Architecture et système

Les systèmes de fichiers

Christian NGUYEN

Département d'informatique
Université de Toulon

Plan

Un rôle fondamental

Le traitement de l'information requiert la notion de fichiers.

Ils sont gérés par l'OS et constituent un aspect très important de leur conception (*file system*).

Fonctions :

- enregistrer et retrouver des informations (de taille importante),
- de façon pérenne,
- avec prise en compte de l'accès simultané.

Plan

Le fichier du point de vue utilisateur

Une abstraction (un ensemble d'informations) manipulée par son nom.

Le nom est constitué :

- de caractères alphanumériques (majuscules et minuscules) et spéciaux,
- de deux parties séparée par un point (exemple : `prg.py`),

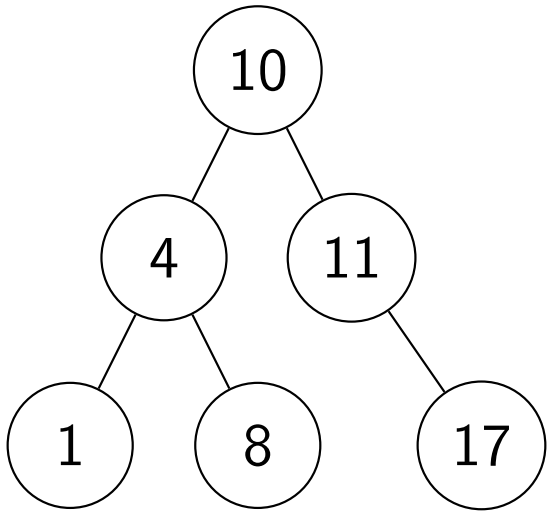
La deuxième partie est l'extension, qui donne une indication sur son contenu. Il peut y avoir plusieurs extensions (exemple : `archive.tar.gz`).

Il y a des extensions imposées par l'usage (exemple : `.c` permet de distinguer fichier à compiler de fichier à lier) et de simples conventions (exemple : `.txt`).

Structure des fichiers

On distingue trois structures différentes :

- simple suite d'octets : souple, pas d'aide, pas de restriction,
- suite d'enregistrements de taille fixe,
- arborescence d'enregistrements de taille variable (possédant chacun une clé).



Types de fichiers

On distingue les fichiers :

- ordinaires : informations des utilisateurs,
- répertoires (*directory*) : structure le système de fichiers,
- spéciaux : caractère (périphériques d'E/S, exemple le clavier) ou bloc (unités de stockage, exemple le disque dur).

Les fichiers ordinaires peuvent être textuels ou binaires :

- un fichier texte contient des lignes de caractères terminées par le caractère « fin de ligne », « retour chariot » ou les deux, l'interprétation de ces fichiers est plus simple, y compris entre deux processus,
- un fichier binaire a une structure interne qui lui est propre, son en-tête comporte souvent un nombre magique (*magic number*) qui indique son type.

Types de fichiers

Exemples de fichiers binaires.

```
$ hexdump -C a.out | more

00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  80 10 00 00 00 00 00 00  |..>.............|
```

ELF signifie Executable and Linkable Format

```
$ hexdump -C adresses.tar |more
00000000  61 64 72 65 73 73 65 73  2e 63 00 00 00 00 00 00  |adresses.c......|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000200  23 69 6e 63 6c 75 64 65  20 3c 73 74 64 69 6f 2e  |#include <stdio.|
00000210  68 3e 0a 23 69 6e 63 6c  75 64 65 20 3c 73 74 64  |h>.#include <std|
```

Accès aux fichiers

Le premier type d'accès fut séquentiel. La lecture se fait à partir du début du fichier.

L'apparition d'unités de stockage telles que les disques durs a autorisé la lecture dans le « désordre » (à accès aléatoire), le plus souvent à partir d'une clé.

Une opération de lecture (`read`) se fait par défaut à partir du début d'un fichier mais on peut spécifier la position de départ de la lecture (`seek`).

Attributs des fichiers

En plus du nom et des données associées, un fichier comporte d'autres informations, par exemple la date et l'heure de création, sa taille, etc.

La liste des attributs varie d'un système à l'autre mais on retrouve le plus souvent :

- le propriétaire du fichier,
- des attributs liés à la protection du fichier,
- des indicateurs [1] : lecture seule, fichier système, fichier temporaire, verrouillage, etc.
- différentes dates : création, modification, dernier accès,
- la taille courante.

1. bits ou petits champs qui caractérisent certaines propriétés

Opérations sur les fichiers

Tous les systèmes fournissent des méthodes pour stocker et rechercher des informations.

Principaux appels système :

- `create` : le fichier est créé sans données,
- `delete` : le fichier est « supprimé »,
- `open` : chargement en mémoire des propriétés du fichier,
- `close` : libération de la mémoire et (re)mise à disposition,
- `read` : lecture à partir de la position courante,
- `write` : écriture à partir de la position courante,
- `append` : ajoute des données à la fin du fichier,
- `seek` : modifie la position courante dans le fichier,
- `get/set attributes` : en particulier date de modification (`make`),
- `rename` : modification du nom du fichier.

Opérations sur les fichiers

Exemple de programme en Python.

```
"""
r, pour une ouverture en lecture (READ).
w, pour une ouverture en ecriture (WRITE).
a, pour une ouverture en mode ajout (APPEND).
b, pour une ouverture en mode binaire.
t, pour une ouverture en mode texte.
x, cree un nouveau fichier ouvert en ecriture.
"""
fd = open('data.txt', 'r')  # 'w'

fd.read()  # fd.write('hello')

fd.close()
```

Espace d'échange (`swap`)

[Wiki] Partie de la mémoire de masse utilisée par l'OS pour stocker des données qui se trouvent en RAM.

L'espace d'échange peut prendre la forme d'une partition dédiée (systèmes Unix) ou d'un simple fichier (`C:\pagefile.sys` sous Windows).

La RAM et l'espace d'échange constituent ensemble la mémoire virtuelle du système.

Sur certains systèmes, la partition d'échange est aussi utilisée pour la mise en hibernation, ou veille sur disque, du système.

Plan

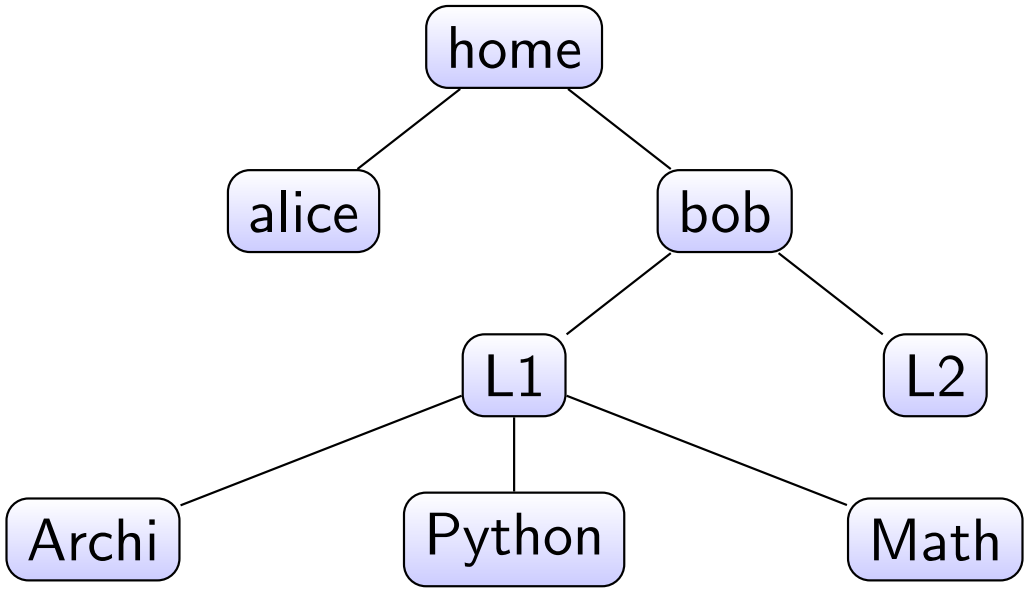
Introduction

Le système mémorise les noms, attributs et adresses des fichiers dans des répertoires (`directories`), qui sont eux-mêmes le plus souvent ... des fichiers.

```
$ df
Sys. de fichiers blocs de 1K  Utilise Disponible Uti% Monte sur
tmpfs                 1605452     2164    1603288   1% /run
/dev/nvme0n1p3       76319516 22980664   49416216  32% /
...
$ sudo debugfs /dev/nvme0n1p3
debugfs:  dump / dumproot
...
$ hexdump -C dumproot | more
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 01 00 26 00  |lost+found....&.|
00000030  0c 00 04 02 62 6f 6f 74  01 00 44 00 0c 00 04 02  |....boot..D.....|
00000040  68 6f 6d 65 0c 00 00 00  10 00 08 01 73 77 61 70  |home........swap|
```

Répertoires hiérarchiques

Dans un système multi-utilisateurs, une structure adaptée est une arborescence de répertoires (une par utilisateur) contenant une hiérarchie de sous-répertoires.



Cette organisation autorise un maximum de souplesse (possibilité de créer des fichiers de noms identiques) et de protection (un seul répertoire racine associé à chaque utilisateur).

Chemins d'accès

Deux méthodes pour spécifier les noms des fichiers :

- un chemin d'accès absolu, constitué à partir du répertoire racine /. Exemple : `/home/bob/L1/Python/prg.py`. Remarque : le caractère séparateur est / sous Unix et \sous Windows.
- un chemin d'accès relatif au répertoire de travail (ou répertoire courant) désigné par l'utilisateur. Tous les chemins qui ne commencent pas à la racine sont relatifs à ce répertoire courant. Exemple : `Python/prg.py`.

La plupart des systèmes qui gèrent une arborescence de répertoires possèdent deux entrées de répertoire spéciales : « . » (courant) et « .. » (son père).

Opérations sur les répertoires

Elles varient davantage d'un système à un autre que celles relatives aux fichiers.

Principaux appels système sous Linux :

- `create` : un répertoire vide (sauf . et ..) est créé,
- `delete` : un répertoire vide (sauf . et ..) est supprimé,
- `opendir` : ouvert pour consultation ou modification,
- `closedir` : libération des tables internes,
- `readdir` : entrée suivante d'un répertoire ouvert,
- `rename` (un répertoire est un fichier),
- `link / unlink` : création et destruction d'un alias.

Les répertoires Linux

`/bin`	user binaries
`/sbin`	system binaries
`/etc`	configuration files
`/dev`	device files
`/proc`	process information
`/var`	variable files
`/tmp`	temporary files
`/usr`	user programs
`/home`	home directories
`/boot`	boot loader files
`/lib`	system libraries
`/opt`	optional applications
`/mnt`	mount directory
`/media`	removable devices
`/srv`	service data

Plan

Stockage des fichiers

Du point de vue du concepteur, on recherche un fonctionnement efficace et fiable.

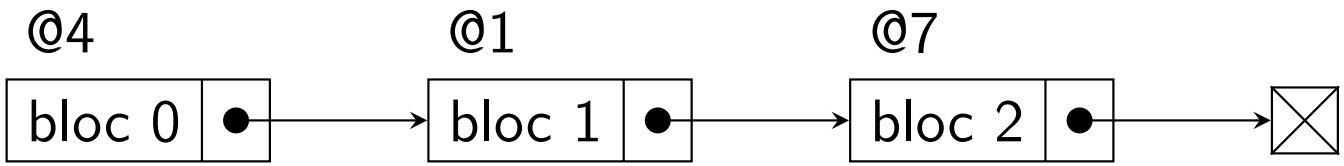
Comment mémoriser l'adresse des blocs de chaque fichier ? Il y a plusieurs méthodes d'allocation.

Contiguë : chaque fichier est stocké dans une suite de blocs consécutifs.

$\oplus$	$\ominus$
simple	taille connue à la création
performances excellentes	fragmentation induite

Stockage des fichiers

Liste chaînée : le premier mot de chaque bloc d'un fichier est un pointeur [2] sur le bloc suivant, le reste du bloc contient les données.



⊕	⊖
pas d'espace perdu	accès « aléatoire » lent
adresse du 1er bloc	

2. variable qui contient une adresse mémoire

Stockage des fichiers

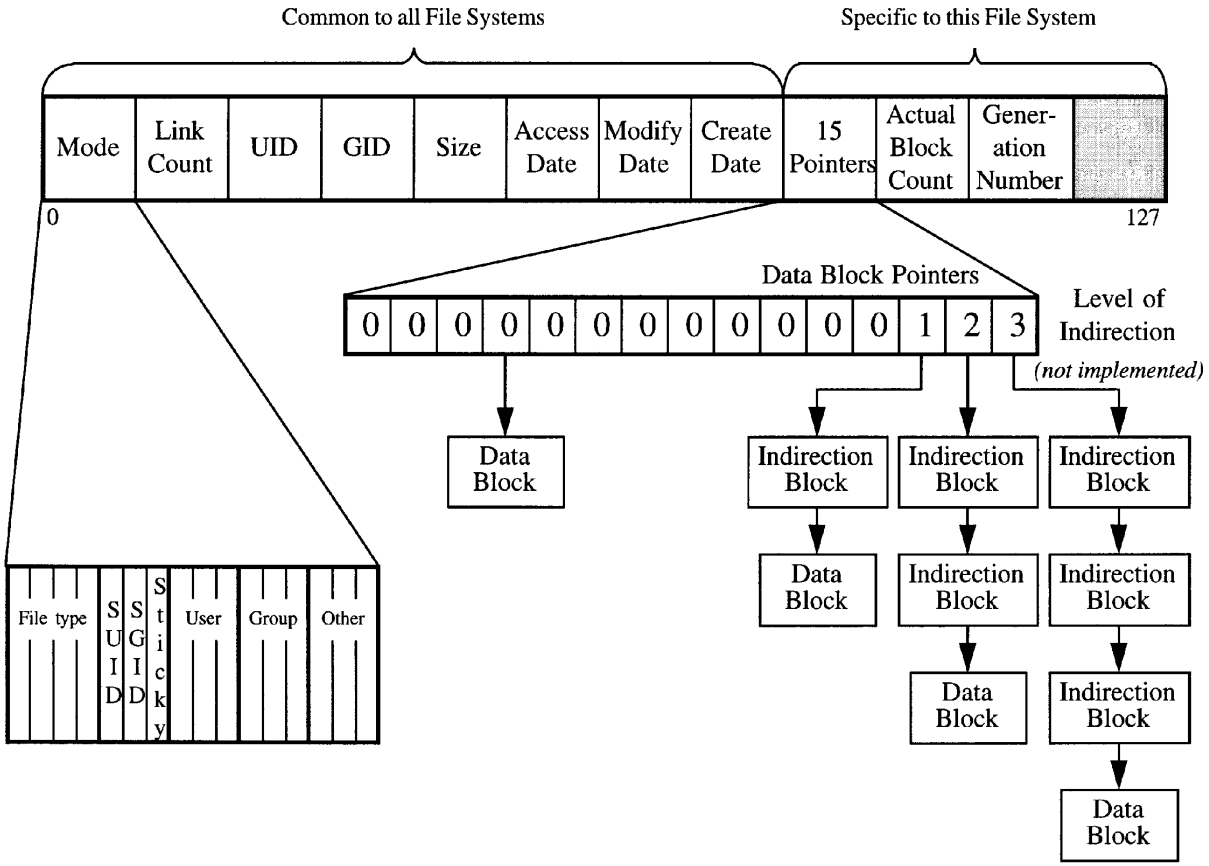
Liste chaînée indexée : élimination des inconvénients de la liste chaînée en utilisant une table.

0	
1	7
2	
3	
4	1
5	
6	
7	NULL

⊕	⊖
liste chargée en mémoire ⟶	en totalité et en permanence
blocs de données uniquement	
accès aléatoires facilités	

Stockage des fichiers

Nœuds d'information (*i-node*) : association d'une petite table à chaque fichier qui contient les attributs et les adresses des blocs.



Stockage des répertoires

Le système est chargé d'établir la correspondance entre chemin d'accès et informations requises pour trouver les données.

Le stockage des attributs associés peut se faire directement dans les entrées des catalogues ou dans les nœuds d'information.

Exemple MS-DOS (1994) :

8 o	3 o	1 o	10 o	2 o	2 o	2 o	4 o
nom	ext.	attr.	(réservé)	heure	date	bloc1	taille

Exemple Unix (1994) :

2 o	14 o
inode	nom

Les fichiers partagés

C'est la possibilité d'avoir un fichier dans plusieurs répertoires qui appartiennent éventuellement à des utilisateurs différents.

On dit qu'il existe un lien (*link*) entre les fichiers des répertoires concernés.

La modification d'un fichier lié doit se répercuter à toutes ses itérations. Pour cela, le système a deux solutions :

- créer un fichier qui contient simplement le chemin d'accès au fichier partagé,
- mémoriser les blocs du fichier partagé ainsi que le compteur de liens dans son *i-node*.

Les fichiers partagés

En cas de destruction du fichier original, la première solution (le chemin d'accès) entraîne une erreur en cas de consultation des liens.

Dans la deuxième solution (*i-node*), le système conserve le i-node, en diminuant le compteur de liens de 1. Lorsque ce nombre passe à 0, le fichier est effacé.

L'utilisation des liens symboliques a un coût en temps (parcours supplémentaire) et en mémoire (informations supplémentaires). De plus, en cas de recherche ou de sauvegarde, le fichier est traité plusieurs fois.

Organisation de l'espace disque

Deux stratégies pour stocker un fichier :

- allouer un seul espace, mais il faudra le déplacer s'il augmente de taille (coûteux),
- diviser le fichier en plusieurs blocs de taille fixe pas nécessairement adjacents.

Taille des blocs : trop grande on perd de la place inutilement, trop petite le temps de lecture d'un fichier augmente considérablement.

Il y a un compromis à trouver, souvent basé sur une étude statistique.

Organisation de l'espace disque

Mémorisation des blocs libres, deux solutions (mémorisées ... dans un espace libre de la mémoire de masse) :

- dynamique : utiliser une liste chaînée de blocs, chaque bloc contenant des numéros de blocs libres,
- statique : utiliser une table de bits, chaque bit codant l'état d'un bloc (libre ou non).

Si la mémoire de masse est pratiquement pleine, la liste chaînée occupe moins de place que la table de bits.

Quotas de disque

Dans un contexte multi-utilisateurs, il convient d'empêcher les utilisateurs d'occuper trop d'espace disque.

Toute augmentation de la taille d'un fichier est déduite du quota de son propriétaire, mais il faut distinguer limites logicielle et matérielle : la première peut être temporairement dépassée contrairement à la seconde.

Lors de la connexion, l'utilisateur est averti d'un dépassement de la limite logicielle un certain nombre de fois jusqu'à interdiction d'accès au compte. Les limites matérielles ne peuvent jamais être dépassées.

Fiabilité

La destruction d'un ordinateur est rarement un probléme (sauf financier), contrairement à celle d'un système de fichiers.

S'il ne peut protéger les équipements contre une destruction physique, il peut, en revanche, protéger les informations.

Gestion des blocs endommagés, deux étapes :

- matérielle : un emplacement est réservé à l'initialisation [3] du support pour établir une liste de correspondance entre blocs endommagés et blocs de substitution,
- logicielle : le système s'appuie sur un fichier qui contient la liste à jour de tous les blocs endommagés pour les retirer de la liste des blocs libres.

3. ou formatage

Fiabilité

Sauvegarde : la pérennité des informations passe par une sauvegarde complète d'un système de fichier (*dump*) si ce dernier est de petite taille.

Pour les systèmes de grande capacité, on a recourt à différentes stratégies de réplication (exemple : *RAID*[4]). La sauvegarde peut être :

- globale (RAID 1 : deux disques miroirs),
- répartie (RAID 0),
- incrémentale (on ne sauvegarde que les fichiers modifiés depuis la dernière sauvegarde globale).

4. Redundant Array of Independent Disks

Fiabilité

Cohérence : quand un système tombe en panne avant récriture des blocs modifiés, il peut se trouver dans un état incohérent (i-nodes, répertoires, liste des blocs libres, . . .).

La plupart des systèmes ont un programme de vérification de la cohérence du système de fichiers au démarrage.

La cohérence se vérifie à deux niveaux :

- blocs : deux tables contenant chacune un *compteur* par bloc, mémorisant le nombre de références à un fichier ou d'apparition dans la liste des blocs libres,
- fichiers : parcours récursif de toute l'arborescence du système de fichiers.

Fiabilité

1ères lignes blocs utilisés, 2èmes lignes blocs libres.

0	1	2	...	62	63
1	1	0	...	0	0
0	0	1	...	1	1

état cohérent

0	1	2	...	62	63
1	1	0	...	0	0
0	0	0	...	1	1

bloc manquant

0	1	2	...	62	63
1	1	0	...	0	0
0	0	2	...	1	1

incohérence blocs libres

0	1	2	...	62	63
1	2	0	...	0	0
0	0	1	...	1	1

incohérence blocs utilisés

Fiabilité

Pour vérifier la cohérence des fichiers, le système parcourt récursivement l'arborescence et incrémente un compteur pour chaque i-node rencontré.

À la fin du traitement, on obtient une liste indexée par les numéros des i-nodes associés au nombre de références trouvées.

En comparant ces nombres l_i avec ceux contenus dans les i-nodes n_i, on peut déterminer si le système de fichiers est dans un état cohérent ($n_i = l_i$).

Dans le cas contraire :

- si $n_i > l_i$: le i-node n'est jamais supprimé, perte de place,
- si $n_i < l_i$: le i-node est supprimé de façon prématuré, les blocs associés sont considérés libre.

Fiabilité

D'autres contrôles sont possibles :

- nombre de liens d'un i-node ou d'entrées d'un répertoire très élevé,
- droits d'accès suspects (interdits au propriétaire, autorisés aux autres),
- super-utilisateur propriétaire d'un fichier utilisateur, etc.

Il faut aussi protéger l'utilisateur contre ses propres erreurs. Exemple classique : `rm * .txt`. Une solution possible est de retarder la mise à disposition des blocs associés (corbeille).

Performance

L'accès à la mémoire de masse est beaucoup plus lent (facteur 100 000) que l'accès à la mémoire centrale.

Une solution est de réduire au maximum le nombre d'accès.

La plus courante est l'antémémoire (*cache*[5]), une suite de blocs appartenant logiquement à la mémoire de masse mais située en mémoire centrale.

Problème : en cas de panne, le système a de forte chance d'être dans un état incohérent.

Les stratégies mises en œuvre pour le cache matériel (FIFO, LRU, ...) doivent être adaptées, en particulier pour l'écriture des blocs.

5. du verbe français cacher

Performance

L'approche MS-DOS consistait à écrire un bloc dès sa modification (*write-through cache*), ce qui génère beaucoup d'accès à la mémoire de masse.

Une approche plus performante est d'établir un cycle de sauvegardes (par exemple toutes les 30 s) au moyen d'une boucle sans fin qui effectue l'appel système idoine (`SYNC`).

Attention : si on retire le périphérique mémoire avant un `SYNC`, on est presque sûr de perdre des données ou pire de rendre le système de fichier incohérent.

Autre solution : réduire les déplacements en rapprochant les blocs susceptibles d'être adressés en séquence, c'est la défragmentation (inutile pour un SSD).

Plan

Contexte

Les systèmes de fichiers contiennent des informations très importantes qu'il convient de protéger des accès non autorisés.

Pour cela, il faut distinguer sécurité (problèmes) et protection (mécanismes utilisés par l'OS pour assurer la sécurité).

La sécurité est concernée par :

- les pertes de données : catastrophes naturelles, erreurs matérielles ou logicielles, erreurs humaines,
- les intrus : indiscrétions, défis, appât du gain, espionnage.

Failles

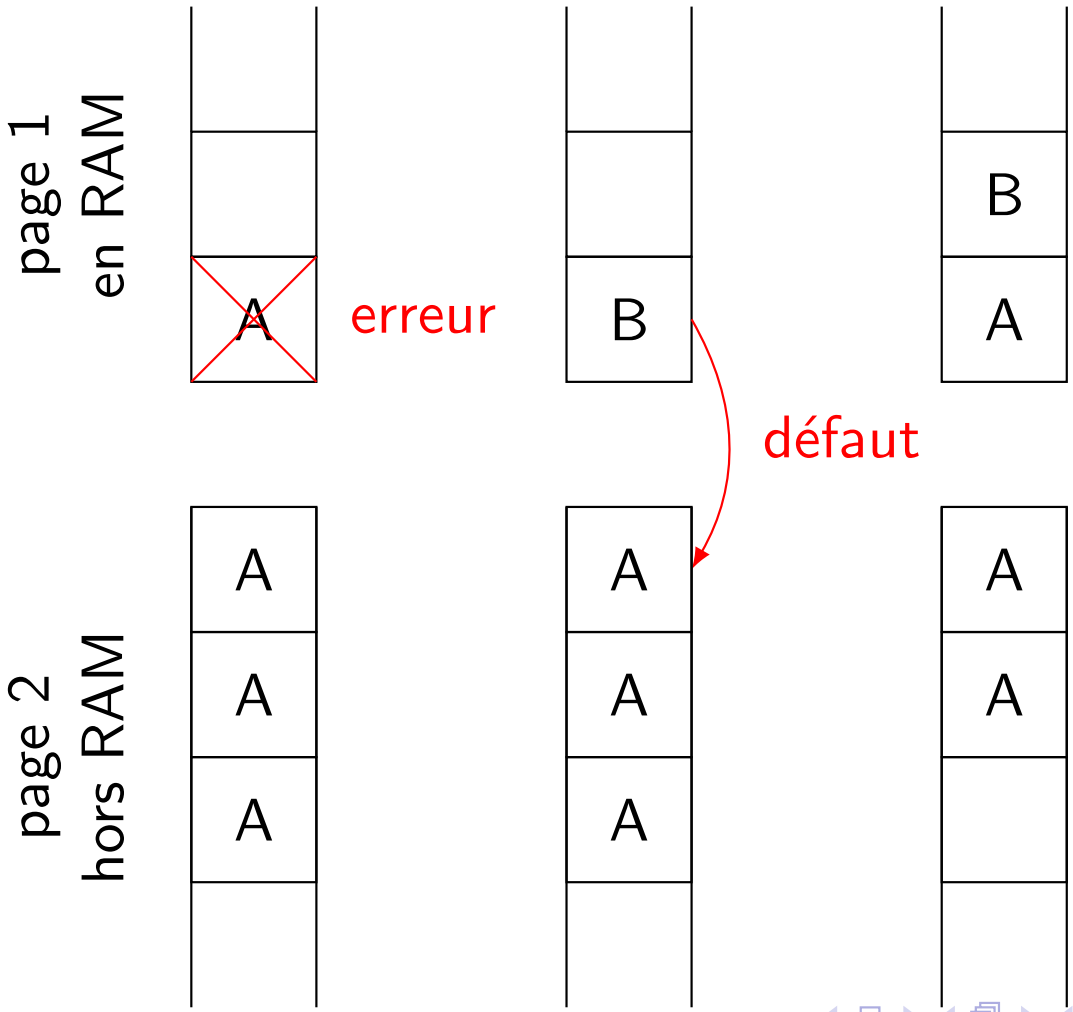
L'une des plus simples à comprendre repose sur le système de pagination de l'OS Tenex. Lorsqu'un défaut de page avait lieu, on pouvait lui associer l'appel d'une fonction.

Sachant que la vérification d'un mot de passe se faisait caractère par caractère, un hacker a établi la stratégie suivante : placer le caractère à tester à la fin d'une page, et les caractères suivants dans une autre page absente de la mémoire.

Soit ce caractère n'est pas valide et le processus s'interrompt, soit il est valide et lors de la recherche du caractère suivant un défaut de page est signalé au hacker.

Failles

Mot de passe à trouver : `Bxxx`



Vers (*Worm*)

Le ver Morris (1988) : premier ver et première condamnation.

[Wiki] Ce ver exploitait deux vulnérabilités connues dans `sendmail` et `fingerd`, ainsi que la faiblesse des mots de passe de certains utilisateurs qu'il tentait de décrypter une fois installé.

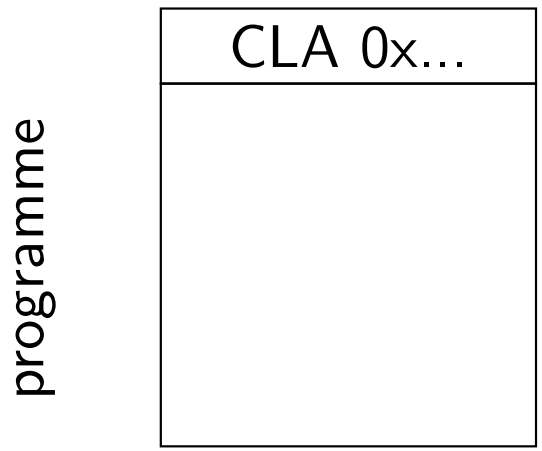
En mode 'debug', `sendmail` permettait d'envoyer des fichiers sur machine distante, dans ce cas une « amorce » qui utilisait un shell pour être compilé. Pour cela, il appellait `finger` et provoquait un débordement mémoire du démon `fingerd` qui lançait l'exécution du shell local.

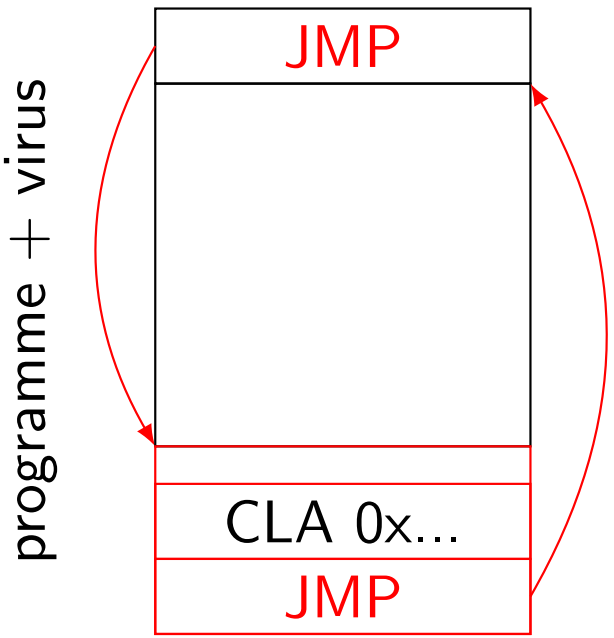
L'amorce se connectait à la machine d'où elle provenait, transférait le ver et l'exécutait. Le ver examinait les tables de routage de la machine hôte pour identifier les machines accessibles puis tentait de diffuser l'amorce à celles-ci.

Virus

Un virus est un morceau de programme rattaché à un programme normal et qui tente de se répliquer dans d'autres programmes.

La première instruction est remplacée par un branchement à son code, puis il exécute celle-ci et effectue un branchement à la deuxième.





Conception d'un système sécurisé

Principes généraux :

1. rendre la conception publique,
2. refuser la connexion en cas de doute,
3. vérifier les droits d'accès régulièrement,
4. donner à chaque processus le moins de privilège possible,
5. les mécanismes de protections doivent être simples, uniformes et implantés dans les couches les plus basses du système,
6. les mécanismes retenus doivent être psychologiquement acceptable.

Identification de l'utilisateur

La vérification de l'identité d'un utilisateur se fait la plupart du temps à la connexion.

La forme la plus courante d'identification est le mot de passe (crypté). Mais selon Google (2021), 24 % de leurs utilisateurs ont utilisé le mot "password", "Qwerty" ou "123456".

Selon Ponemon Institut [6] (2021) :

- 42 % des organisations s'appuient sur des notes autocollantes,
- 59 % des entreprises se fient à la mémoire humaine pour gérer les mots de passe.

6. centre de recherche américain sur la protection

Identification de l'utilisateur

Cette forme de protection est souvent doublée ou remplacée par une autre, basée sur :

- l'identification physique : empreintes digitales, spectre de la voix, reconnaissance rétinienne, longueur des doigts, analyse d'urine (!) ou de sang,
- défense active : exigence cyclique d'identité, enregistrements de toutes les connexions (*proxy*), pièges (*honeypot*),
- la double authentification (MFA [7]) : à « ce que vous savez » (mot de passe) se combine « ce que vous possédez » (code reçu par mail ou par SMS, jeton USB, carte à puce).

7. multi-factor authentication

Mécanisme de protection

Types d'objets à protéger :

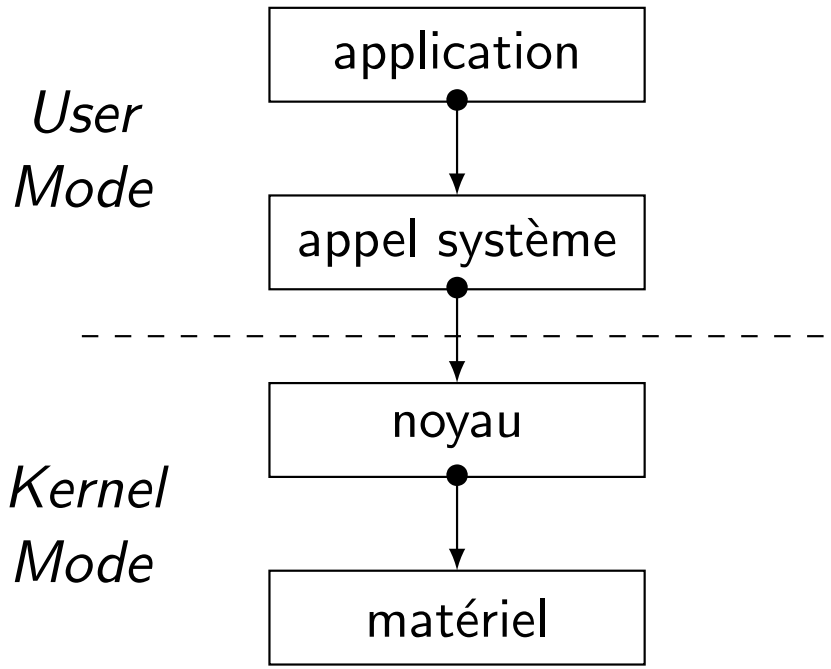
- matériel : processeurs, mémoires, terminaux, imprimantes, ...
- logiciel : processus, fichier, bases de données, ...

Domaine : ensemble de paires (objet, droits), un objet peut être dans plusieurs domaines avec des droits différents, les droits étant l'ensemble des opérations autorisées.
Exemple : pour un fichier ce serait `Read`, `Write` et `eXecute`.

Réciproquement, on associe à chaque objet une liste des domaines qui peuvent y accéder et de leur mode d'accès.
Exemple : sous Unix, chaque fichier possède trois fois trois bits (rwx) pour le propriétaire, son groupe et les autres utilisateurs.

Mécanisme de protection

Un processus s'exécute dans un domaine de protection donnée (et peut en changer en cours d'exécution).
Exemple : sous Unix, le domaine d'un processus est déterminé par la paire (uid, gid), de plus un processus possède deux parties, utilisateur et noyau (appels système).



Canaux cachés (*covert channel*)

Malgré toutes les précautions prises, il est toujours possible que des processus échangent des informations confidentielles.

Dans le contexte d'un serveur protégé des indiscrétions (analyses médicales, déclarations de revenus, etc.), on peut mettre en place des canaux de communication d'informations basés sur :

- la modulation d'utilisation du processeur (très utilisé pour 1, en sommeil pour 0),
- la modulation du taux de défauts de pages (élevé pour 1, nul pour 0),
- l'obtention et la libération de ressources dédiées (fichiers, imprimantes, etc.),
- ...