La mémoire en C/C++

Nicolas Gazères

Dassault Systèmes
DS Research, Life Sciences
ngs@3ds.com

La mémoire

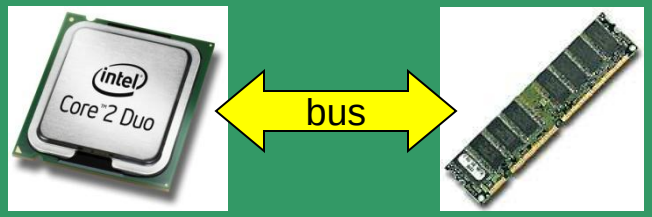
- La mémoire est une ***ressource***
  - une *matière première* indispensable au fonctionnement du programme:
    - le *code* du programme occupe de la mémoire.
    - les *variables* du programme occupent de la mémoire.
  - présente en *quantité limitée*
    - donc à gérer soigneusement dans le temps
      - allouer le minimum nécessaire
      - libérer rapidement ce dont on n'a plus besoin.

- **ATTENTION**
  - Un programme peut être *parfait* du point de vue algorithmique, mais être malgré tout perçu globalement comme *de mauvaise qualité* simplement parce qu'il gère mal la ressource "mémoire".

La mémoire physique (RAM)

- Composants mémoire
  - Barrette de RAM (transistors)
  - Séparé du processeur (il y a aussi les caches)

- Bus d'adresse / bus de données
  - Machine 32 bits / 64 bits
- Que contient la mémoire quand on allume l'ordinateur ?
  - N'importe quoi, pas forcément zéro.



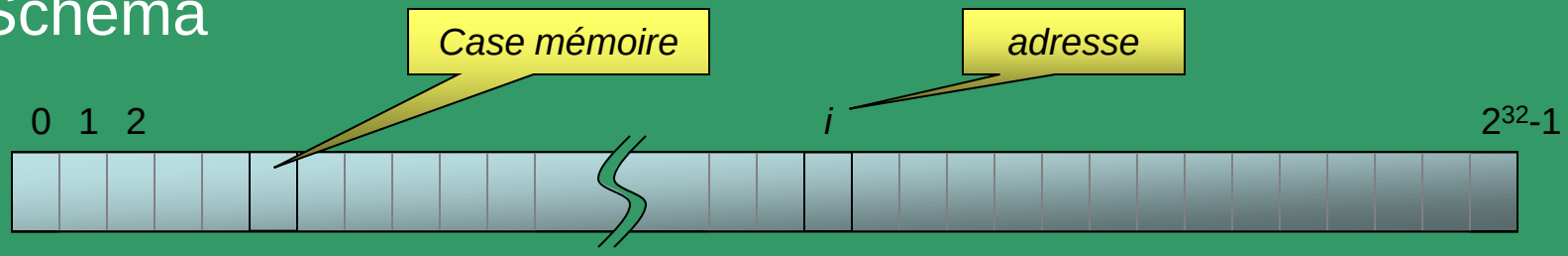
- Remarques
  - Dans les machines modernes, il y a une hiérarchie de composants mémoire.
    - Registres
    - Caches niveau 1, niveau 2…
  - Les caches n'ont une influence que sur le temps d'accès à l'information, pas sur la représentation de la mémoire.

La mémoire virtuelle

C'est ce type de mémoire qu'utilise un process.

- Modèle mental simplifié (machine 32 bits)
  - La mémoire mise à disposition d'un *process* peut être vue comme une suite de 2^{32} *octets* numérotés.
  - Le numéro de l'octet s'appelle *l'adresse*.

- Schéma



- Remarque:
  - Ce modèle mental simplifié reste encore valable en présence de registres et de caches.

Représentation mémoire des types

- Exemple pour un type scalaire: l'*int*
  - Quatre octets consécutifs → 32 bits → 2^{32} valeurs entières possibles.

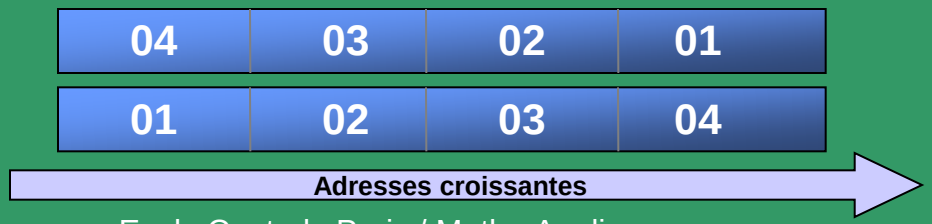
- Exemple pour un type flottant: *le float* (Norme IEEE-754)
  - Les quatre octets *(32 bits)* se décomposent comme ceci:
    - *d'abord 1 bit de signe (poids fort) (0=positif, 1=négatif).*
    - *puis 8 bits d'exposant (*ces bits stockent « *exposant+126 »).*
    - *puis 23 bits de significand (ie. mantisse)*

  - Le format IEEE-754 complet est complexe (cf. dénormalisations)

- Attention à l'interprétation comme un *int* d'une représentation *float*.
  - et vice-versa…

Little endian / Big endian

- Principe
  - Lorsqu'un type est représenté sur plusieurs octets (par ex. int, float, …), se pose la question de savoir dans quel ordre ranger les octets du type *en mémoire*.
- Définition: Little endian / Big endian
  - Sur une machine little-endian (ie. intel), c'est l'octet de poids faible qui est stocké à l'adresse-mémoire la plus basse.
  - Sur une machine big-endian (ie. sparc), c'est l'inverse.
  - Important pour
    - les échanges sur le réseau
    - Les sauvegardes de fichiers…
- Exemple
  - Soit l'entier 16909060 (0x01020304).
  - Little endian
  - Big endian



Opérateur *sizeof*

Types élémentaires

- Tous les éléments d'un type donné occupent le même nombre d'octets en mémoire.
  - → On peut donc parler de *la taille d'un type*.
- L'opérateur *sizeof* est un opérateur « compile-time » qui renvoie
  - La taille en octets du type d'une expression (sans parenthèses):
  - La taille en octets d'un type écrit explicitement (il faut alors des parenthèses).
- Exemples (machine Windows/Intel):

```
… sizeof(int) …          // renvoie 4
float f=2.8 ;
… sizeof f …             // renvoie 4
… sizeof(unsigned char)… // renvoie 1
```

- Remarques:
  - *sizeof* renvoie une valeur entière non-modifiable (de type *size_t*)
  - Le calcul de *sizeof* est fait à la *compilation*, pas à l'exécution.
  - Le calcul de *sizeof* est fait sur la base du *type* de l'expression qu'on lui donne, pas du résultat.

Fin