Chapitre 8

Mémoires

– Généralités

– Objectifs du gestionnaire de mémoire

8.1 Mémoire : généralités

- Différents type = différentes stratégies
- Caractéristiques communes :
    - Structure hiérarchique
    - Gestion par le SE des programmes user en MC
    - MC et MS (swap, mémoire virtuelle) : allocation, gestion, libération, ...

8.1.1 Hiérarchies de mémoires

- Plusieurs types de mémoire reçoivent des représentations ϕ des programmes ;
- Pour le SE, l'important est la vitesse d'accès et la capacité de stockage ;
- Rapport inverse coût / vitesse / capacité
- 2 types de mémoires : **volatile** et **permanente**

8.1.2 Mémoire Volatile

- Circuits intégrés ;

- Accès rapide ;

- Coût élevé ;

- Une ressource critique ;

- 2 types courants : **Registres** et **Mémoire Centrale** (MC) ;

Voir aussi la mémoire cache ;

⊡ Un programme est chargé en MC :

l'UC transfère les instructions de la MC vers les registres pour les exécuter ; puis range les résultats dans la MC ;

- Registres d'adresse de de données ; nombre limité ; accès rapide ;

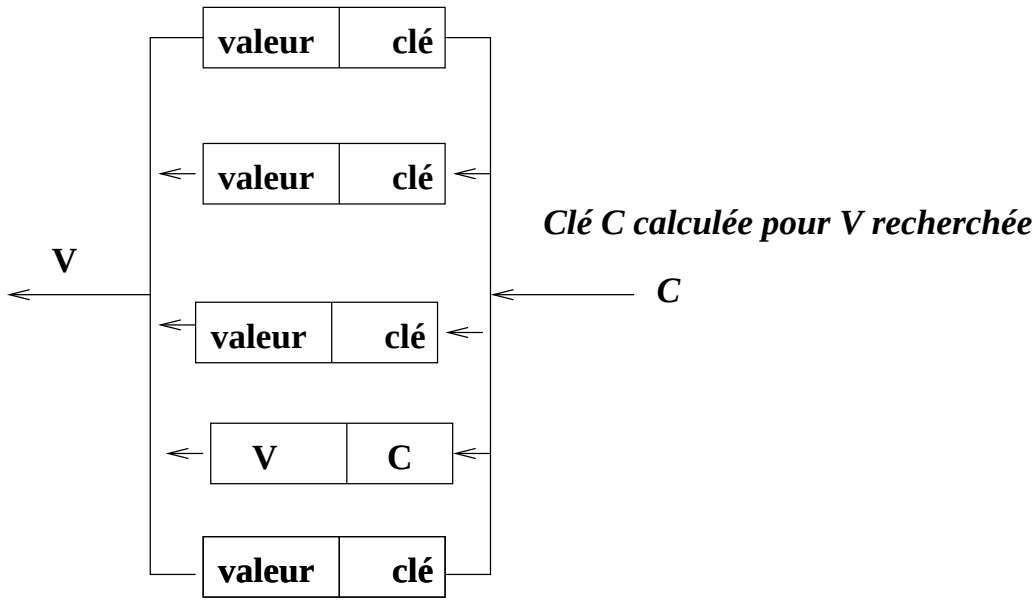
⊡ Mémoire **cache** : volatile, accès rapide ; fonctionnement comme un tampon ; occupe beaucoup de transistors dans un UC (caches L1,L2,L3 = niveaux 1 à 3)

- Souvent des registres **associatifs** (ou à **contenu adressable**) ;

- A chaque registre est associé une *clé* et une *valeur* ;

- La recherche se fait par une comparaison **simultanée** d'une clé (de la valeur recherchée) avec **toutes** les clés enregistrées (opération faite par le matériel) ;

- En cas de succès, le dispositif émet la valeur associée à la clé retrouvée ;

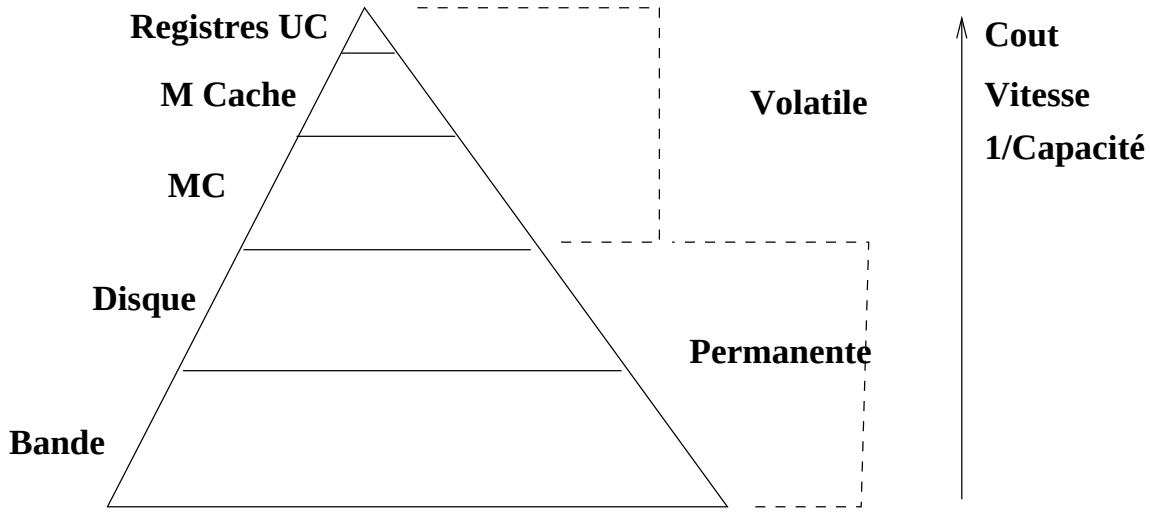


8.1.3 Mémoire Permanente (MS, de masse, auxiliaire, ...)

- Support magnétique (disque / bande / optique ...)

- Mémoire flash récente (une ROM = fausse mémoire !)

- Temps d'accès elevé mais le coût faible ;



type	taille O.	T. accès S.(2002)	coût relatif
$Cache$	10^3.. 10^6	500 MHz=2.10^{-9}=2 ns.et -	10
$M.C.$	10^6 ..10^9	100 MHz=10^{-8}=10 ns.et -	1
$Disque$	10^9 ..10^{11}	1 à 10 ms. et -	10^{-2} ..10^{-3}
$Bande$	10^8 ..10^{12}	10 à 100 ms. et -	10^{-4}

Remarque : ces valeurs varient fréquemment.

8.1.4 Représentation Logique et ϕ d'un programme

- Représentation Logique : procédures, modules, packages, librairies

- Représentation Physique : suite binaire

- Remarque : la MS ne fait que stocker les programmes, toute manipulation nécessite son chargement.

- Lors de sa création puis son exécution, un programme subit plusieurs représentations :

– Exécution → un processus (représentation en mémoire)

– → suspension (en mémoire, sur disque),

– → sauve de contexte,.. :

Ainsi, le processus (code + data) devient fichier sur disque (ou en mémoire) avec des représentation différentes.

- Un processus est tantôt (re)chargé, tantôt stocké sur disque ou en mémoire ;

- A chaque étape, le SE choisit et manipule une représentation donnée ;

- A la fin, le processus est détruit et sa représentation physique disparaît

- Le programmeur a une vision Logique, le SE assure la correspondance transparente ϕ et Logique

8.2 Objectifs d'une gestionnaire de mémoire

Organisation matérielle de la mémoire ϕ et sa gestion

8.2.1 Organisation de la mémoire

- La mémoire : un TDA = un ensemble de mots avec une adresse pour chacun
- Le SE structure la mémoire en **Zones** de taille N avec des asdresses consécutives ;
- La taille d'une zone peut être fixe ou variable
- Les zones reçoivent une représentation ϕ
- Le SE garantie :

1. **l'intégrité** des zones et
2. **protège** les zones des accès non autorisés ;
3. assure leur **partage** (sans duplication),
4. etc.

8.2.2 Gestion de la mémoire

- Stratégie d'**allocation** : attribution de zones ;

- **Libération** : à la fin/suspension d'un processus (PCB) ;

- La **destruction** d'un fichier (descripteur en mémoire),...

⊡ Opérateurs du TDA mémoire :

- **Mot** : lecture, écriture

- **Zone** : allocation, libération, extension de zone, division d'une zone, regroupement de zones, ...

⊡ Le SE doit fournir à l'utilisateur la quantité de mémoire demandée en l'affranchissant des **limites** physiques de la MC (Mémoire virtuelle) ;

- Les techniques de gestion sont classées en allocation **contiguë** et **non contiguë** ;

8.2.3 Méthodes d'allocation

1 - Placement de fichier (code+data) à des adresses consécutives (contigües) ;

2 - Placement de fichier à des adresses dispersées (non contigües) ;

- Allocation par bloc (> 1 mot / nbr. fixe de mots). Par exemple, blocs de 16/256/1024 mots

- Le nombre de mots dépend de la capacité matérielle à traiter un bloc comme un tout (ou un multiple de ce nombre)

- On alloue un nombre entier de blocs

8.2.4 Méthodes d'allocation contiguë

- L'espace mémoire divisé en zones de tailles variables

- Chaque zone contiguë contient un nbr entier de blocs

- Une zone est soit allouée soit libre

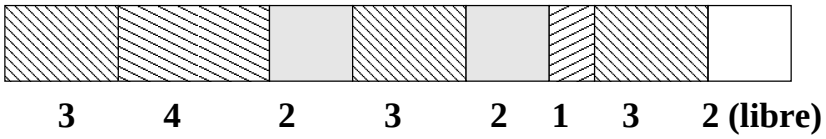
- Le SE maintient une table de zones libres

→ au départ, elle contient une seule zone (tout l'espace) ;

→ Quand une zone est allouée, elle est prélevée à l'extrémité d'une zone libre

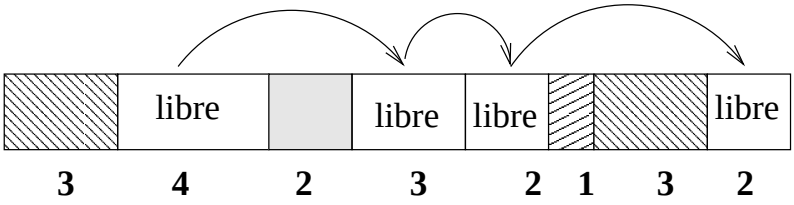
→ On diminue cette taille (ou supprime si tout est allouée)

Ex1 : un espace de 20 blocs après des allocations successives (avec des tailles 3,4,2,3,2,1,3) :



Si une zone est libérée, elle est ajoutée à *la liste des zones libres*.

Ex2 : On libère 3 zones allouées ci-dessus :



Problème : dans cette organisation, ceraines zones seront laissées inutilisées (**fragmentation**) . → Deux type de fragmentations : **interne** et **externe**.

Interne : On a une zone libre de taille 3 mais l'information fait 2,5. On alloue 3 mais tout n'est pas utilisé.

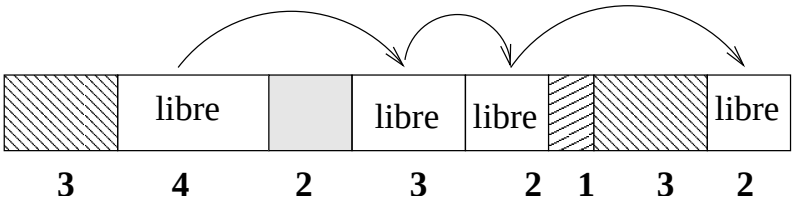
Le même cas arrive si les blocs sont de taille > un mot et que l'on veut placer un mot.

Externe : On a une zone libre de taille 3 mais la zone demandée est de taille 2 : le SE découpe en 2+1 et le bloc de

1 a peu de chance d'être demandé.

→ Pour la fragmentation Externe, le SE procède régulièrement à des **fusions** : fusion des zones allouées/libres.

Dans l'exemple :



Pour satisfaire une demande de 11 blocs, on peut fusionner de diverses façons (compactage au milieu, aux extrémités, ..).

Problèmes de compactage :

- Occupation de l'UC (surtout si très fréquent) ;

- Le déplacement des données nécessite du post-traitement pour les retrouver !

8.2.5 stratégies de placement

- Choix d'une zone libre

- Caractéristiques :

– La rapidité du choix de la zone

– La fragmentation externe (à minimiser)

– L'utilisation globale de l'espace (à maximiser)

Les 3 stratégies les plus utilisées :

1. **First-Fit** : première zone libre affectée

   → optimise le temps du choix

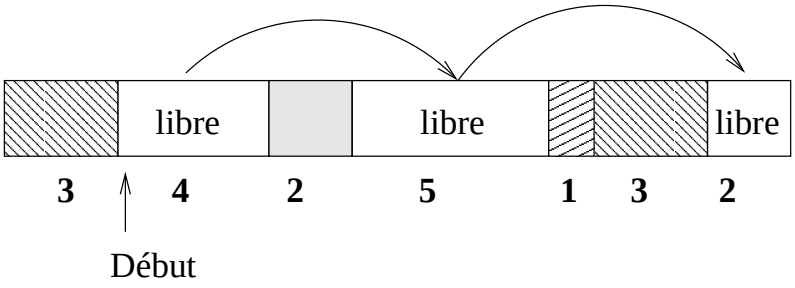
2. **best-Fit** : première zone libre de taille suffisante produisant un fragment minimum.

   → optimise l'utilisation globale mais crée des fragments ; elle est plus lente que la méthode précédente (parcours ou tri de la liste).

3. **worst-Fit** : choisir la zone la plus grande

   → les fragments crées ont plus de chance !

Exemple :



Demande d'un bloc.

1- First-fit : on laissera les zones libres 3,5,2

2- Best-fit : on laissera les zones libres 4,5,1

1- Worst-fit : on laissera les zones libres 4,4,2

Remarque : Ces stratégies sont également utilisées dans l'allocation de **disque**.

8.2.6 allocation non contiguë

- On fractionne les données à placer dans des blocs libres pas nécessairement adjacents.
- Supprime la fragmentation externe
- Evite les compactages et fusions
- Le SE conserve beaucoup d'information
- 2 méthodes : **chaînage** et **Indexation**.

Chaînage :

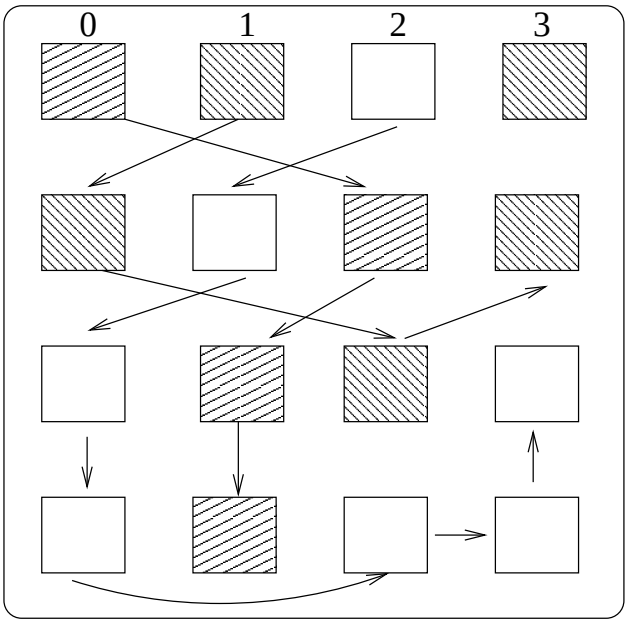


Table :	Début	Fin
	0	13
	1	3
	2	12

- Liste de blocs libres
- Blocs numérotés
- Une liste pour un ensemble de données
- Blocs chainés

Cette organisation a plusieurs inconvénients :

- Parcours séquentiel pour arriver à un bloc ;

- Une perte de lien = perte de plusieurs blocs (*double chaînage* pour éviter ces pertes) ;

- Place occupée par les pointeurs (dans les blocs)

Indexation :

- Une table de numéros de blocs utilisés pour un ensemble de données

- Evite la présence des pointeurs dans les blocs

- Accès direct possible à un bloc par son numéro ;

Incovénient : la table elle même peut être grande.

Exemple : pour l'exemple précédent et pour la première donnée, la table contiendra : 0,6,9,13

Remarques :

- Ces deux méthodes sont utilisées dans la gestion de la mémoire secondaire (disque) ;

- L'indexation est également utilise dans la gestion de la mémoire paginée.

8.3 Adressage

8.3.1 Adressage absolu

8.3.2 Adressage relatif

8.3.3 Translation d'adresses

Un programme est une suite d'instructions de base interprétables par un processeur.

Exemple :

N : entier := 0 ;

N := N+1 ;

afficher N ;

Supposons que la traduction de ces instructions donne en assembleur :

```
0000    move    #0 , 20     % mettre 0 dans N (déplacement +20)
0002    load    20 , R      % charger N (déplacement +20) dans R
0006    add     #1,  R      % ajouter la constate 1 au registre R
0010    store   R,   16     % sauvegarder R dans N
0014    push    16          % mettre N sur la pile
0016    call    affiche     % appeler affiche
0020    data    4           % place pour N (déplacement 20
```

Lors du chargement de ce programme, le code est placé à une certaine adresse mémoiré (appelée **Base** contenue dans un registre spécial appelé *Resistre Base*).

Lors du chargement du code, le SE procède à une translation d'adresses. Par exemple, si Base=2000, alors N sera

placée à 2000+20 = 2020.

Cette translation peut être faite dynamiquement lors de l'exécution de chauqe instruction ou statiquement.

Dans le mode dynamique, chaque translation a lieu lors de l'exécution.

Dans le mode statique, toutes les translations auront lieu au moment du chargement et le code sera placé à l'adresse Base (appelée aussi offset = début).

Le mode dynamique représente l'intérêt de permettre de recharger le code (après une commutaion de contexte) à n'importe quel endroit alors que dans le cas statique, le code doit être chargé à un endroit précis (ou bien recalculer toutes les adresses).