l'autruche)
- ou bien
  - détecter et reprendre (graphes)
  - éviter
  - prévenir

ces techniques ne sont pas sans coût

Ignorer

Exemple : Linux et Windows ignore. L'OS ne fais rien du tout, c'est l'administrateur qui doit s'en gérer.

Détecter - reprendre

détecter

- on représente l'état de l'allocation/demande par un
- graphe
- on détecte les cycles dans le graphe (c'est couteux en temps.)

reprendre

- suppression de processus
- préemption sur la ressource(retirer provisoirement une ressource à son propriétaire pour l'attribuer à un autre processus.)
- technique de Rollback (point de reprise, l'état du processus est inscrit dans un fichier pour pouvoir être restauré ultérieurement.)

Éviter

Attributions réfléchies → nécessite des informations sur les attributions futures
Basé sur la notion d'état sûr : état duquel il y a au moins une issue.

Prévenir

rendre partageable →cas du spooler et de son répertoire
allocation en bloc- non optimal
ordonner - ne fonctionne pas toujours (grand nombre de ressources)

Spooler (processus dédié aux impressions) : Le problème de l'accès à l'imprimante est résolu par une technique de prévention : le spool ; aucun processus n'accède directement à l'imprimante à l'exception du spooler d'impression

Les processus qui demandent des impressions écrivent dans des fichiers temporaires déposés dans un répertoire dédié au spooler. Ce dernier commande l'impression d'un fichier à la fois et évite ainsi le mélange des sorties. Le spooler n'imprimera un fichier que si il est complet (fermé)

Questions

Un interblocage ralentit le processeur[V-F]
Un interblocage peut ne pas être détecté par l'OS[V-F]
Une ressource préemptible et non partageable peut occasionner un interblocage[V-F]