Mémoire : du bit à l'abstraction vue par les processus

Motivation 1 Comprendre comment le matériel stocke l'information et la manipule.

I. [CSAPP] Stockage de l'information dans le matériel

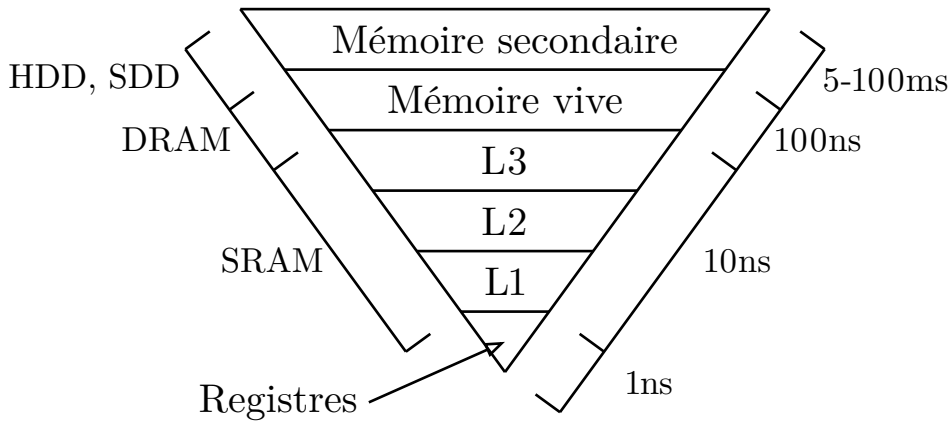
A. Du matériel au bit

Définition 2 **Bit** Un ordinateur utilise des données binaires, puisque les circuits logiques utilisés ne possèdent que deux états : alimenté ou non-alimenté. On appelle **bit** un chiffre binaire (de valeur 0 ou 1) représentant l'absence ou la présence de courant.

Définition 3 **Octet** En général, dans un ordinateur les informations sont stockés par séquences de 8-bits, appelées **octets** (byte en anglais).

Définition 4 **Hiérarchie mémoire** Un ordinateur actuel utilise plusieurs types de mémoire :

- La **mémoire secondaire** (disque dur, SSD, ...) contient la majorité des données stockées dans l'ordinateur, elle est lente mais peu chère et efficace en place.
- La **mémoire vive** (RAM en anglais pour Random Access Memory) contient les données actuellement en utilisation ainsi que les programmes en cours exécution. Elle s'efface une fois l'ordinateur éteint.
- Le **cache** (L1, L2 ou L3 selon la vitesse) contient les informations les plus utilisées de la mémoire vive. Il est bien plus rapide que la mémoire et se trouve sur la puce du processeur.
- Les **registres** stockent de l'information directement dans le processeur entre chaque opérations, ils sont très rapides et directement adressables par le processeur mais ne contiennent que très peu d'information (en général 4 à 8 octets chacun).



Définition 5 **Mémoire vive dynamique** Dans nos ordinateurs actuels, la mémoire vive utilisée est appelée DRAM, pour **mémoire vive dynamique** (Dynamic Random Access Memory). Physiquement, pour chaque bit, la DRAM comporte un condensateur et un transistor. Cette mémoire est donc dynamique car la charge des condensateurs doit être rafraîchie.

Définition 6 **Adresse physique** On accède à la mémoire vive en utilisant une **adresse physique**, indice de la case mémoire (de 8 bits) à laquelle lire/écrire.

Définition 7 **Boutisme** Lorsque l'on accède à une séquence d'octet contiguë en mémoire, il existe deux conventions pour l'ordre d'adressage :

- le **petit-boutisme**, les octets de poids faible en premier
- le **grand-boutisme**, les octets de poids fort en premier

Remarque 8 La plupart des processeurs modernes sont petit-boutistes, mais la convention pour les échanges en réseau est le gros-boutisme.

B. Du bit à la donnée

Définition 9 **Entiers non signés** Les entiers positifs (ou non signé) sont codés par leur représentation en base 2 : $b_0 b_1 ... b_k$ représente l'entier $n = \sum_{i=0}^{k} b_{k-i} 2^i$.

Définition 10 **Entiers signés** Les entiers signés sont codés par leur représentation en base 2 avec complément à 2^k: $b_0 b_1 ... b_k$ représente l'entier $n = -b_0 2^k + \sum_{i=1}^{k} b_{k-i} 2^i$.

Exemple 11 **1110** interprété comme un entier signé sur 4 bits vaut $-2^3 + 2^2 + 2^1 = -2$.

Remarque 12 En pratique, le nombre de bit disponibles pour coder une donnée est fini et dépend du type de la donnée.

Exemple 13 **Pour les entiers en C:**

Signés	Non signés	Nombre d'octets
short	unsigned short	2
int	unsigned int	4
long int	unsigned long int	4 ou 8

Définition 14 En pratique, les réels sont approximés par des flottants, dont le format utilisé est généralement celui décrit par la convention IEEE-754, où un flottant f est de la forme $f = (-1)^s * m * 2^e$ avec : $s \in \{0, 1\}$, $m \in [1, 2[$, $e \in \mathbb{Z}$. Un flottant est alors la donnée (s, e, m). DEV

Remarque 15 **Représentation des caractères** Il n'y a aucun consensus pour la représentation des caractères, et plusieurs conventions sont possibles : ASCII, Unicode, ISO-8859-1 etc...

Exemple 16 En C, le type char utilise la convention ASCII dont la représentation tient sur 1 octet.

Définition 17 Les programmes sont représenté par des **instructions machine** qui sont définie par un **jeu d'instruction**.

Exemple 18 RISC-V, MIPS et x86 sont des jeux d'instructions.

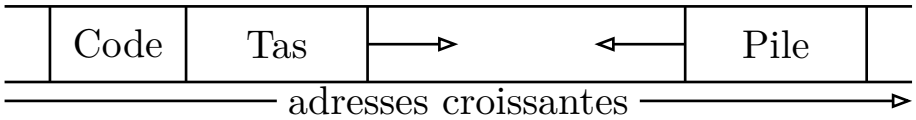
Définition 19 **L'assembleur** est le programme qui traduit le **langage d'assemblage** (texte lisible par les humains) en instructions machines.

II. Point de vue d'un processus [OSC]

A. Organisation mémoire d'un processus

Définition 20 L'espace d'adressage d'un processus est découpé en plusieurs parties :

- La **pile** contient les variables locales et les données nécessaires aux appel de fonction. Elle grandit par adresses décroissantes.
- Le **tas** contient les variables globales et les données alloué par le programme (comme avec `malloc` en C). Il grandit par adresses croissantes.
- Le **code** contient le code machine du programme exécuté.



B. Utilisation de la mémoire en C

Définition 21 **Un pointeur** est une variable qui contient l'adresse d'une donnée plutôt qu'une donnée elle-même. La taille des pointeur ne dépend que de l'architecture. Par exemple, les données de type **short*** et **long int*** ont la même taille.

Remarque 22

- Les variables globales sont attribuées dans le .data
- Les variables locales sont attribuées dans la pile

Remarque 23 Il est possible d'opérer sur des pointeurs pour obtenir des adresses différentes:

```
int *ptr1;
int *ptr2 = ptr1 + 1; // adresse suivante
int *ptr3 = &(ptr1[1]) // équivalent à ptr + 1
```

Remarque 24 En pratique, les adresses sont protégées : si un processus essaye d'accéder à une adresse sans autorisation, une exception est alors levée (erreur de segmentation). Ces autorisations sont indiquées dans une table associée au processus.

Remarque 25 En C, il existe des outils comme les tableaux (pointeur) permettant à l'utilisateur d'organiser la mémoire de son programme.

Définition 26 Certains appels systèmes permettent de manipuler de la mémoire:

- **malloc** permet d'allouer de la mémoire dans le tas
- **free** permet de libérer cette mémoire

Exemple 27 Par exemple, ce programme alloue, manipule et libère de la mémoire dans le tas:

```
int *tab = malloc(size * sizeof(int)); // alloue un tableau
d'entier de taille 'size'
tab[0] = 1; // manipulation de tab
free(tab); // libère la mémoire
```

DEV

Remarque 28 Politique d'allocation dans le tas [MOS 3.2.3] Allouer de la mémoire dans le tas peut faire l'objet de plusieurs politiques pour minimiser la place perdue.

Exemple 29 FIRST-FIT Exemple l'allocation pour cette stratégie:

```
char *tab1 = malloc(3); // A       Tas : [A|A|A| | | | | | ]
char *tab2 = malloc(4); // B       Tas : [A|A|A|B|B|B|B| | ]
free(tab1);             // C       Tas : [ | | |B|B|B|B| | ]
char *tab3 = malloc(2); // D       Tas : [D|D| |B|B|B|B| | ]
```

C. Adressage virtuel

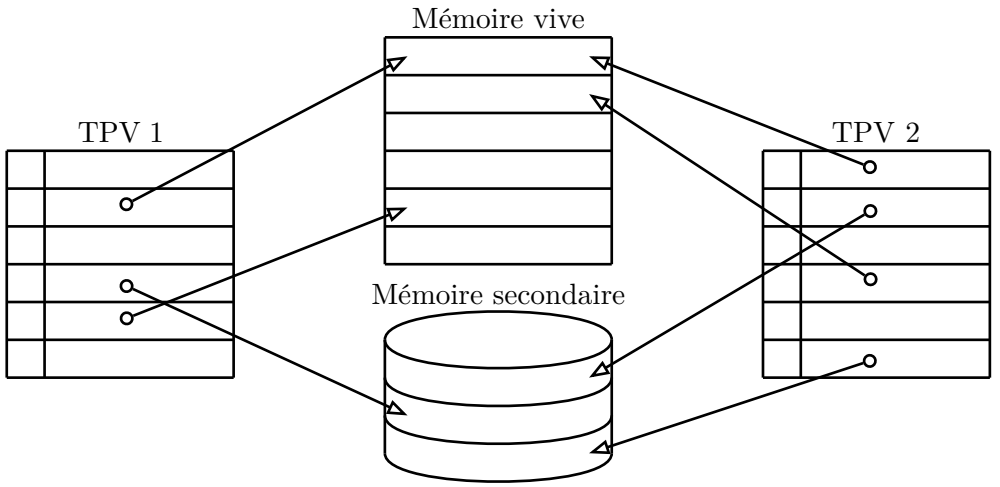
Définition 30 Adressage virtuel Un processus n'a pas un accès direct à l'espace d'adéressage de la mémoire physique, il interagit avec la mémoire de manière indirecte avec l'illusion d'une mémoire monolithique de grande taille. Cette abstraction est possible grâce à l'**adressage virtuel**, qui offre aux processus un espace d'adressage (dit virtuel ou logique) qui est associé à un autre espace mémoire (dit réel ou physique) stocké dans les mémoires principale et secondaire.

Remarque 31 La taille de l'espace d'adressage virtuel varie en fonction de l'architecture : 2^{32} ou 2^{64} selon si l'architecture est en 32 bits ou en 64 bits.

Définition 32 Page Les espaces d'adressage sont découpés en **pages** d'adresse consécutives et de taille fixe. Sur Linux, les pages mémoires font 4ko.

Remarque 33 L'adressage virtuel permet d'isoler les espaces mémoire des processus, d'offrir l'illusion d'une mémoire simple et très grande, et d'accéder malgré tout rapidement aux pages dans la plupart des cas.

Définition 34 MMU et TPV Les adresses virtuelles sont converties en adresses physiques par l'**Unité de Gestion de Mémoire** (MMU ou Memory Managment Unit en anglais) à l'aide de la **Table des Pages Virtuelles** (TPV). Chaque processus possède une TPV qui lui est associé.



Remarque 35 Pour associer les adresses virtuelles à leur adresse physique correspondante, la MMU vérifie dans la TPV du processus que la page concerné est dans la mémoire vive. Si ce n'est pas le cas, une exception est levé qui demande au système d'exploitation de charger la page en mémoire vive depuis la mémoire secondaire.

DEV

Remarque 36 Plusieurs politique de remplacement de page [MOS 3.4] existent lors d'un chargement de page. En effet si aucun emplacement n'est disponible, l'OS doit choisir une page à décharger.

Mémoire : du bit à l'abstraction vue par les processus

Commentaires

Bibliographie

[CSAPP] R. E. Bryant & D. R. O'Hallaron, *Computer Systems, A programmer's perspective.*

[OSC] A. Silberschatz & P. B. Galvin & G. Gagne, *Silberschatz's Operating System Concepts, Global Edition.*

[MOS] Tanenbaum & Bos, *Modern Operating System, 5th Edition.*

Adrien Decosse & Axel Stengel