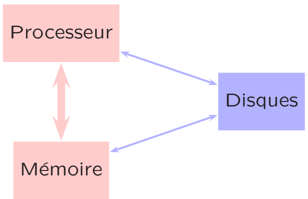
Organisation et gestion mémoire

- Rôle de la mémoire
- Accès direct à la mémoire
- Mémoire segmentée
- Mémoire paginée

Rôle crucial de la mémoire

Sollicitée en permanence



Faible nombre de registres : on ne peut presque rien faire sans la mémoire.

À part des calculs numériques, les algorithmes parcourent des structures de données complexes représentées en mémoire.

Il y a alors des accès mémoires toutes les 2-5 instructions...

Rôle crucial de la mémoire

Sollicitée en permanence

Goulot d'étranglement

- Les opérations mémoires sont plus lentes que les instructions des processeurs (croissance plus rapides).
- Plusieurs niveaux de caches.

Rôle crucial de la mémoire

Sollicitée en permanence

Goulot d'étranglement

La sécurité passe par la mémoire

- Au minimum : zone système et zone utilisateur.
- Si possibles, protéger des zones plus fines parmi les zones utilisateurs.

Rôle crucial de la mémoire

Sollicitée en permanence

Goulot d'étranglement

La sécurité passe par la mémoire

La modularité passe par la mémoire

- Avoir plusieurs programmes chargés en parallèle.
- Partageant du code.

Rôle crucial de la mémoire

Sollicitée en permanence

Goulot d'étranglement

La sécurité passe par la mémoire

La modularité passe par la mémoire

Besoin de support matériel

- Pour pouvoir faire les vérifications de sécurité.
- Pour les faire efficacement.
- Pour permettre et faciliter le partage de code.

Contrôle d'accès souhaité

Limiter les accès à la mémoire en écriture, lecture, etc.

- Pour la sûreté : avoir des zones non écrivables par l'utilisateur.
  - La zone système.
  - Les zones des autres utilisateurs.
- Pour la sécurité (secret) : contrôler également la lecture.
- Pour le confort : l'utilisateur peut lui-même protéger certaines zones en écritures.
- Pour l'efficacité : copie à l'écriture.
  - On interdit une zone en écriture.
  - On rattrape l'interruption de violation de droit pour sauvegarder la zone, puis autoriser l'écriture et reprendre l'exécution.

Autres opérations souhaités

Programme partiellement chargé en mémoire

- Si le programme ne loge pas tout entier en mémoire centrale. (La mémoire disque est plus grande que la mémoire centrale).

Plusieurs programmes chargés en mémoire

- Peuvent-ils partager un même sous-programme ?
- Nécessite du code réentrant :
  Comment le permettre/favoriser ?

Code relogeable

- On veut pouvoir sauver un programme en train de tourner sur le disque (par manque de place), puis plus tard le recharger en mémoire, éventuellement à un autre emplacement.

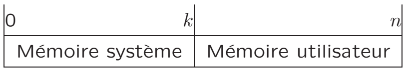
Accès direct à la mémoire

Ce modèle est obsolète. Tous les processeurs modernes ont une unité de mémoire virtuelle. Il permet de comprendre les besoins.

Schéma

- Le code fait référence à des adresses (du code *et* des données)
- Ces adresses sont des adresses réelles.

Protection mémoire



- Une barrière, k (constante ou variable), permet de séparer les zones système et utilisateur.
- Chaque accès à une adresse a en mode utilisateur vérifie que $a > k$ et lève une interruption sinon.

Accès direct à la mémoire

Ce modèle est obsolète. Tous les processeurs modernes ont une unité de mémoire virtuelle. Il permet de comprendre les besoins.

Schéma

- Le code fait référence à des adresses (du code *et* des données)
- Ces adresses sont des adresses réelles.

Protection mémoire

Réalisée par le matériel

- Opérations simples et peu coûteuses à implémenter *par le matériel* (une comparaison).
  - Coûteuses à implémenter par le logiciel.
  - Le logiciel ne peut pas garantir la sécurité.

Swap

Principe

- Les processus sont chargés en mémoire.
- Lorsqu'il n'y a plus de place en mémoire, un processus est entièrement copié sur le disque (dans une partition réservée).
- Plus tard, le processus est rechargé en mémoire pour pouvoir être à nouveau exécuté.
- Le rechargement ne se fait pas forcément à la même place, d'où la nécessité de code relogeable.

Swap

Principe

Pratique

- Quand swapper un processus ?
  - Lorsqu'il y a besoin de mémoire (appel système `fork`, grossissement d'un processus, *etc.*)
  - Les processus endormis sont swappés par priorité.
  - Ainsi que les processus résidants depuis longtemps.
- Quand le ramener en mémoire ?
  - Lorsqu'il est prêt à tourner (travail de l'ordonanceur)
- Les ajustements sont importants :
  - Pour éviter que le système passe son temps à swapper...

Organisation de l'espace de swap

Gestion spécialisée avec allocation de blocs contigus (accès plus rapides) contrairement à une partition pour fichiers.

Code relogeable

Édition de liens

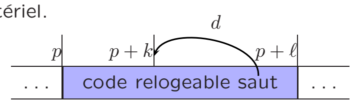
- Un compilateur produit du code sans savoir dans quelle partie de la mémoire ce code sera logé.
- Les adresses symboliques sont résolues au chargement, lorsqu'on connaît les adresses réelles (statiques).

Code relogeable

Édition de liens

Code relogeable (contrainte plus forte)

- On veut pouvoir déplacer le code une fois chargé en mémoire
- Il suffit que le compilateur référence les adresses du code par décalage par rapport à un registre de base (par exemple le début de la zone utilisateur).
  - Permet un calcul dynamique des adresses réelles du code.
  - Peut aussi se faire au niveau logiciel (génération de code) sans support matériel.



  - L'adresse $p + k$ est un saut relatif de d par rapport à pc ou de k par rapport à la $base$ qui vaut ici p pendant l'exécution.

Code relogeable

Édition de liens

Code relogeable (contrainte plus forte)

Remarque : le calcul $p + k$ peut être

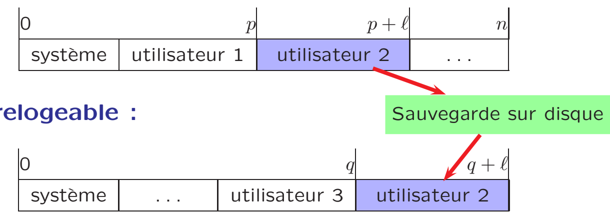
- logiciel : tous les accès sont écrits avec indirection.
- matériel : l'indirection est transparente (tous les accès sont faits par rapport au registre de base courant).

Régions mémoires

Une amélioration de la proctection mémoire

Une région est une zone contiguë de mémoire délimitée par son adresse de *base* p et sa *taille* ℓ (mis dans des registres spéciaux).

Code relogeable :



Remarque

- La taille ℓ peut être ajustée dynamiquement

Code réentrant, code partagé

Définition

Un code est réentrant s'il ne se modifie pas lui-même en cours d'exécution (pas de variables globales dans le code).

Solution On sépare le code des données.

- Le code n'est pas (en général) modifié :
  on peut le partager (même segment dans plusieurs processus).
- On ne partage pas les données :
  - Deux processus ont des espaces de données différentes (différents segments).
  - Le code référence les mêmes adresses, même numéro i mais valeurs de $S[i]$ différentes.

Code réentrant, code partagé

Définition

Un code est réentrant s'il ne se modifie pas lui-même en cours d'exécution (pas de variables globales dans le code).

Solution

- Le code est référencé par rapport à la base c
- Les données sont référencées par rapport à la base d

c		d		
Code	. . .	Données → ← Pile	. . .	Données → ← Pile

Processus : A

Code réentrant, code partagé

Définition

Un code est réentrant s'il ne se modifie pas lui-même en cours d'exécution (pas de variables globales dans le code).

Solution

- Le code est référencé par rapport à la base c
- Les données sont référencées par rapport à la base d

c				d
Code	. . .	Données → ← Pile	. . .	Données → ← Pile

Processus : B

Généralisation : Segmentation

- La mémoire est coupée en (grosses) régions appelés segments (par exemple 10k segments).
- Un segment s est défini par une adresse de base $s.b$ (sa position dans la mémoire physique) et une limite $s.\ell$ (sa taille).
- Un table $i \mapsto S[i]$ permet d'identifier les segments.
- Les processus utilisent des tables de segments différentes.
- Pour adresser la position d dans le segment i, il faut :
  - précharger l'identification du segment $s[i]$ dans un registre de segment dédié (code, donnée, . . .)
  - L'adresse d se réfère alors à la position $s[i].b + d$ si $d < s[i].\ell$
- Permet d'adresser plus de mémoire qu'il n'y a d'adresses.

Variante : Une adresse peut aussi être de la forme $i.d$ où i désigne le segment. On peut alors utiliser une mémoire associative pour calculer le registre base et la limite.

Insuffisance

Problèmes

- La zone mémoire d'un processus est contiguë (au moins à l'intérieur d'un segment s'il y a des segments).
- Un processus doit être entièrement en mémoire, ou bien, le compilateur doit le segmenter manuellement. *i.e.* prévoir manuellement le chargement/déchargement des segments.
- Il y a possibilité de fragmentation de la mémoire : impossibilité d'allouer une grosse zone alors qu'il y a plein d'espaces libres mais de petites tailles.

Remarques

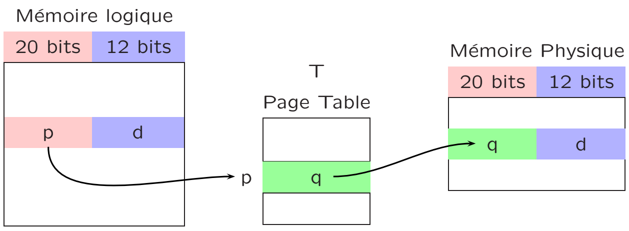
- Les segments partitionnent la mémoire en grosses zones.
- Les adresses peuvent être référencés par rapport au début du segment (le logiciel ajoute un décalage)

Mémoire paginée

Principe

- La pagination est un mécanisme qui permet une indirection (traduction) *systématique* et *dynamique* entre les adresses logiques et les adresses physiques.
- Elle est transparente aux programmes qui n'utilisent que des addresses logiques sans se soucier de leur emplacement physique.

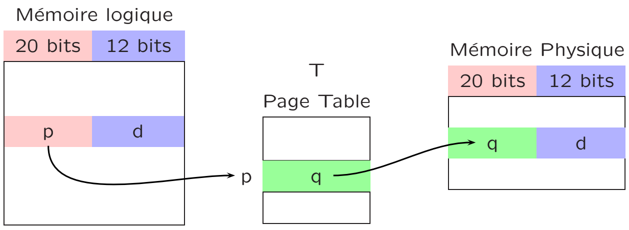
Pages mémoire



Principe

- Les adresses manipulées sont *logiques* et *systématiquement* et *automatiquement* traduites en adresses *physiques*.
- La mémoire est organisée en pages (≈ 4K octets) logiques. Les numéros de pages logiques sont traduites en pages physiques. Le décalage est conservé.

Pages mémoire



Principe

- Chaque processus à sa table des pages :
  - Possibilité de partager les pages entre processus
- On ajoute des droits d'accès sur chaque page.
  - **R**ead, **W**rite, **C**opy-**O**n-**W**rite, **E**mpty

Aspects pratiques

Mémoire associative

On cache les (16–64) dernières valeurs $p \mapsto q$ dans une mémoire associative TLB qui fait une recherche en parallèle.

- Si TLB contient une entrée $p \mapsto q$ valide, alors $T[p] = q$.
- Sinon, on calcule $q = T[p]$ et on ajoute $p \mapsto q$ dans TLB après avoir retiré une (vieille) entrée dans TLB si nécessaire.

Aspects pratiques

Mémoire associative

Tables des pages hiérarchisée

La table des pages T peut être très grosse (1M entrées ou plus), mais la plupart des entrées sont des pages vides.

Elle prend beaucoup de place en mémoire !

On l'organise en un tableau à deux (ou trois) dimensions.

$$T[p_1 \cdot p_2] = T[p_1][p_2]$$

Les pages de pages dont toutes les entrées sont vides peuvent être représentées par des pages vides, ce qui permet de conserver une petite taille. (*c.f.* représentation des 0 dans les fichiers.)

Aspects pratiques

Mémoire associative

Tables des pages hiérarchisée

Table des pages inversée

Une autre solution pour réduire la taille de la table des pages est de la remplacer par une table de hache. La place est proportionnelle au nombre de pages non vides et non au nombre de pages logiques.

On peut alors se contenter de maintenir une table $q \mapsto p$ des pages physiques vers les pages logiques pour résoudre les conflits de hash.

Avantages immédiats de la pagination

- Le fragment de mémoire est une page, de taille fixe.
  - La mémoire réelle d'un processus n'est plus de taille fixe, mais une liste de pages que l'on peut facilement agrandir.
  - Plus de fragmentation : les pages n'ont pas à être contiguës.
  - La protection mémoire peut se faire page par page.
  - Copie à l'écriture possible.
  - Seule la mémoire utilisée d'un processus coûte (les pages vides ne sont pas allouées .)
- Chaque processus peut utiliser les mêmes adresses logiques, comme si la machine et son espace mémoire étaient à lui seul.
- Deux processus peuvent partager de la mémoire en utilisant la même page physique (pages logiques identiques ou non).
  - partage de code, séparation de leurs données et de leur pile.
  - partage des données en totalité (coprocessus) ou en partie.

Segmentation + Pagination

Combinaison

- La mémoire est segmentée ;
- L'adresse linéaire après résolution de la segmentation est une adresse logique qui est paginée.
- Les adresses sont de la forme $i \cdot p \cdot d$
- La TLB cache les adresses $i \cdot p \mapsto q$ (page physique)

Mémoire à deux dimensions

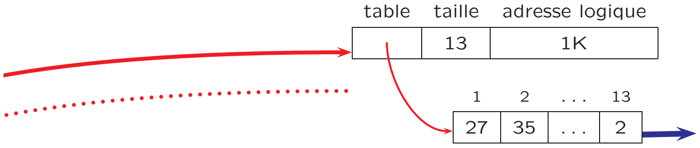
- On peut allouer de nouveaux segments
- On peut agrandir un segment en allouant de nouvelles pages, sans risque de chevauchement.

En fait... combinaison peu utilisée

- La pagination seule donne l'illusion d'un adressage à deux dimensions si l'espace virtuelle est très grand.

Organisation des pages en régions

Région = espace contigu d'adresses logiques

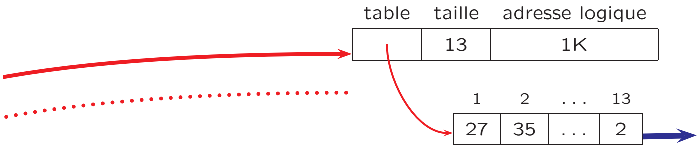


Représentation d'une région

- La première adresse logique de la région.
- Le tableau des pages physiques.
- La taille (nombre de pages).

Organisation des pages en régions

Région = espace contigu d'adresses logiques

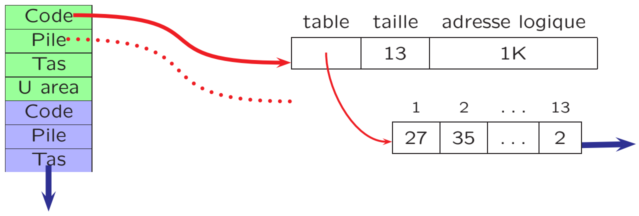


Principe

- Une région peut allouer/désallouer de nouvelles pages.
- Les pages peuvent être partagées (compteur de référence).

Organisation des pages en régions

Région = espace contigu d'adresses logiques

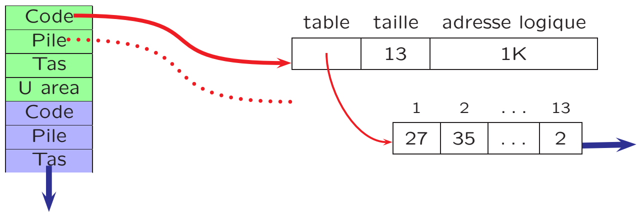


Espace à 1.9 dimension...

- Les régions peuvent grossir physiquement (plus de pages)
- Mais leurs espaces logiques ne peuvent pas se chevaucher !
- La grande taille de l'espace logique donne l'illusion de deux dimensions.

Organisation des pages en régions

Région = espace contigu d'adresses logiques



Régions = Organisation logicielle des pages

- Indépendante du matériel

La pagination

Principe

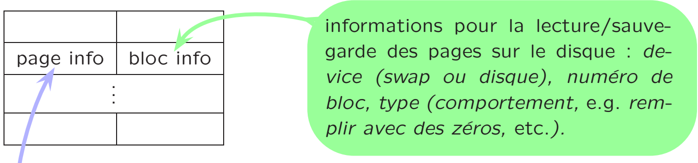
- C'est un raffinement de la notion de `swap`.
- La mémoire est sauvegardée sur le disque page par page plutôt que processus par processus.
  - Un processus peut s'exécuter avec une partie de ses pages cachées sur le disque (qui peuvent être temporairement inutilisées.)
  - Mieux, un processus peut s'exécuter sans jamais être complètement chargé, *e.g.* sa mémoire virtuelle est plus grande que la mémoire réelle.

Mise en œuvre

- Stratégie de pagination (et structures-système associées)
- Optimisation (réduction) des transferts mémoire.

Information sur les pages

Les entrées du tableau des pages physiques d'une région sont de la forme :



Adresse	de la page physique	géré par le système
Age	depuis le dernier swap	
Cp/Wr	copie à l'écriture ?	
Mod	Page modifiée ?	géré par le matériel
Ref	Page référencée ?	

Le voleur de pages

C'est un processus système

- ne s'exécute qu'en mode système
- ordonnancé comme un autre processus (plus forte priorité)
- activé lorsque la mémoire libre descend en dessous d'un seuil critique.
- Mécanisme (presque) transparent pour le reste du système.

Stratégie : vieillissement des pages (exemple)

- Le voleur de pages examine périodiquement l'ensemble des pages en mémoires, et ajuste leur âge.
  - Un bit géré par le matériel indique si la page a été récemment référencée (depuis la dernière passe).
  - Si oui, l'âge de la page est remis à zéro.
  - Sinon, l'âge de la page est incrémenté.
  - Le bit est remis à zéro.

Le voleur de pages

C'est un processus système

- ne s'exécute qu'en mode système
- ordonnancé comme un autre processus (plus forte priorité)
- activé lorsque la mémoire libre descend en dessous d'un seuil critique.
- Mécanisme (presque) transparent pour le reste du système.

Stratégie : vieillissement des pages (exemple)

- Le voleur de pages examine périodiquement l'ensemble des pages en mémoires, et ajuste leur âge.
- Les pages vieilles (seuil à ajuster) sont swappées jusqu'à ce que la mémoire libre atteigne un seuil satisfaisant.
- On peut ne parcourir qu'un petit nombre de pages à chaque fois en organisant les pages en cercle (clock algorithm).

Sauvegarde des pages

Principe

Forte ressemblance avec l'allocation/déallocation des blocs.

- Si la page n'est pas sur le disque, le voleur de page la place dans une liste de pages à copier sur le disque.
- Si la page est déjà sur le disque...
  - Elle devient recyclable si elle n'a pas été modifiée.
  - Sinon, elle est programmée pour être écrite sur le disque, a un nouvel emplacement et l'ancien emplacement est libéré.

Sauvegarde des pages

Principe

Forte ressemblance avec l'allocation/déallocation des blocs.

- Si la page n'est pas sur le disque, le voleur de page la place dans une liste de pages à copier sur le disque.
- Si la page est déjà sur le disque...
- Lorsque la liste est suffisamment longue, Les pages sont effectivement copiées sur le disque dans une zone contigüe (plus efficace).
- Les pages deviennent alors effectivement recyclables.

Remarque : les pages recyclables restent valides tant qu'elles n'ont pas été réutilisées...

Rappel des pages

Une faute de page provoque une interruption.

Soit la page est invalide

- La page est référencée sur le disque (swap ou fichier exécutable) : Elle peut être ramenée en mémoire
  Remarque : le processus est mis en attente. C'est un cas particulier où une interruption bloque, mais seulement le processus courant.
- La page n'est pas référencée : c'est une violation mémoire.

Soit la page est protégée (en écriture ou lecture)

- Si la page est Cp/Wr (copie à la lecture), il faut allouer une nouvelle page et l'initialiser avec la page actuelle.
- Sinon, c'est une violation de droit.

Optimisation de l'appel système fork

Problème

- L'appel système `fork` doit copier la mémoire du processus appelant pour créer le fils.
- C'est potentiellement très coûteux, notamment si `fork` est suivi de `exec`.
  Remarque : `vfork` était une variante (dangereuse) de `fork` qui ne copie pas la mémoire et suppose que le fils fait `exec` immédiatement après (le comportement est indéfini sinon).
  C'était essentiel pour des raisons d'efficacité dans les systèmes sans pagination, mais devenu inutile dans les systèmes avec pagination.

Optimisation de l'appel système fork 23(2)/30

Problème

- L'appel système `fork` doit copier la mémoire du processus appelant pour créer le fils.
- C'est potentiellement très coûteux, notamment si `fork` est suivi de `exec`.

Solution

- La table des pages est copiée mais les pages elles-mêmes sont seulement marquées `Cp/Wr`.
  (En fait, seules les pages de pages sont copiées dans un système à double pagination...)
- Les pages qui ne sont jamais modifiées ne sont jamais copiées
  - Le code, en général read-only.
  - Les pages désallouées (lorsque `fork` est suivi de `exec`).

L'appel système exec

Problème

L'appel `exec` doit charger le code d'un fichier exécutable en mémoire.

- Cela peut-être coûteux si le code est gros et seulement une partie du code est finalement utilisée.
- Le code est partagé (read-only) et est peut-être déjà chargé en mémoire.

L'appel système exec 24(2)/30

Problème

L'appel `exec` doit charger le code d'un fichier exécutable en mémoire.

Solution

- Le code n'est pas chargé immédiatement. Seule la table des pages est créée (en examinant la taille du fichier).
  Les pages sont décrites comme étant sur le disque (comme les pages swappées mais elles sont dans une partition régulière).
- Elle seront chargés paresseusement lors d'un défaut de page.
  Seules les pages effectivement lues seront chargées
- Si le fichier est déjà chargé par un autre processus (table de hash inverse *bloc-sur-disque* $\mapsto$ *page*), alors les pages sont partagées.

Projection d'un fichier en mémoire

- Permet d'allouer une région dont le contenu est identique à celui d'un fichier.
- Accès direct sans passer par `read`/`write`.
- La projection peut être privée (copie à l'écriture) ou partagée.
  - Pour un fichier structuré, permet un accès direct à une adresse calculée (sans passer par lseek).
  - Évite des recopies si le mode est partagée ou privée mais sans écriture.

Retour sur le rappel des pages invalides

1. La page est sur la partition swap.
   Elle est lue sur le disque, sauf si elle est en train d'être chargée ... (défaut de page d'un autre processus) : attente.
2. La page est dans la liste des pages libres.
   Elle n'a pas encore été réutilisée : elle est récupérée.
3. La page désigne une partie d'un fichier exécutable.
   Similaire à 1. mais la page est lue dans le fichier exécutable.
4. La page est `à remplir de zéro`
   La page est créée et remplie de zéro.
5. La page est `à remplir`
   La page est simplement crée (son contenu quelconque).

Pagination et swap

Complémentarité

Les deux systèmes peuvent être combinés.

- Les deux mécanismes ont des stratégies différentes et sont complémentaires.
- Si le voleur de pages n'arrive pas à trouver suffisamment de pages vieilles, alors le mécanisme de swap arrive à la rescousse : un processus est sauvé en totalité sur le disque.
- Lorsque le processus est ramené en mémoire, le rechargement des pages se fera paresseusement par le mécanisme de défaut de pages.

Remarques

Beaucoup de sous-entendus, à détailler

- Algorithmes,
- Structures de donnée et mise en œuvre.

De nombreuses variations

- Des variantes logiciels : d'autres stratégies sont possibles.
- Variantes matériel très importantes d'une architecture à une autre.
- Les principes restent similaires.

Résumé

L'abstraction mémoire

- Chaque processus voit la mémoire comme si elle était tout entière à lui seul avec un espace d'adressage énorme.
- C'est un espace linéaire organisé en régions contigües protégées (séparées par des trous).
- Le matériel et le logiciel permettent :
  - De garantir la sécurité (protection mémoire).
  - Un partage de la mémoire entre processus.
  - Sans surcoût à runtime ; au contraire. . .
  - Allocation et copie sont effectuées paresseusement.

Équilibre entre les solutions matérielles/logicielles

- Lorsque la solution logicielle est trop lente.
- Une petite aide matérielle la rend raisonble (e.g. cache).
- Réconcilie expressivité et efficacité.

Les trois grandes abstractions

Les descripteurs de fichiers

- Abstaction de toute la communication entre les processus et avec le système de fichiers.
- Sur le disque, les fichiers sont une suite de blocs.
- Pour l'utilisateur, les fichiers sont une suite d'octets.
- Entre les deux une représentation sophistiquée pour améliorer les performances.

Les trois grandes abstractions

Les descripteurs de fichiers

Les processsus

- Chaque processus se comporte comme s'il avait la machine à lui tout seul (CPU, appels systèmes, disque, *etc.*)
- Un processus est l'entité élémentaire du calcul, d'allocation mémoire, de traitement des signaux et de communication.

  Même si un programme peut être organisé autour de plusieurs processus qui communiquent entre eux.

Les trois grandes abstractions

Les descripteurs de fichiers

Les processsus

La mémoire virtuelle

- Elle donne l'illusion que les processus sont seuls.
- Tout en gérant le partage des ressources entre les processus à un coût raisonable.

Les trois grandes abstractions

Les descripteurs de fichiers

Les processsus

La mémoire virtuelle

Modularité, complémentarité, inter-dépendance

- Séparation des concepts/trois abstractions.
- Des solutions similaires, voire partagées :
  - Cache d'inodes / Pages mémoires
  - Swapping / Projection d'un fichier en mémoire
- Des solutions interdépendantes :
  - Fork efficace grace à la pagination et Copy-On-Write
  - Latence réduite par un chargement paresseux à la demande.