

R3.05 - Programmation Système

Mémoire

Cyril Grelier

Université de Lorraine - IUT de Metz



- Dans un OS multi-tâche et multi-utilisateur
⇒ mémoire partagée entre :
 - les **processus** (navigateur internet, jeu, IDE, serveur web, ...)
 - le **noyau**
- Besoin de gérer efficacement la mémoire et l'isoler entre les processus
- OS fournit des **adresses virtuelles** et non les **adresses physiques**.
- Bénéfices :
 - **Isolation** des processus.
 - **Flexibilité** (déplacement/relocation transparents).
 - **Optimisation** (code partagé entre processus).
 - **Swap** (programmes plus gros que la RAM).

Hiérarchie mémoire (ordre de grandeur)

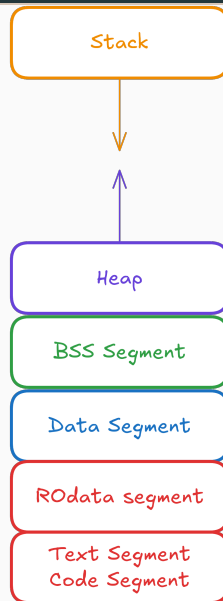
Type de mémoire	Vitesse d'accès	Taille	Coût	Remarques
CPU (Ryzen 7 / i9)	–	–	~ 550 €	n'inclut pas que les caches
Registres	~ 0,3 ns	~ 2-4 Ko	inclus CPU	plus rapide
Cache L1	~ 1 ns	640 Ko	inclus CPU	–
Cache L2	3–5 ns	8 Mo	inclus CPU	–
Cache L3	10–15 ns	96 Mo	inclus CPU	–
RAM (DDR5)	50–100 ns	64 Go	~ 200 €	–
SSD NVMe	50–100 µs	2 To	~ 150 €	–
HDD	5–10 ms	4 To	~ 130 €	plus lent
Swap (sur disque)	HDD/SSD	–	–	–

- Plus près du CPU ⇒ **rapide, petit et cher.**
- Plus loin du CPU ⇒ **lent, gros et peu cher.**

- **Pagination** (Linux) : mémoire découpée en **pages** (souvent 4 Ko)
- **Demand paging** : pages chargées à la première utilisation, tout n'est pas chargé en mémoire d'un coup (page fault normale)
- **Segmentation** : historique ; aujourd'hui surtout pagination
- **MMU** (*Memory Management Unit*) :
traduit **adresse virtuelle** → **adresse physique** via la **table des pages**
- **TLB** (*Translation Lookaside Buffer*) :
cache de traductions virtuel→physique (hit rapide, miss coûteux)
- Accès interdit ⇒ SIGSEGV (segmentation fault)
- **Swap** : extension de la RAM sur le disque (lent) ; trop de swap ⇒ *thrashing*

Espace d'adressage d'un processus

- **Text** : code exécutable (r-x).
- **Rodata** : constantes (r-).
- **Data** : globales/statics *initialisées* (rw-).
- **BSS** : globales/statics *non initialisées* (rw-, zérotées).
- **Heap** : allocation dynamique (malloc/free).
- **Stack** : variables locales, cadres d'appels (LIFO).
- + **VMA** (*Virtual Memory Areas*) diverses :
mmap, bibliothèques partagées, TLS, I/O mappées, ...



Observer la mémoire : /proc/self/maps

```
1  $ cat /proc/self/maps
2  5f481c31f000-5f481c321000 r--p 00000000 fc:01 11797247      /usr/bin/cat
3  5f481c321000-5f481c325000 r-xp 00002000 fc:01 11797247      /usr/bin/cat <- text
4  5f481c325000-5f481c327000 r--p 00006000 fc:01 11797247      /usr/bin/cat <- rodata
5  5f481c327000-5f481c328000 r--p 00007000 fc:01 11797247      /usr/bin/cat
6  5f481c328000-5f481c329000 rw-p 00008000 fc:01 11797247      /usr/bin/cat <- data et BSS
7  5f483e114000-5f483e135000 rw-p 00000000 00:00 0             [heap]          <- heap
8  73d384200000-73d3849f0000 r--p 00000000 fc:01 11799968      /usr/lib/locale/locale-archive
9  73d384a00000-73d384a28000 r--p 00000000 fc:01 11813669      /usr/lib/x86_64-linux-gnu/libc.so.6
10 73d384a28000-73d384bbd000 r-xp 00028000 fc:01 11813669      /usr/lib/x86_64-linux-gnu/libc.so.6
11 ...                                                         <- lib partagées/mmap
12 73d384c1c000-73d384c29000 rw-p 00000000 00:00 0
13 73d384c71000-73d384c96000 rw-p 00000000 00:00 0
14 73d384cb0000-73d384cb2000 rw-p 00000000 00:00 0
15 73d384cb2000-73d384cb4000 r--p 00000000 fc:01 11808014      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
16 ...
17 7ffdba237000-7ffdba258000 rw-p 00000000 00:00 0             [stack]         <- stack
18 7ffdba388000-7ffdba38c000 r--p 00000000 00:00 0             [vvar]          <- variables noyau
19 7ffdba38c000-7ffdba38e000 r-xp 00000000 00:00 0             [vdso]          <- lib noyau
20 ffffffff600000-ffffffff601000 --xp 00000000 00:00 0          [vsyscall]      <- ancien mécanisme
```

r = read, w = write, x = exec, - = none, p = private, s = shared

Formes d'allocation

Automatique (stack)

```
1 void f(void){  
2     int local = 0; // stack - pile  
3 } // libérée en sortie
```

Dynamique (heap)

```
1 int *p = malloc(100*sizeof *p); // heap - tas  
2 /* ... */  
3 free(p); p = NULL;
```

Statique (data/bss, variables globales)

```
1 // en dehors du main ou fonction  
2 int g; // BSS (zéro)  
3 int h = 42; // Data (init)
```

Autres allocations

- mmap fichiers/gros blocs anonymes
- posix_memalign, aligned_alloc (SIMD)
- TLS (*Thread-Local Storage*) mémoire par thread

Stack overflow (ex.)

```
1 void f(int n){ f(n+1); } // pas d'arrêt  
2 int main(void){ f(0); } // SIGSEGV
```

valgrind ou ASan pour diagnostiquer.

Performances d'accès (sans optimisations du compilateur)

Test d'incrémenter 10 000 000 000 fois un long int avec des variables allouées différemment :

Type d'allocation	Adresse	Temps
Automatique (stack)	0x7ffd7d2c1c50	6.8 s
Dynamique (heap)	0x64749b2132a0	8.5 s
Statique locale	0x647470a26020	30.9 s
Globale	0x647470a26018	30.8 s

- **Stack** ⇒ très rapide (offset relatif au pointeur de pile, souvent en L1 / registres).
- **Heap** ⇒ un poil plus lent (déréférencement de pointeur).
- **Globales/Statique** ⇒ souvent relues depuis la mémoire (ne peuvent pas toujours rester en registre car le compilateur ne peut pas savoir si elles sont modifiées ailleurs).

Le compilateur, les optimisations et les caches influencent beaucoup les résultats.

Erreurs fréquentes

- *Memory leak* (oubli de free).
- *Invalid access, buffer over/underflow*.
- *Double free, use-after-free*.
- *Dangling pointer, memory corruption*.
- *Data race* (concurrency).

Bonnes pratiques

- Toujours free après usage.
- Mettre le pointeur à NULL après free/init.
- Vérifier les retours (malloc, mmap).
- Outils :
 - valgrind
 - ASan (AddressSanitizer)
 - UBSan (UndefinedBehaviorSanitizer)