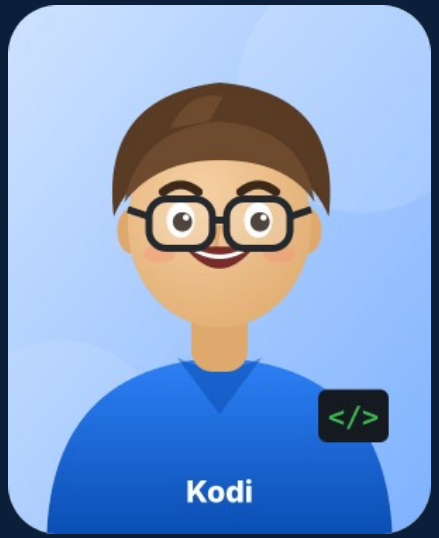
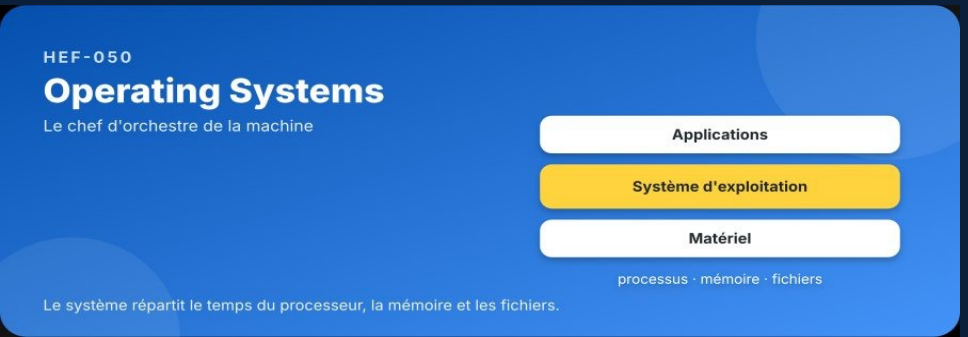
HEF-050 · MISSION 05 / 10

La mémoire virtuelle

Chaque programme croit posséder toute la mémoire.
Le noyau lui fait croire cette belle histoire.



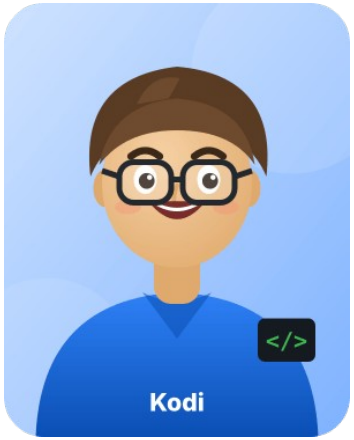


Badge : Cartographe de la mémoire

80 min · intermédiaire

L'histoire

Kodi lance un programme qui réserve 16 Go de mémoire sur une machine qui n'en a que 8. Il n'est pas tué tout de suite, mais la machine devient atrocement lente.



> *« Le système a utilisé le disque comme mémoire de secours, dit Sam. Chaque programme vit dans un espace d'adresses virtuel que le noyau relie à la mémoire réelle, à la demande. »*

Tu comprends comment le noyau gère la mémoire de tous les processus, et pourquoi la saturation ralentit tout.

Ta mission

À la fin, je serai capable de…

1. expliquer l'adresse virtuelle et l'adresse physique
2. décrire pages, cadres et table des pages
3. expliquer le défaut de page et le swap
4. simuler les politiques de remplacement FIFO et LRU
5. expliquer pourquoi la mémoire virtuelle protège les processus

Défi d'échauffement

Défi d'échauffement

Un hôtel a 100 chambres réelles, mais chaque client reçoit un plan indiquant « chambre 1 à 1 000 ». Un employé à la réception relie chaque numéro de plan à une vraie chambre, ou au garde-meuble si la chambre n'est pas prête. Quels sont les avantages pour le client ? Pour l'hôtel ? Que se passe-t-il si trop de clients demandent leurs vraies chambres en même temps ?

Le concept

L'idée

Chaque processus travaille avec des **adresses virtuelles**. Le noyau et un composant matériel (la **MMU**) traduisent chaque adresse virtuelle en **adresse physique** (dans la vraie RAM).
Avantages :

- **Isolation** : un processus ne voit que sa propre mémoire.
- **Simplicité** : chaque programme croit disposer d'un espace continu.
- **Partage** : des pages physiques communes (bibliothèques) peuvent apparaître dans plusieurs processus.
- **Plus que la RAM** : on peut utiliser le disque comme extension.

Le concept

Pages et cadres

La mémoire est découpée en blocs de taille fixe (souvent **4 Ko**) :

- une **page** pour la mémoire virtuelle ;
- un **cadre** (*frame*) pour la mémoire physique.

La **table des pages** donne, pour chaque page d'un processus, le cadre où elle se trouve (ou « absente »).

```
adresse virtuelle = numéro de page + décalage dans la page

page 0 → cadre 5
page 1 → cadre 2
page 2 → (sur le disque)
```

Le concept

Pages et cadres (suite)

Avec des pages de 4 096 octets, l'adresse virtuelle 8 200 correspond à la page 2 (8 200 // 4 096) et au décalage 8 (8 200 % 4 096).

La TLB

Consulter la table des pages à chaque accès serait coûteux. La **TLB** (*Translation Lookaside Buffer*) est un petit cache de traductions récentes dans le processeur.

Le défaut de page

Quand un programme accède à une page qui n'est pas en RAM, c'est un **défaut de page** (*page fault*) :

1. le CPU lève une exception ;
2. le noyau trouve la page (sur le disque, dans le **swap** ou dans le fichier du programme) ;

Le concept

Le défaut de page (suite)

3. il trouve un cadre libre, ou en libère un (remplacement) ;
4. il charge la page, met à jour la table, et le programme reprend.

Un défaut de page **majeur** (lecture du disque) coûte des **milliers de fois** plus qu'un accès normal.

Le remplacement de pages

Quand il n'y a plus de cadre libre, quelle page expulser ?

Politique	Principe
FIFO	la plus ancienne arrivée
LRU	la moins récemment utilisée
Optimal	celle qui ne servira pas avant le plus longtemps (théorique)

Le concept

Le thrashing

Quand la mémoire est trop petite pour les programmes en cours, le système passe plus de temps à échanger des pages qu'à travailler : c'est le **thrashing** (écroulement). La machine « rame » alors énormément.

Simuler le remplacement

```
def simuler(references, cadres, politique="FIFO"):
    memoire, defauts = [], 0
    for page in references:
        if page in memoire:
            if politique == "LRU":
                memoire.remove(page)
                memoire.append(page)
            continue
        defauts += 1
        if len(memoire) == cadres:
```

Le concept

Simuler le remplacement (suite)

```
            memoire.pop(0)              # FIFO : le plus ancien ; LRU : le moins ré
    cent
         memoire.append(page)
     return defauts

print(simuler([1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5], 3, "FIFO"))    # 9
print(simuler([1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5], 3, "LRU"))     # 10
```

Exemple pas à pas

Problème : traduire des adresses virtuelles avec une table des pages, et compter les défauts de page.

```
TAILLE_PAGE = 4096
table = {0: 5, 1: 2}            # page -> cadre (la page 2 n'est pas en mémoire)

def traduire(adresse_virtuelle):
    page, decalage = divmod(adresse_virtuelle, TAILLE_PAGE)
    if page not in table:
        return f"défaut de page (page {page})"
    return table[page] * TAILLE_PAGE + decalage

for a in (100, 4200, 8300):
    print(a, "→", traduire(a))
# 100 → 20580
# 4200 → 8296
# 8300 → défaut de page (page 2)
```

Exemple pas à pas (suite)

Calcul : l'adresse 100 est dans la page 0 (cadre 5) au décalage 100 : 5 × 4 096 + 100 = 20 580. L'adresse 4 200 est dans la page 1 (cadre 2) au décalage 104 : 2 × 4 096 + 104 = 8 296.
Anomalie de Belady : avec FIFO, **ajouter** de la mémoire peut augmenter les défauts !

```
def fifo(refs, cadres):
    memoire, defauts = [], 0
    for p in refs:
        if p not in memoire:
            defauts += 1
            if len(memoire) == cadres:
                memoire.pop(0)
            memoire.append(p)
    return defauts

refs = [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]
print([fifo(refs, k) for k in (3, 4)])     # [9, 10]
```

Exemple pas à pas (suite)

Avec 3 cadres : 9 défauts. Avec 4 cadres : **10** défauts. LRU n'a pas ce défaut.

À toi, avec aide

La séquence de pages demandées est `7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2` avec 3 cadres.

1. Compte les défauts de page en FIFO à la main, puis vérifie avec un programme.
2. Même question avec LRU.
3. Quelle politique est meilleure ici ? Pourquoi ?

Indices progressifs

Indice 1

Dessine la mémoire après chaque accès.

Indice 2

En FIFO, on expulse la page arrivée le plus tôt, même si elle vient d'être utilisée.

Indice 3

En LRU, la page 0 est souvent réutilisée : elle reste.

Réponse

```
def simuler(refs, cadres, lru):
    memoire, defauts = [], 0
    for p in refs:
        if p in memoire:
            if lru:
                memoire.remove(p); memoire.append(p)
            continue
        defauts += 1
        if len(memoire) == cadres:
            memoire.pop(0)
        memoire.append(p)
    return defauts

refs = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]
print("FIFO:", simuler(refs, 3, False), "LRU:", simuler(refs, 3, True))   # FIFO:
   10 LRU: 9
```

Réponse (suite)

LRU est légèrement meilleur : la page 0, utilisée souvent, y reste ; FIFO l'expulse à cause de son ancienneté.

À toi, en solo

Simule le **thrashing**. Trois processus ont chacun un « ensemble de travail » de 4 pages qu'ils parcourent en boucle. Avec un total de 12, 9 puis 6 cadres, compte les défauts de page (politique LRU) en alternant les accès des trois processus.

Trace la courbe défauts = f(cadres). À partir de quel nombre de cadres le thrashing disparaît-il ?

À toi, en solo · réponse

Pistes de correction

```
def lru(refs, cadres):
    memoire, defauts = [], 0
    for p in refs:
        if p in memoire:
            memoire.remove(p); memoire.append(p)
            continue
        defauts += 1
        if len(memoire) == cadres:
            memoire.pop(0)
        memoire.append(p)
    return defauts

# trois processus (P, page) qui tournent en boucle sur 4 pages chacun
refs = []
for _ in range(50):
    for page in range(4):
        for proc in range(3):
            refs.append((proc, page))
```

À toi, en solo · réponse

Pistes de correction (suite)

```
for cadres in (12, 9, 6):
    print(cadres, "cadres :", lru(refs, cadres), "défauts sur", len(refs), "accès"
   )
```

Avec 12 cadres (l'ensemble de travail total = 12 pages), il n'y a que 12 défauts initiaux. En dessous, la boucle cyclique avec LRU provoque des défauts à presque chaque accès : c'est le thrashing.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Confondre adresse virtuelle et physique	Erreur de raisonnement sur les pointeurs	Toujours préciser laquelle
Ignorer le coût d'un défaut de page majeur	Programme figé	Surveiller l'usage du swap
Allouer plus que la RAM	Thrashing, machine lente	Dimensionner la mémoire ou limiter les processus
Croire que le swap est gratuit	Un SSD reste ~1 000 fois plus lent que la RAM	Éviter d'en dépendre

Mini-boss

Mini-boss

Gestionnaire de mémoire virtuelle. Écris une classe `MMU` avec :

- une table des pages par processus ;
- `acceder(pid, adresse_virtuelle)` qui traduit l'adresse, gère le défaut de page et le remplacement LRU global ;
- des statistiques : accès, défauts, taux de défauts ;
- une protection : un processus ne peut pas accéder à l'adresse virtuelle d'un autre (même numéro de page, cadres différents).

Teste avec deux processus qui utilisent les mêmes numéros de page et un nombre de cadres limité.

Mini-boss · réponse

Pistes de correction

```
class MMU:
    def __init__(self, cadres, taille_page=4096):
        self.cadres, self.taille = cadres, taille_page
        self.tables = {}           # pid -> {page: cadre}
        self.lru = []              # liste de (pid, page), la plus ancienne en têt
   e
        self.libres = list(range(cadres))
        self.acces = self.defauts = 0

    def acceder(self, pid, adresse):
        self.acces += 1
        page, decalage = divmod(adresse, self.taille)
        table = self.tables.setdefault(pid, {})
        cle = (pid, page)
        if page in table:
            self.lru.remove(cle); self.lru.append(cle)
        else:
            self.defauts += 1
            if self.libres:
```

Mini-boss · réponse

Pistes de correction (suite)

```
                cadre = self.libres.pop()
            else:
                pid_v, page_v = self.lru.pop(0)
                cadre = self.tables[pid_v].pop(page_v)
            table[page] = cadre
            self.lru.append(cle)
        return table[page] * self.taille + decalage

m = MMU(cadres=4)
a1 = m.acceder(1, 100)
a2 = m.acceder(2, 100)         # même adresse virtuelle, autre cadre physique
assert a1 != a2
for i in range(10):
    m.acceder(1, (i % 6) * 4096)
print(m.acces, m.defauts, round(m.defauts / m.acces, 2))
```

Deux processus utilisant la même adresse virtuelle obtiennent des adresses physiques différentes : c'est l'isolation.

Quiz éclair

Quiz éclair

1. La table des pages sert à :

A. stocker les fichiers
B. chiffrer la mémoire
C. traduire les adresses virtuelles en adresses physiques

2. Un défaut de page survient quand :

A. la page demandée n'est pas en RAM
B. le disque est plein
C. le CPU surchauffe

3. Le thrashing est :

A. un virus

Quiz éclair

Quiz éclair (suite)

B. un système qui passe son temps à échanger des pages
C. un type de cache

4. 🔁 *Révision mission 04* : Round Robin donne :

A. le CPU au plus court
B. le CPU au plus prioritaire
C. à chaque processus le CPU à tour de rôle pour un quantum

5. 🔁 *Révision mission 03* : Les threads d'un même processus partagent :

A. la mémoire
B. la pile
C. les registres

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

Je réfléchis

1. Quel équivalent du « garde-meuble » (swap) utilises-tu pour ranger ce qui n'est pas utile tout de suite ?
2. Pourquoi l'isolation de la mémoire est-elle une base de la sécurité ?

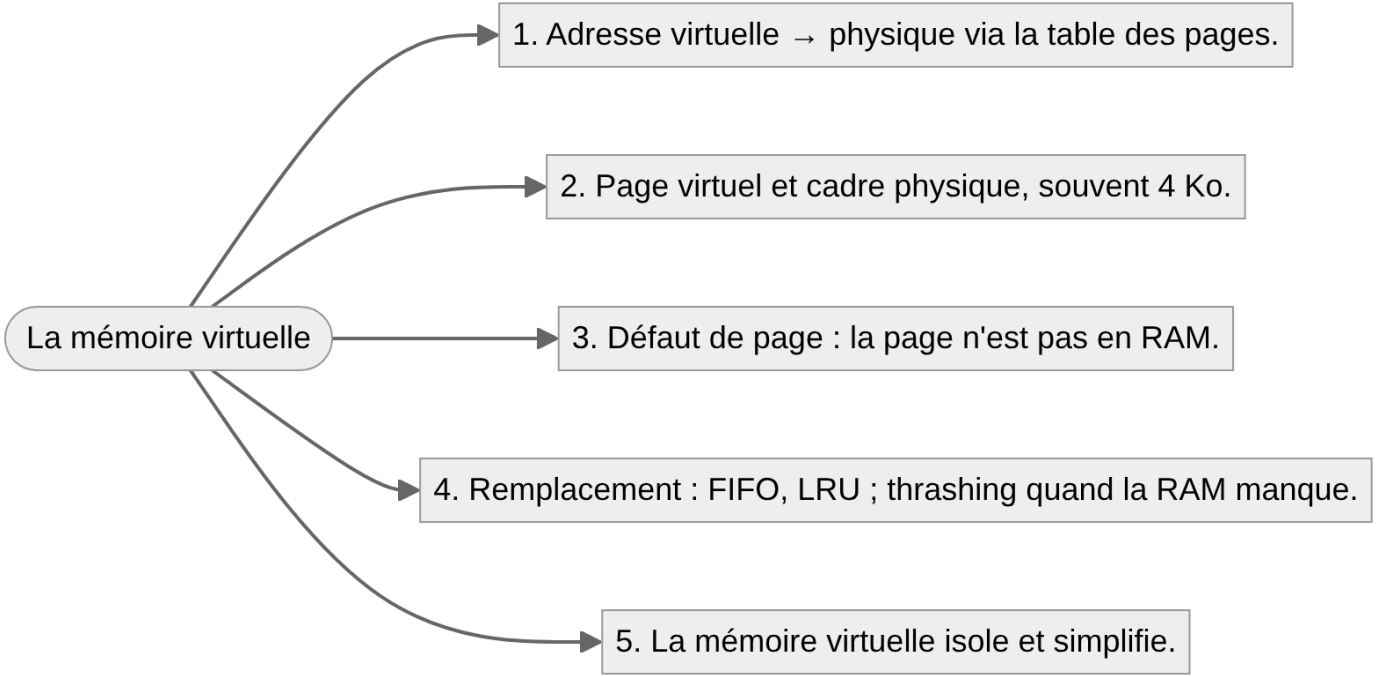
À retenir

À retenir

1. Adresse virtuelle → physique via la table des pages.
2. Page (virtuel) et cadre (physique), souvent 4 Ko.
3. Défaut de page : la page n'est pas en RAM.
4. Remplacement : FIFO, LRU ; thrashing quand la RAM manque.
5. La mémoire virtuelle isole et simplifie.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Cartographe de la mémoire

Prochaine mission : Les systèmes de fichiers

