cour 05: gestion de la mémoire

I. **Gestion de la mémoire centrale :**

1. Introduction :

a.**Définitions :**

- **La gestion de mémoire:** est un aspect essentiel des systèmes informatiques qui concerne la manière dont un système d'exploitation ou un logiciel gère et utilise la mémoire disponible sur un périphérique informatique tel qu'un ordinateur.
- **l' objectif principal:**
    - est d'optimiser et d'organiser la mémoire système pour garantir que les programmes et les processus s'exécutent de manière efficace et sans conflits.
    - permettre aux processus de s'exécuter sans conflits
    - Utilisation efficace de la mémoire et de l ’UC
    - Bonnes performances (temps de réponse) pour les utilisateurs

b.**les Contraintes** :

- Système multiprocessus
- On suppose que la somme des tailles des processus est supérieure à la taille de la mémoire allouable
- La taille d’un processus est en général beaucoup plus petite que celle de la mémoire disponible
- Avant exécution, le code d’un processus est conservé dans un fichier sur disque.

c.**Différentes stratégies**

c.1 monoprogrammation :

- **La monoprogrammation:** "monotâche" est un modèle simple de gestion de processus dans un système informatique.
    - Dans ce schéma, un seul processus est exécuté à la fois et on lui attribue l'ensemble des ressources du système.
- **fonctionnement de la monoprogrammation :**
    1. **Un processus en mémoire à un instant donné :** Seul un processus est chargé en mémoire et est exécuté par l'unité centrale (CPU) à tout moment.
    2. **Chargement à la création du processus :** Lorsqu'un processus démarre, il est chargé en mémoire, et l'UC exécute ses instructions.

3. **Mémoire libérée à la fin du processus :** Une fois que le processus a terminé son exécution, la mémoire qu'il occupait est libérée pour être utilisée par d'autres processus.

- **Inconvénients de la monoprogrammation :**
    - **UC oisive pendant les chargements et les entrées/sorties :**
        - Lorsque le processus effectue des entrées/sorties ou lors du chargement d'un nouveau processus en mémoire, l'unité centrale est inactive.
    - **Mémoire mal utilisée :**
        - Puisqu'un seul processus est en mémoire à la fois, cela peut conduire à une utilisation inefficace de la mémoire.

  Ce modèle de gestion des processus est très basique et n'est pas efficace pour les systèmes où une utilisation optimale des ressources est requise. Les systèmes d'exploitation modernes utilisent souvent des modèles multitÂches (exécutant plusieurs processus simultanément) pour une utilisation plus efficace des ressources matérielles et une meilleure réactivité du système aux besoins des utilisateurs.

- **Utilisation des temps morts dus aux entrées- sorties lentes: réquisitionnament de la mémoire centrale**
    - La gestion des processus dans un système informatique implique souvent des stratégies pour gérer les processus bloqués ou inactifs, en particulier en raison d'opérations d'entrées/sorties lentes.

  L'une des méthodes pour optimiser l'utilisation de la mémoire centrale est de réaffecter l'espace mémoire aux processus prêts à s'exécuter, tout en stockant temporairement les processus bloqués sur un support de stockage plus lent, comme un disque dur, pour pouvoir les recharger ultérieurement.
    - Cette stratégie est souvent utilisée pour minimiser les temps morts dus aux opérations d'E/S.
    - les étapes clés de ce processus de réquisition de mémoire centrale :
        1. **Processus prêt à s'exécuter (éligible) :** Seul le processus actif et prêt à être exécuté est conservé en mémoire centrale. Cela garantit que l'UC peut immédiatement traiter ce processus, ce qui minimise le temps d'attente.
        2. **Processus bloqués :** Les processus en attente d'E/S ou d'autres ressources sont considérés comme bloqués. Plutôt que de conserver ces processus inactifs en mémoire centrale, ils sont déplacés vers un stockage à plus long terme, tel qu'un disque dur, pour libérer de l'espace en mémoire.
        3. **Stockage temporaire sur disque :** Les informations sur l'état de ces processus (contexte, données, etc.) sont stockées temporairement sur un support de stockage plus lent. Cela libère de la mémoire centrale pour d'autres processus actifs.
        4. **Rechargement ultérieur :** Lorsque les ressources nécessaires deviennent disponibles pour ces processus bloqués (par exemple, les données attendues sont arrivées), les

processus peuvent être rechargés en mémoire centrale pour reprise d'exécution.

Application : Temps d'écriture d'un processus :

- **Question :**
    - Combien des cycles CPU pour écrire un prcessus de 30Mo sur le disque dur (on donne latence : 5ms , débit : 100Mos/s , CPU 1GHZ) ?
- **Réponse :**

  Pour calculer le nombre de cycles CPU requis pour écrire un processus de 3 Mo sur le disque dur, nous pouvons diviser cette opération en deux parties : le temps nécessaire pour transférer les données du CPU au disque dur (en tenant compte de la latence et du débit) et le nombre de cycles du CPU pour cette opération.

  1. **Temps de transfert des données du CPU au disque dur :**

  Temps de latence = 5 ms = 0,005 secondes Taille du fichier = 3 Mo Débit = 100 Mo/s

  Temps nécessaire pour transférer 3 Mo à un débit de 100 Mo/s :

  Temps = Taille du fichier / Débit Temps = 3 Mo / 100 Mo/s = 0,03 secondes

  Le temps total pour transférer les données, en tenant compte de la latence et du débit, serait la somme de la latence et du temps de transfert réel :

  Temps total = Temps de latence + Temps de transfert Temps total = 0,005 s + 0,03 s = 0,035 secondes

  2. **Nombre de cycles CPU :**

  Fréquence du CPU = 1 GHz = 1 000 000 000 cycles par seconde

  Pour calculer le nombre de cycles CPU nécessaires pour cette opération, nous utilisons le temps total calculé précédemment :

  Nombre de cycles CPU = Temps total * Fréquence du CPU Nombre de cycles CPU = 0,035 s * 1 000 000 000 cycles/s Nombre de cycles CPU = 35 000 000 cycles

  Ainsi, pour écrire un processus de 3 Mo sur le disque dur, avec une latence de 5 ms, un débit de 100 Mo/s et une fréquence du CPU de 1 GHz, il faudrait environ 35 millions de cycles CPU.

c.2 multiprogrammation :

i.Définition et Caractéristiques :

- **La multiprogrammation** est une technique de gestion des processus dans les systèmes informatiques qui permet l'exécution simultanée de plusieurs processus en partageant la mémoire de manière efficace.
- Dans ce schéma, la mémoire est divisée en zones et chaque zone contient un processus. L'unité centrale (UC) est attribuée à un processus chargé et éligible pour l'exécution.

- Les principes clés de la multiprogrammation sont les suivants :

    1. **Division de la mémoire :** La mémoire est divisée en différentes zones ou partitions, et chaque partition peut contenir un processus distinct. Chaque processus a sa propre zone mémoire dédiée.

    2. **Exécution simultanée de processus :** Étant donné que plusieurs processus sont chargés en mémoire simultanément, l'UC peut passer d'un processus à un autre pour l'exécution, ce qui crée l'illusion d'une exécution simultanée. En réalité, l'UC exécute des instructions de différents processus en utilisant des techniques de commutation de contexte.

    3. **Allocation dynamique de l'UC :** L'UC est allouée à un processus éligible pour l'exécution, et cette allocation est gérée dynamiquement pour garantir que chaque processus a une part de temps d'exécution.

ii.les différentes types de la multiprogrammation :

- **Multiprogrammation sans réquisition :**

    - Dans le contexte de la multiprogrammation sans réquisition, les processus sont chargés en mémoire une seule fois et y restent jusqu'à leur achèvement.

    - Les principes clés de la multiprogrammation sans réquisition sont les suivants :

    1. **Chargement unique des processus :** Chaque processus est chargé en mémoire une seule fois au début de son exécution et reste en mémoire jusqu'à ce qu'il se termine.

    2. **Processus en mémoire jusqu'à la terminaison :** chaque processus reste en mémoire sans être retiré jusqu'à ce qu'il ait terminé son exécution.

    3. **L'UC peut être active pendant le chargement d'un nouveau processus :** L'unité centrale (UC) peut être active pendant le chargement d'un nouveau processus en mémoire, ce qui réduit le temps d'inactivité potentiel de l'UC.

    Bien que cette approche ait l'avantage de minimiser les temps de chargement et de déchargement de processus, elle peut également conduire à une utilisation moins efficace de la mémoire.

- **Multiprogrammation avec réquisition :**

    - Dans un environnement de multiprogrammation avec réquisition, une stratégie consiste à permettre le déchargement (vidage sur disque) des processus dont l'exécution a commencé.

    - Cela est souvent effectué pour gérer :

        - des entrées/sorties longues
        - ou pour exécuter un processus considéré comme prioritaire.

    - **le fonctionnement :**

        1. **Vidage sur disque :**

- Lorsqu'un processus est vidé sur disque, ses données (contexte, instructions, données en cours) sont stockées temporairement sur un support de stockage à plus long terme, comme un disque dur. Cela libère de la mémoire centrale pour d'autres processus actifs ou pour exécuter des processus jugés prioritaires.

2. **Rechargement des processus :**

   - Pour recharger un processus vidé en mémoire centrale, il est essentiel de maintenir une certaine cohérence dans le rechargement.
   - Généralement, un programme est "attaché" à une zone mémoire spécifique en raison des adresses absolues utilisées dans le programme.
   - Pour assurer l'isolation des zones, le rechargement des processus doit respecter ces limites d'adresses, afin de ne pas interférer avec d'autres processus ou écraser leurs données en mémoire.

RQ : **Le problème des adresses absolues lors de la réquisition de la mémoire centrale**

- Les adresses absolues sont des adresses mémoire spécifiques utilisées par un programme pour accéder à des emplacements mémoire particuliers. Si un programme est écrit en utilisant des adresses absolues et qu'il est attaché à une zone mémoire spécifique, recharger ce programme dans une autre zone pourrait poser des problèmes .
- Les solutions possibles incluent :
  - **Adressage virtuel :** Les systèmes d'exploitation utilisent des mécanismes d'adressage virtuel qui fournissent une couche d'abstraction entre les adresses logiques des programmes et les adresses physiques réelles de la mémoire, permettant une répartition dynamique de la mémoire sans affecter les adresses logiques des programmes.

2. **Mémoire virtuelle :**

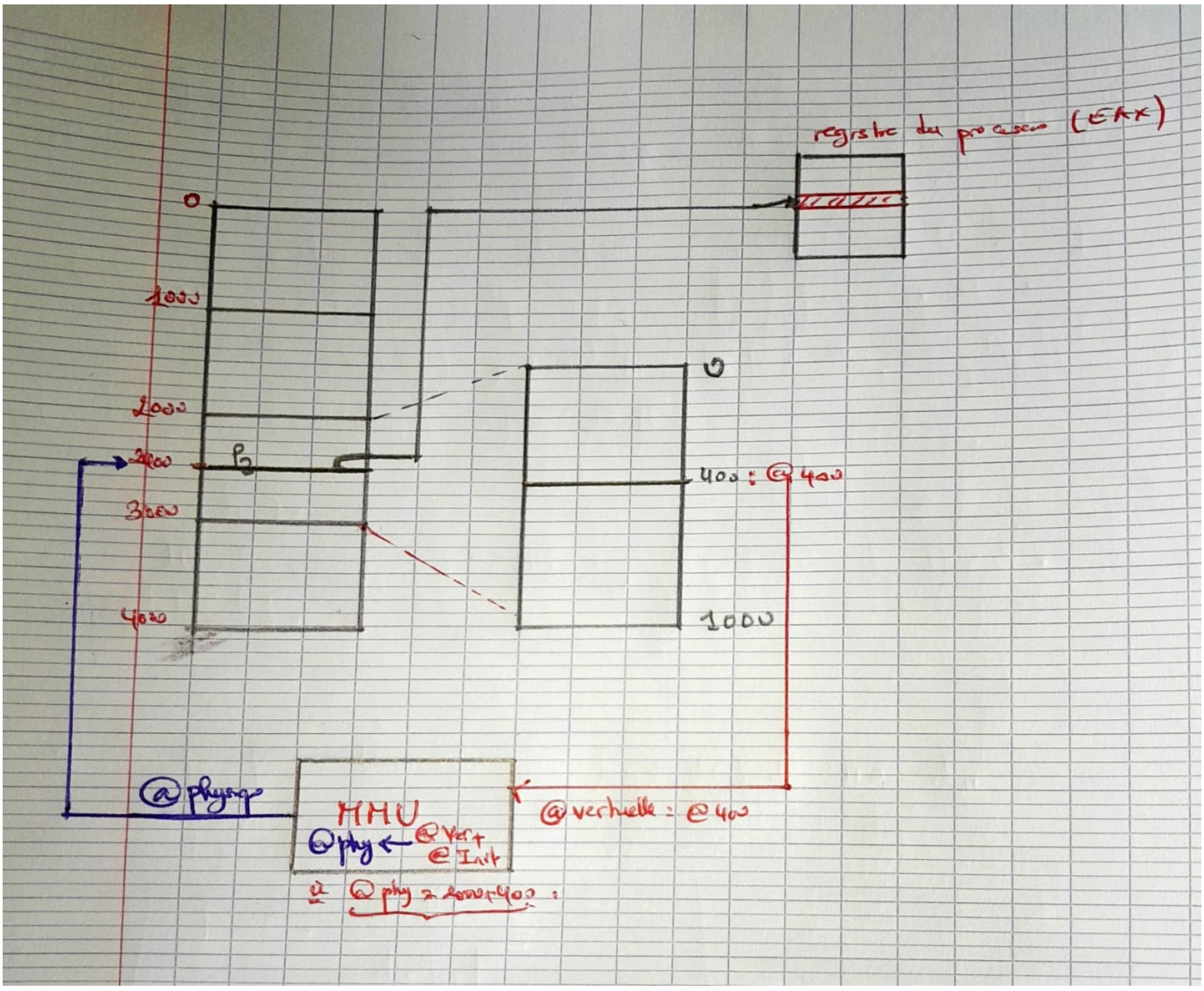
a. Définition :

- **La mémoire virtuelle:** est une technique utilisée par les systèmes d'exploitation pour permettre aux processus de s'exécuter de manière indépendante des zones physiques spécifiques de la mémoire.
- Cette approche vise à fournir une abstraction entre les adresses mémoire logiques utilisées par les processus et les adresses physiques réelles de la mémoire.
- **Objectif :**
  - L'objectif principal de la mémoire virtuelle est de permettre aux programmes de s'exécuter en utilisant des adresses mémoire logiques, indépendamment de la zone physique spécifique de la mémoire où ils sont stockés.
  - ce qui rendre un processus indépendant de la zone de mémoire physique où il s’exécute.

b. **MMU - Memory Management Unit :**

- **Mécanisme de traduction dynamique des adresses :**

- Pour mettre en œuvre la mémoire virtuelle, les systèmes d'exploitation utilisent un mécanisme câblé de traduction dynamique d'adresses, généralement géré par une unité de gestion de la mémoire (MMU - Memory Management Unit).
- L'unité de gestion de la mémoire (MMU) est responsable de traduire les adresses mémoire logiques utilisées par les programmes en adresses physiques réelles.
- La MMU utilise des tables de correspondance, souvent appelées tables de pages, pour associer les adresses logiques aux emplacements physiques dans la mémoire.
- Lorsque le processus accède à une adresse logique, la MMU traduit cette adresse en une adresse physique correspondante, permettant ainsi l'accès aux données correctes dans la mémoire physique.



RQ : Doit on charger en mémoire la totalité d ’un processus?

- C’est en général inutile.

3. Les problèmes de l’allocation de mémoire :

1. **Correspondance entre adresses virtuelles et physiques :**

- Les systèmes utilisent la mémoire virtuelle pour fournir aux processus des adresses logiques indépendantes des adresses physiques réelles.
- Toutefois, les contraintes matérielles peuvent limiter la manière dont cette correspondance est réalisée
- Les contraintes matérielles, comme la taille de l'espace d'adressage virtuel par rapport à la mémoire physique disponible, influencent la gestion de cette correspondance.

2. **Gestion de la mémoire physique :**
   - La gestion de la mémoire physique est un défi majeur.
   - La MMU (Memory Management Unit) gère la correspondance entre les adresses virtuelles et les emplacements physiques dans la mémoire.
   - La fragmentation de la mémoire, où l'espace libre est divisé en petits blocs non contigus, peut affecter l'efficacité de l'allocation et de la libération de la mémoire.
3. **Protection :**
   - La protection est cruciale pour garantir l'isolation des processus.
   - Les mécanismes de protection sont nécessaires pour empêcher qu'un processus puisse accéder ou modifier une zone de mémoire qui ne lui est pas affectée. Cela est souvent mis en œuvre via des bits de protection associés à des pages de mémoire pour limiter les accès non autorisés(le droit d'accées).
4. **Partage de mémoire :**
   - Permettre le partage de certaines zones de mémoire entre plusieurs processus est une considération importante.
   - Le partage réduit la duplication de données et optimise l'utilisation de la mémoire. Cependant, cela exige une gestion précise pour éviter les conflits et garantir l'intégrité des données partagées.

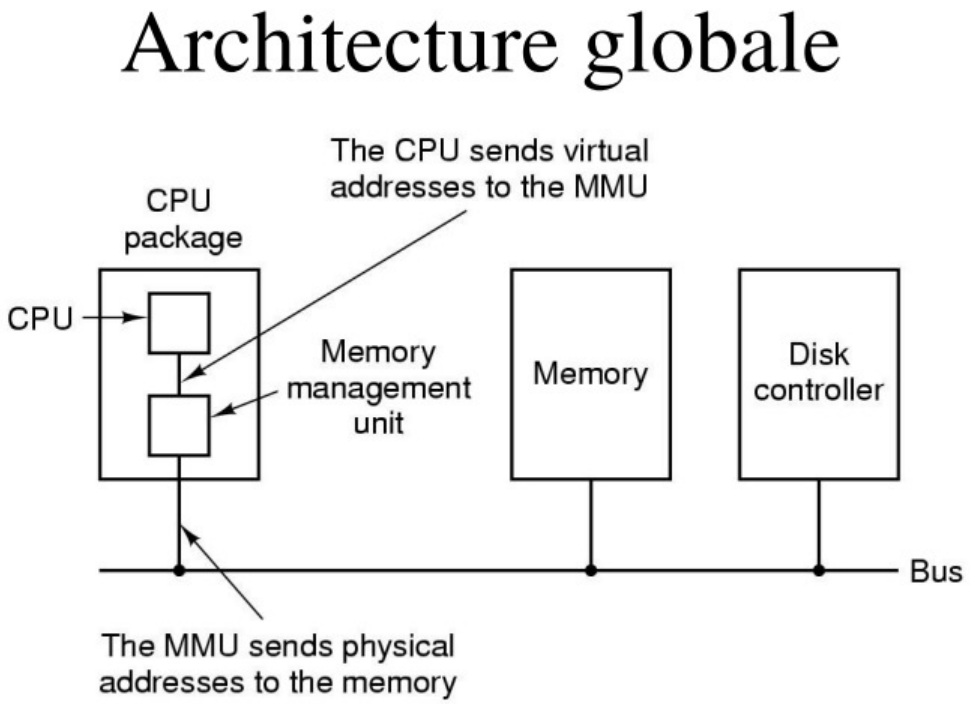
II. Gestion de la mémoire par zones :Segmantation (TP1)

III. Gestion de la mémoire par pages :Pagination

1. Définition :

- La pagination est une technique de gestion de la mémoire virtuelle utilisée dans les systèmes d'exploitation pour organiser et gérer l'allocation de la mémoire :
  - Elle divise *la mémoire virtuelle* en zones de taille fixe appelées **pages** ,
  - Elle divise *le mémoire physique* en blocs de même taille que **les pages** appelés **les cases**.

- **Principe :**
  - Une page peut être chargée dans n'importe quelle case.
  - Chaque page de la mémoire virtuelle est associée à un case correspondant dans la mémoire physique. Cette correspondance est gérée par la MMU (Unité de Gestion de la Mémoire).



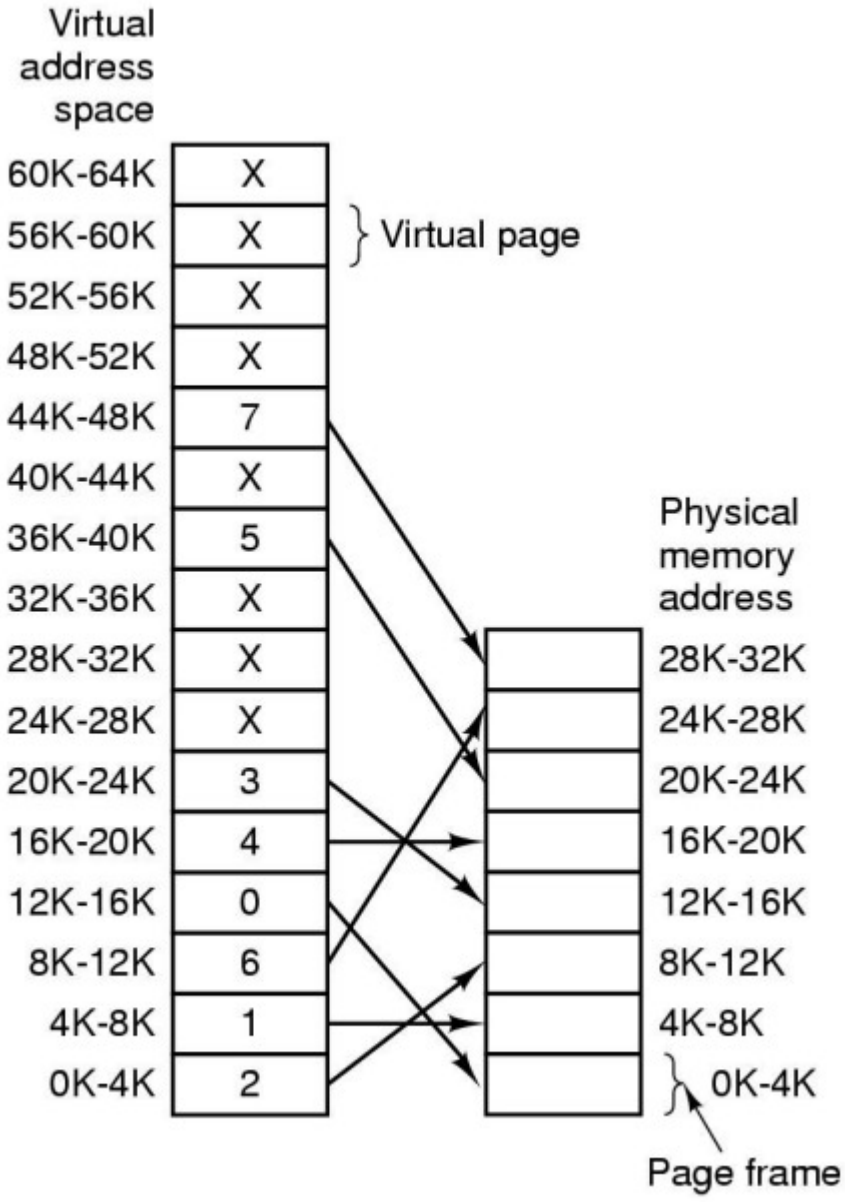
2. **les tables de page :**

2.1 **Table de pages simple :**

- **les tables de pages** sont utilisées pour suivre et gérer la correspondance entre les adresses mémoire virtuelles et physiques. Ces tables sont généralement organisées hiérarchiquement pour permettre un accès efficace aux pages mémoire.
- En réalité : une table de pages est utilisée pour stocker la traduction `des numéros de pages virtuelles` vers `les numéros de cases physiques` correspondants dans la mémoire physique :
  - **Numéro de page virtuelle :** C'est l'index ou le numéro de la page dans l'espace d'adressage virtuel du processus.
  - **Numéro de case physique :** C'est l'index ou le numéro de la case ou du cadre dans la mémoire physique, où réside actuellement la page mémoire
- Lorsqu'un processus accède à une adresse mémoire virtuelle, la MMU utilise la table de page pour traduire cette adresse en une adresse physique en localisant l'entrée correspondante dans la table de pages:
  - En utilisant le numéro de page virtuelle, la MMU trouve l'entrée associée dans la table de pages,

    - récupère le numéro de case physique correspondant, et utilise cette information pour accéder à l'adresse mémoire physique réelle.

- Ainsi, la table de pages joue un rôle crucial dans la traduction des adresses mémoire virtuelles en adresses physiques en fournissant une correspondance rapide et efficace entre les numéros de pages virtuelles et les numéros de cadres physiques dans la mémoire physique.



2.2 **Tables de pages structurées en arbre :**

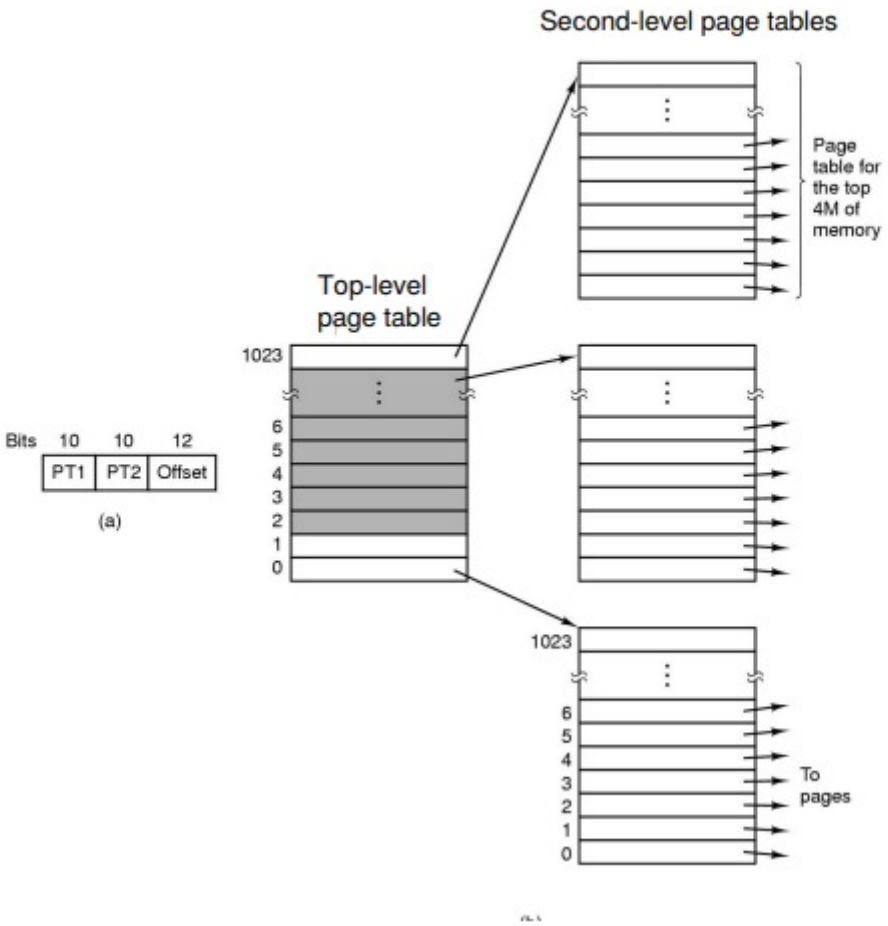
- Les tables de pages structurées en arbre, souvent appelées **pagination à n niveaux** sont des techniques de gestion de la mémoire virtuelle utilisées pour réduire la complexité et économiser de l'espace dans la traduction des adresses virtuelles en adresses physiques.

- **Pagination à 2 niveaux :**

    - Dans ce schéma, la table de pages est organisée hiérarchiquement en deux niveaux.

    - Le premier niveau contient une table de page , Chaque entrée dans cette table de niveau 1 pointe vers une table de pages de niveau 2.

- Le deuxième niveau contient les entrées réelles de la table de pages, où chaque entrée est associée à une adresse physique.



- **Exemple:**

  Dans un schéma de pagination à deux niveaux avec des adresses virtuelles sur 32 bits et des pages de 4 Ko (4096 octets), chaque niveau de la table de pages aura une structure pour traduire l'adresse virtuelle en adresse physique.

  1. **Taille d'une adresse virtuelle sur 32 bits :** Chaque adresse virtuelle est composée de 32 bits. Pour un système de pagination sur deux niveaux, la pagination est souvent divisée en deux parties pour chaque niveau.

  2. **Taille de l'offset de page :** Avec des pages de 4 Ko, l'offset d'une page (déplacement à l'intérieur de la page) est déterminé par le nombre de bits nécessaires pour adresser les 4096 octets dans une page.

  $(\log_2(4096) = 12)$ bits sont nécessaires pour l'offset de la page.

  Pour un exemple simple, considérons une adresse virtuelle de 32 bits avec une pagination sur deux niveaux :

  - **Premier niveau de pagination :** Il peut consister en 10 bits pour l'index du premier niveau.
  - **Deuxième niveau de pagination :** Puis, 10 bits pour l'index du deuxième niveau.
  - **Offset de page :** Les 12 bits restants pour l'offset de la page.

Ainsi, pour traduire une adresse virtuelle en adresse physique :

- Les 10 premiers bits (premier niveau) sont utilisés pour accéder à une entrée dans la table de pages du premier niveau.
- Ces 10 bits pointent vers une table de pages de deuxième niveau.

Dans cette table de pages de deuxième niveau :

- Les 10 bits suivants (de 11 à 20, par exemple) sont utilisés pour accéder à une entrée spécifique dans cette table.
- Cette entrée indique l'adresse physique de la page.

- Ces approches hiérarchiques permettent de diviser la table de pages en plusieurs niveaux, ce qui réduit le nombre total d'entrées à parcourir pour traduire une adresse virtuelle en adresse physique. Au lieu de parcourir une table de pages unique pour une traduction directe, le système suit une séquence hiérarchique, en commençant par le premier niveau pour localiser progressivement l'entrée correspondante.

3. **fonctionnement de la MMU :**

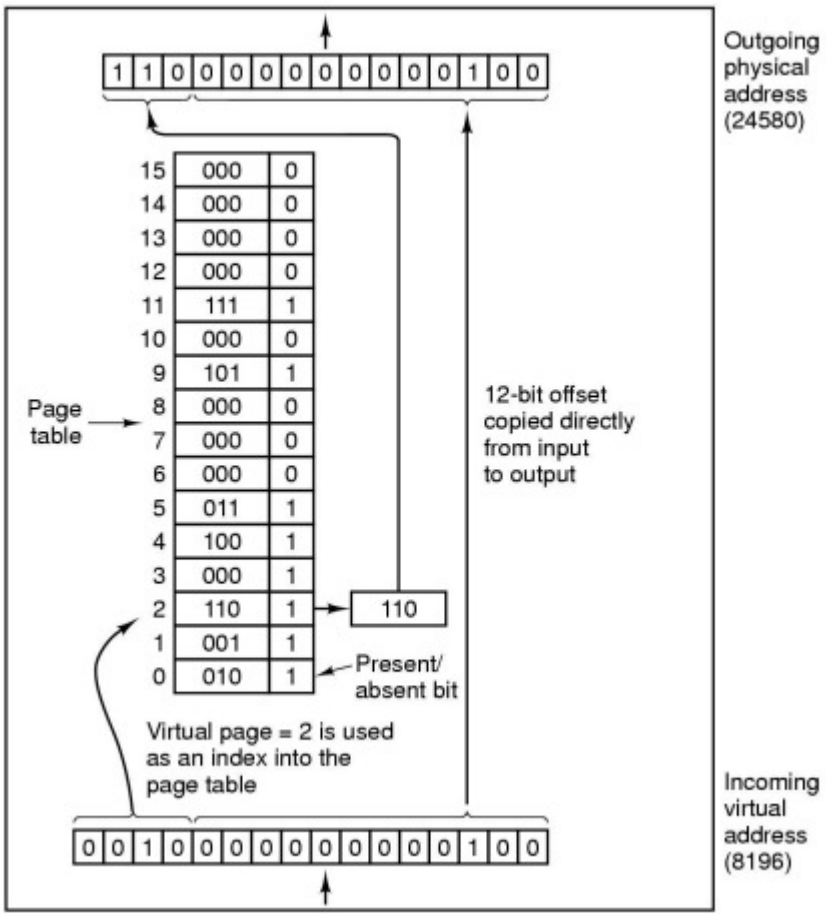
3.1 **Traduction du @vertuille vers @physique :**

- Dans un système de mémoire virtuelle avec pagination, la traduction des adresses virtuelles en adresses physiques se fait via la MMU (Memory Management Unit) en utilisant les tables de pages.
- Pour illustrer le processus de traduction, prenons un exemple simplifié :
  - Supposons un système où :
    - Les adresses virtuelles sont sur 12 bits.
    - Les adresses physiques sont sur 12 bits.
    - La taille des pages est 16 octets.
  - Considérons que la table de pages a les entrées suivantes :
    - Page 0 est en case 4
    - Page 1 est en case 8
    - Page 2 est en case 2
    - Page 3 est en case 11
- un processus utilise une adresse virtuelle `0011 1101 0101` ,voici comment la traduction pourrait se faire :

1. **Séparation de l'adresse :**
   - L'adresse 3D5 est sur 12 bits, où les premiers 4 bits (les bits de poids fort) représentent le numéro de page, et les 8 bits restants représentent le décalage dans la page.
2. **Extraction du numéro de page :** Le numéro de page est 0011 (3 en décimal) et le décalage est 1101 0101 (D5 en hexadécimal).
3. **Consultation de la table de pages :**

- La MMU consulte la table de pages pour traduire le numéro de page en adresse physique.
- D'après la table de pages, la page 3 (numéro de page 0011) est située dans le case 11.

4. **Combinaison avec le décalage :**
   - L'adresse physique est alors construite en combinant le numéro de case (11) avec le décalage (D5) qui se trouve dans la page.
   - L'adresse physique correspondante est donc `1011 1101 0101`.



RQ 1 :

- **Question :**
  - Dans un mémoire vertuielle : 16 pages chacun de taille 4KB
  - le nombre de bit de l'offset ?
- **Réponse :** Dans une mémoire virtuelle avec 16 pages, chacune de taille 4 KB, le nombre de bits de l'offset est déterminé par la taille de la page. Étant donné que chaque page a une taille de 4 KB (kilooctets), convertissons cela en bits pour déterminer la taille de l'offset.

  1 KB (kilooctet) = 1024 octets (car 1 octet = 8 bits)

  Donc, 4 KB = 4 * 1024 octets = 4096 octets

Pour connaître le nombre de bits nécessaires pour adresser 4096 octets (4 KB), vous pouvez utiliser le logarithme en base 2. Le résultat nous donne le nombre de bits nécessaires pour adresser chaque octet dans une page.

log2(4096) ≈ 12 bits

Cela signifie qu'un offset de 12 bits est nécessaire pour adresser les emplacements spécifiques à l'intérieur d'une page de 4 KB.

RQ 2 :

- vous pouvez exprimer la conversion de l'adresse virtuelle vers l'adresse physique à l'aide de cette équation :

```
@p = T(m-n) + @V
```

Où :

- `@p` : est l'adresse mémoire physique en décimal .
- `T` est la taille de la page ex : T = 4KB = 4 * 1024 = 4096 .
- `m` est le numéro de bloc associé à l'adresse physique.
- `n` est le numéro de bloc associé à l'adresse virtuelle.
- `@V` est l'adresse virtuelle.
- Cette formule exprime la traduction de l'adresse virtuelle vers l'adresse physique.

3.2. **Comment le MMU fait il la traduction :**

- La traduction d'adresses virtuelles en adresses physiques est effectuée par la MMU (Unité de Gestion de la Mémoire).
- Il existe plusieurs méthodes pour réaliser cette traduction, chacune ayant ses avantages et inconvénients.
- Parmi les approches couramment utilisées, on retrouve l'utilisation de :
    - **registres intégrés**,
    - **de tables de pages en mémoire centrale** et
    - **l'utilisation de registres associatifs**.
- **une explication de ces méthodes :**
    - **Registres intégrés (mémoire topographique) :**
        - Certains MMU utilisent des registres internes pour stocker des traductions d'adresses récentes.

        - Ces registres contiennent les correspondances entre certaines adresses virtuelles et leurs adresses physiques respectives.
        - Cela permet une traduction rapide pour les adresses fréquemment utilisées, évitant ainsi de consulter les tables de pages.
    - **Table de pages en mémoire centrale :**
        - La méthode la plus courante consiste à stocker les tables de pages en mémoire centrale.
        - Ces tables contiennent les traductions des adresses virtuelles en adresses physiques. Lorsqu'une adresse virtuelle est à traduire, la MMU consulte ces tables pour trouver la correspondance et obtenir l'adresse physique associée.
    - **Les registres associatifs** :
        - sont des composants spécifiques utilisés dans les Unités de Gestion de la Mémoire (MMU) pour accélérer la traduction des adresses virtuelles en adresses physiques.
        - Les registres associatifs sont des structures matérielles spéciales intégrées à la MMU. Ces registres permettent de stocker temporairement un certain nombre d'entrées de la table de pages.
        - Lorsqu'une adresse virtuelle doit être traduite, la MMU vérifie d'abord si l'entrée correspondante est présente dans les registres associatifs. Si l'entrée est trouvée dans ces registres, la traduction peut être effectuée rapidement sans accéder à la table de pages en mémoire centrale. Cela permet de gagner du temps en évitant un accès supplémentaire à la mémoire.
        - **Cache de traduction :** En effet, ces registres associatifs peuvent être vus comme une forme de cache de traduction, où les traductions d'adresses virtuelles en adresses physiques fréquemment utilisées sont temporairement stockées pour un accès plus rapide.

4.**page fault :**

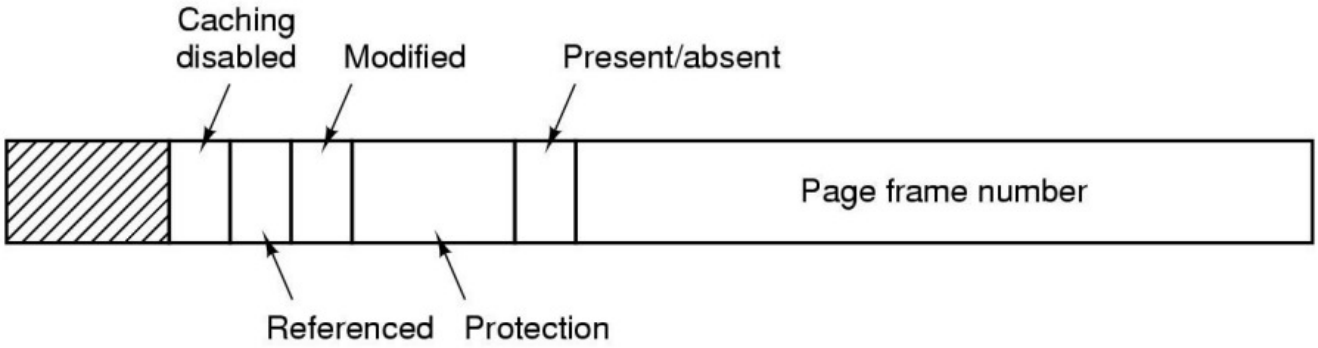
- lorsqu'un accès mémoire est effectué à une page qui n'est pas actuellement chargée en mémoire physique, une situation appelée **faute de page** (page fault) se produit.
- Cela se produit lorsque le Memory Management Unit (MMU) ne peut pas traduire l'adresse virtuelle entrée en une adresse physique correspondante, généralement parce que la page mémoire en question n'est pas actuellement présente dans la mémoire physique.
- **Traitement d' une faute de page :**
    1. **Interruption du processus :** Lorsqu'un processus tente d'accéder à une page mémoire qui n'est pas en mémoire physique, cela déclenche une exception matérielle appelée **faute de page**.
    2. **Traitement de la faute de page :**

    - Le système d'exploitation intervient pour gérer cette **faute de page**. Le traitement de la faute de page implique généralement :
        - **Chargement de la page absente :** Le système d'exploitation charge la page demandée depuis le stockage de masse (disque dur, SSD) vers un case disponible en mémoire physique.
        - **Mise à jour des tables de pages :** Une fois la page chargée, le système d'exploitation met à jour les tables de pages pour refléter la correspondance entre l'adresse virtuelle et l'adresse physique de la page nouvellement chargée.
        - **Reprise de l'exécution du processus :** Une fois la page chargée en mémoire, le processus est relancé et peut accéder à la page mémoire comme prévu.
- La gestion des fautes de page permet d'optimiser l'utilisation de la mémoire. Plutôt que de charger toutes les pages d'un processus en mémoire dès le début (ce qui serait inefficace), le système n'apporte en mémoire que les pages nécessaires lorsqu'elles sont demandées. Cela permet d'économiser de l'espace mémoire et de ne charger en mémoire que ce qui est effectivement utilisé, en utilisant le stockage de masse pour stocker les autres pages jusqu'à ce qu'elles soient nécessaires.

5. **informations associer à chaque page :**

- Dans un système de gestion de mémoire virtuelle, chaque page virtuelle peut être associée à diverses informations pour gérer sa présence en mémoire et ses droits d'accès.
- Voici les informations couramment associées à chaque page :

    1. **Numéro de case physique :** Cela indique l'adresse de la case (ou cadre) en mémoire physique où la page est chargée. C'est la correspondance entre la page virtuelle et son emplacement dans la mémoire physique.
    2. **Bit de présence (bit P) :** Ce bit indique si la page est actuellement présente en mémoire physique. Lorsqu'il est à 1, cela indique que la page est chargée en mémoire. Si ce bit est à 0, cela indique que la page est actuellement absente de la mémoire physique et nécessitera un chargement à partir du stockage de masse lors de son prochain accès.
    3. **Droits d'accès :** Ces informations définissent les permissions ou les droits accordés pour cette page en termes de **lecture**, **écriture** et **exécution**. Ces paramètres déterminent si la page peut être lue, écrite ou exécutée.
    4. **Bit de modification :** Ce bit indique si la page a été modifiée depuis son chargement en mémoire. Lorsqu'une page est modifiée, ce bit est activé. Cela peut être utilisé pour la gestion des remplacements de pages et pour déterminer si une sauvegarde est nécessaire avant de libérer la page de la mémoire.



- **TLB :**
  - Le terme "TLB" signifie "Translation Lookaside Buffer" en anglais, traduit en français par "Mémoire Associative de Traduction" ou "Registre de Traduction d'Adresses".
  - Le TLB est une mémoire cache spéciale utilisée dans les systèmes de gestion de mémoire virtuelle pour accélérer le processus de traduction d'adresses virtuelles en adresses physiques. Il fonctionne en associant les adresses virtuelles à leurs adresses physiques correspondantes.
  - Le TLB conserve les traductions récentes ou fréquemment utilisées d'adresses virtuelles en adresses physiques. Il agit comme une mémoire cache pour ces traductions.

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

A TLB to speed up paging

6. **pagination et processus :**

6.1 **Généralités sur la mise en œuvre :**

- **Espace virtuel pour chaque processus :**
  - Chaque processus en cours d'exécution dispose de son propre espace d'adressage virtuel(sa propre table de pages).

- Cet espace lui permet d'accéder à une mémoire virtuelle plus grande que la mémoire physique réelle disponible.

- **Gestion des défauts de page :**
  - Lorsqu'un processus accède à une page qui n'est pas actuellement chargée en mémoire physique, cela génère ce qu'on appelle un "défaut de page" (ou "page fault"). En cas de défaut de page,
  - le système doit allouer une case (un cadre) en mémoire pour cette page et transférer la page du stockage de masse (disque dur, SSD) vers cette case mémoire nouvellement allouée.

- **Gestion de la mémoire pleine :**
  - Si la mémoire est pleine, et qu'aucune case mémoire supplémentaire n'est disponible pour accueillir de nouvelles pages, un algorithme de remplacement entre en jeu.
  - Cet algorithme sélectionne une page à retirer de la mémoire pour faire de la place à la nouvelle page. La page sélectionnée est généralement déchargée vers le stockage de masse pour libérer de l'espace.

RQ :

- Ce système est efficace lorsque les défauts de page sont rares.

6.2 **Comportement des programmes :**

- Le comportement des programmes, en termes d'accès à la mémoire, a une importance cruciale dans la gestion efficace de la mémoire virtuelle.
- Voici des éléments importants concernant ce comportement :

1. **L'exécution d'un programme:**
   - L'exécution d'un programme consiste en une séquence d'accès à des pages mémoire (soi de type `exécution`,`lécture` ou `écriture`).
   - Les programmes accèdent à différentes parties de leur espace d'adressage virtuel au fil de leur exécution.
2. **Distribution des références de pages :**
   - Les programmes ont tendance à accéder à des pages différentes tout au long de leur exécution.
   - Certains modèles de comportement peuvent montrer des accès fréquents à certaines pages (appelé localité).
   - tandis que d'autres affichent une distribution plus uniforme des accès.
   - l'étude de distribution des références de pages *ie* : l'étude de l'ensmbles de pages qui seront accéedes pendant tout la durée de vie de notre programme permet d'optimiser la gestion de mémoire vertuielle sur tout dans le cas de mémoire restreinte .

3. **Comportement en mémoire restreinte :**
    - Lorsque la mémoire disponible est limitée, les pages actuellement en mémoire peuvent être remplacées par de nouvelles pages si nécessaire.
    - Un défi majeur est de prédire quelles pages sont susceptibles d'être utilisées dans un proche avenir pour optimiser les performances en minimisant les défauts de page.
4. **Prévision des pages à utiliser dans le futur proche :**
    - Différents algorithmes de remplacement de page tentent de prédire les pages qui seront les plus pertinentes à garder en mémoire.
    - Des stratégies telles que *le principe de localité*, qui suppose que les pages récemment utilisées ont plus de chances d'être utilisées à nouveau, sont souvent utilisées pour optimiser cette prédiction.

6.3 **Algorithmes de remplacement :**

a. Propriété de localité :

- La propriété de localité est un concept clé en informatique, particulièrement dans le contexte de la gestion de la mémoire, et se divise en deux types principaux :
    - **Localité temporelle :** Cette propriété suggère que si une donnée est référencée récemment, il est probable qu'elle soit à nouveau référencée dans un futur proche.
    - **Localité spatiale :** Cette propriété suppose que si une donnée est référencée, il est probable que des données proches dans l'espace mémoire soient également référencées. En d'autres termes, les données situées à proximité de celles récemment référencées ont une forte probabilité d'être accédées également.
- Dans le contexte de la gestion de la mémoire, la localité temporelle est particulièrement utile pour prédire les futures références de pages mémoire. Si une page a été récemment référencée, la probabilité qu'elle soit requise à nouveau dans un futur proche est élevée, basée sur le principe de localité temporelle.
- En utilisant cette propriété de localité, **les algorithmes de remplacement** de pages, tels que l'algorithme de remplacement *LRU* (Least Recently Used), peuvent prédire les pages à garder en mémoire. Ils favorisent le maintien des pages les plus récemment utilisées en mémoire pour minimiser les défauts de page, car ces pages ont une forte probabilité d'être nécessaires à nouveau dans un court laps de temps, en accord avec la localité temporelle.

b. working set :

- L'ensemble de travail (ou "working set" en anglais) est un concept utilisé pour décrire le comportement d'accès à la mémoire par un processus à un instant donné.
- Il est défini comme l'ensemble des pages mémoire référencées par un processus dans un intervalle de temps donné.

- La représentation formelle de l'ensemble de travail est notée `W(t, T)`, où :
    - `t` représente un instant de temps spécifique.
    - `T` est la fenêtre de temps considérée pour définir cet ensemble de travail.
    - Ainsi, `W(t, T)` est l'ensemble des pages mémoire référencées par un processus entre l'instant `t-T` et l'instant `t`.
- **Proposition :**
    - `W(t,T)` est une bonne approximation de `W(t+T,T)` .

C. Algorithmes de remplacement :

- **Algorithme FIFO (First-In-First-Out) :**
    - Les pages sont placées dans une file d'attente (FIFO).
    - Lorsqu'une page doit être remplacée, c'est celle qui est en tête de file qui est retirée.
- **Algorithme RAND (Random Replacement) :**
    - Cet algorithme choisit aléatoirement une page à retirer de la mémoire.
    - Remplacement d ’une case choisie au hasard
- **Algorithme de la seconde chance :**
    - C'est une amélioration de l'algorithme FIFO. Il utilise un bit (bit R) pour marquer la récence de l'utilisation de chaque page.
    - L'algorithme fonctionne comme FIFO, mais il ne remplace pas la première page dans la file d'attente qui a son bit R activé.
    - Si une page a son bit R activé, cela signifie qu'elle a été utilisée récemment et elle reçoit une "seconde chance" avant d'être remplacée.
- **Algorithme LRU (Least Recently Used) :**
    - LRU remplace la page qui n'a pas été utilisée depuis le plus longtemps.
    - Il se base sur le principe de la localité temporelle, en supposant que les pages utilisées récemment auront plus de chances d'être utilisées à nouveau. Ainsi, il remplace la page qui a été la moins récemment utilisée.

RQ : **Remplacement Cas de plusieurs processus:local ou global**

en cas des plusiéres processus, on deux façons d'effectuer le replacement :

- **Remplacement global:**
    - L ’algorithme de remplacement est exécuté sur l ’ensemble des cases de mémoire

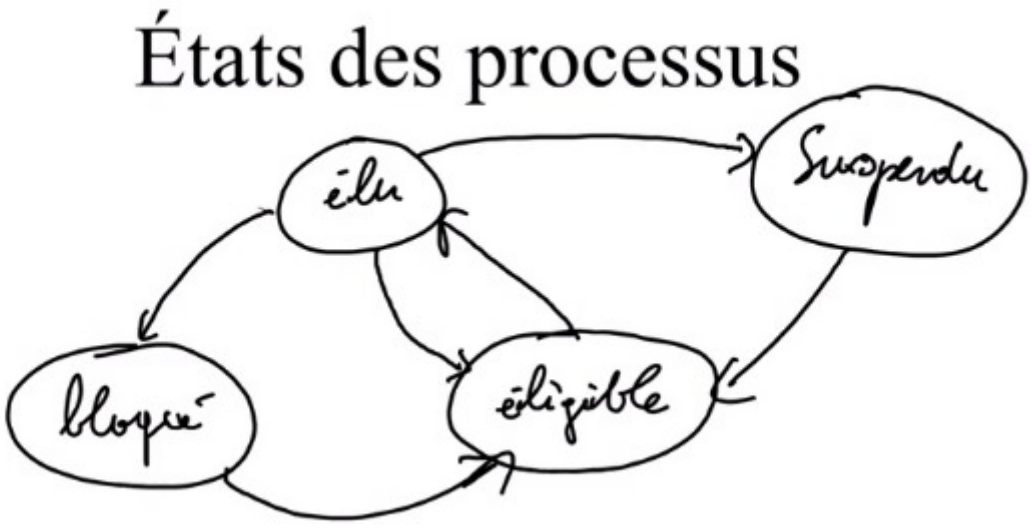
    - On peut alors voler une case à un processus en attente de page
    - **Avantage:** utilisation efficace du mémoire .
    - **inconviennt :** cette opération est très coûteuse

- **Remplacement local:**
    - L ’algorithme de remplacement ne concerne que les cases affectées au processus courant
    - **Avantage:** opération moins coûteuse .
    - **inconviennt :** utilisation inéfficace de mémoire .

7. **Architecture globale d’un système paginé:**

7.1 **Hypothéses simpliste :**

- Un espace virtuel par processus (une table de pages)
- Les pages des processus bloqués sont conservées sur le disque dur .
- Chargement des pages à la demande
- Lorsqu’un processus attend une page, il est placé dans un état dit `Suspendu` (donc on introduit un nouveau état du processus)



- Un algorithme de remplacement détermine quelle case à vider
- Structures de données associées aux pages:
    - Table des pages avec adresse disque et numéro de case
- Structures de données associées aux cases :
    - Table des cases

7.2 **le phénomène d’écroulement `thrashing`:**

- **Définition:**
    - Le phénomène de thrashing, ou "écroulement"se produit lorsqu'un système informatique passe une grande partie de son temps à échanger des pages entre la mémoire principale (RAM) et le stockage de masse (généralement un disque dur) plutôt que d'exécuter des tâches informatiques réelles.
    - Cela conduit à une diminution significative des performances du système.
    - Le thrashing se produit lorsque le système d'exploitation (OS) est incapable de maintenir suffisamment de pages actives en mémoire pour satisfaire les besoins des processus en cours d'exécution. Les pages sont constamment échangées entre la mémoire principale et le stockage de masse, créant une charge excessive sur le système et réduisant considérablement ses performances.
- **Les principales raisons du thrashing :**
    1. **Insuffisance de mémoire physique :** Lorsque la mémoire physique disponible est insuffisante pour répondre aux besoins des processus en cours d'exécution, le système doit constamment charger et décharger des pages entre la mémoire principale et le stockage de masse.
    2. **Taille inappropriée des pages :** Si la taille des pages utilisées dans la pagination est trop petite, cela peut augmenter la probabilité de thrashing car un plus grand nombre de pages doivent être gérées.
    3. **Planification inadéquate des processus :** Si la planification des processus n'est pas optimale et que trop de processus actifs sont en cours d'exécution simultanément, cela peut entraîner une utilisation inefficace de la mémoire.

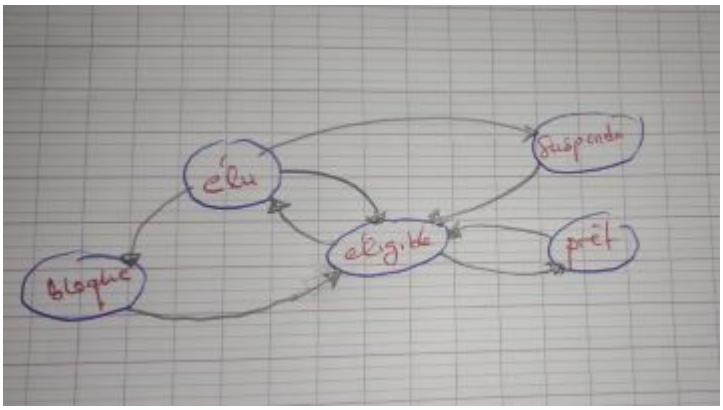
Pour atténuer le thrashing, les systèmes d'exploitation peuvent mettre en œuvre des techniques telles que la gestion intelligente de la mémoire, l'optimisation de la taille des pages, la priorisation des processus, et la détection précoce du thrashing pour prendre des mesures correctives.

RQ:

- en cas de phénoméne d'écoulement toutes les processus sont dans l'état `suspendu` .

7.3 **Gestion globale de la mémoire : solution du phénomène d'écroulement `thrashing`:**

1. **réduisant le nombre de processus admis en mémoire :**
    - Pour éviter le thrashing, il peut être nécessaire de limiter le nombre de processus actifs en mémoire en même temps. Si trop de processus sont admis en mémoire simultanément, cela peut conduire à une utilisation excessive de la mémoire et favoriser le thrashing.
    - **Nouvel état pour les processus : prêt :**
        - Lorsqu'un processus n'est pas actuellement exécuté en raison de la pénurie de mémoire, il peut être placé dans un état `prêt`. Cela signifie que le processus est prêt à être exécuté dès qu'il y aura suffisamment de mémoire disponible pour lui.



2. **Méthode de régulation :**

   - La gestion globale par régulation est une approche utilisée dans la gestion de la mémoire pour maintenir l'équilibre entre le nombre de processus actifs en mémoire et les ressources du système.

   - **Principe :**

     - La gestion globale par régulation vise à surveiller en permanence l'état du système, en particulier le niveau d'occupation de la mémoire, l'activité du processeur, et le taux d'utilisation du disque.

     - En fonction de ces paramètres, le système prend des décisions pour éviter la surcharge (thrashing) ou la sous-charge de la mémoire.

   - **États du système :**

     1. **Système surchargé :**
     - Lorsque la mémoire est surchargée, le système d'exploitation peut ne pas être en mesure de répondre aux demandes des processus de manière efficace. Le thrashing peut se produire, ce qui conduit à une diminution significative des performances.
     - Pour remédier à cela, le système peut décider de chasser un ou plusieurs processus de la mémoire pour libérer de l'espace.
     2. **Système sous-chargé :**
     - Si la mémoire a une capacité non utilisée significative et que des processus sont prêts à s'exécuter, le système peut décider d'admettre en mémoire un processus prêt pour améliorer l'utilisation des ressources.
     3. **Système normal :**
     - Lorsque la mémoire est correctement utilisée, le système fonctionne efficacement et répond aux demandes des processus sans compromettre les performances.

   - **Actions en cas de surcharge :**

     - Si le système est surchargé, le système d'exploitation peut décider de libérer de la mémoire en chassant un ou plusieurs processus hors de la mémoire principale. Cela peut être fait en utilisant des algorithmes de remplacement de pages.

   - **Actions en cas de sous-charge :**

        - Si le système est sous-chargé et qu'il y a des processus prêts à s'exécuter, le système peut décider d'admettre en mémoire un processus prêt pour optimiser l'utilisation des ressources.

    - **Surveillance des indicateurs :**

        - La gestion globale par régulation implique la surveillance constante d'indicateurs tels que :

            - **le taux d'utilisation du disque**,

            - **activité de l ’UC .**

3. **Gestion fondée sur l ’espace de travail:**

- **Principe :**

    - La gestion fondée sur l'espace de travail est une approche de gestion de la mémoire qui tient compte du comportement individuel de chaque processus afin d'optimiser l'utilisation de la mémoire.

    - **Admission en mémoire :** Un processus `prêt` ne sera admis en mémoire que si le système peut lui réserver un nombre de cases mémoire égal à la taille de son espace de travail `W(t,T)` .L'espace de travail d'un processus fait référence à la quantité de mémoire dont il a besoin pour s'exécuter de manière efficace.

        - **Remplacement local :** Plutôt que de remplacer une page entière lors du remplacement de pages, le remplacement local implique de ne remplacer que les parties de la mémoire associées au processus qui quitte la mémoire. Cela réduit la quantité d'informations évincées et minimise le risque de thrashing.

    - **Préchargement en option :**

        - La gestion basée sur l'espace de travail peut ou non être accompagnée de préchargement. Le préchargement consiste à anticiper les besoins futurs en mémoire d'un processus et à charger en mémoire certaines pages avant qu'elles ne soient effectivement nécessaires. Cela vise à améliorer l'efficacité de l'accès à la mémoire.