07 - Mémoire

Gaétan Richard
gaetan.richard@unicaen.fr

L2 – S4 2011/2012

1. Données et mémoire

Encodage

Fait : La mémoire informatique fonctionne sous forme de bits 0 ou 1 (b). Ces bits sont groupés par 8 pour former des octets (bytes an anglais, B).

Alignement et mots : L'espace élémentaire (appelé mot) varie selon le processeur (32, 64, 128, …bits). Les données doivent être alignées en mémoire.

Nombres

Représentation : Il existe trois grandes représentation des nombres en mémoire :

- **Entier non-signé** : l'écriture en binaire standard ;
- **Entier signé** : écriture en complément à 2 ;
- **En virgule flottante** : écriture mantisse / exposant.

Endianess : dans le cas où l'on manipule un entier de taille plus grande qu'un mot mémoire, on peut le décomposer en plusieurs blocs qui sont rangés :

- poids fort en tête : big endian (ex : Motorola,SPARC) ;
- poids faible en tête : small endian (ex : x86) ;
- bizarrement : Middle-endian.

Chaînes de caractères

Encodage : Les chaînes de caractères sont transformées en suite d'entiers à l'aide d'une table de codage de caractère. Il en existe plusieurs :

- **ascii** : l'historique ;
- **latin-1** : pour les langes ouest-européennes (avec latin-9) ;
- **UTF-8** : très général.

Le codage peut se faire en taille fixe ou en taille variable.

Une fois transformée, la chaîne peut être stockée :

- avec un délimiteur de fin : `\0` (ex : C, python, ...) ;
- en indiquant avant la longueur de la chaîne (ex : fortran).

Autres

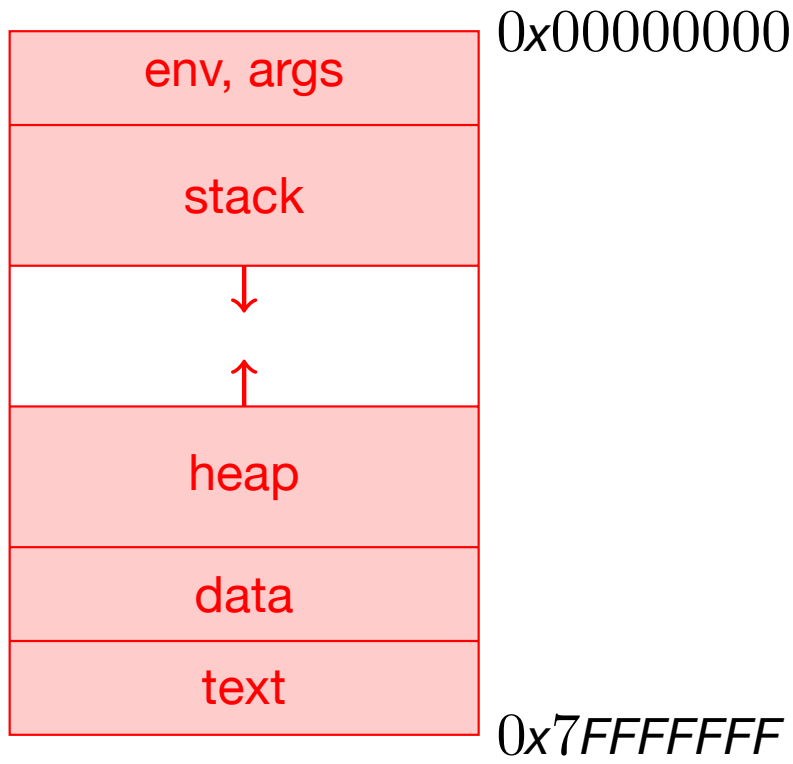
Règles : Toutes les objets sont manipulés en binaire dans l'ordinateur.

Pointeur : Il est possible également de stocker un entier qui indique une adresse en mémoire. On parle alors de pointeur.

2. Organisation de la mémoire

Rappel sur les processus

uid / gid
euid / egid
ordonnancement
files desciptors
signaux
...



Allocation sur la pile

Utilisation : La pile (stack) est utilisée pour stocker les arguments lors de l'appel d'une fonction et le résultat de la fonction. C'est également là que se trouvent les variables locales.

Allocation : En C, on peut allouer de la mémoire sur la pile à l'aide de

```
void * calloc(size_t count, size_t size)
```

Libération : Cette mémoire est automatiquement libérée lors de la sortie d'une fonction.

Allocation sur le tas

Utilisation : La tas (heap) sert à stocker les données permanentes.

Allocation : En C, on peut allouer de la mémoire dans le tas à l'aide de l'appel (ou d'une de ses variantes) :

```
void * malloc(size_t size)
```

Libération : Pour libérer un objet de cette mémoire, il faut faire appel à la fonction :

```
void free(void *ptr)
```

sous peine de se retrouver avec des fuites de mémoires.

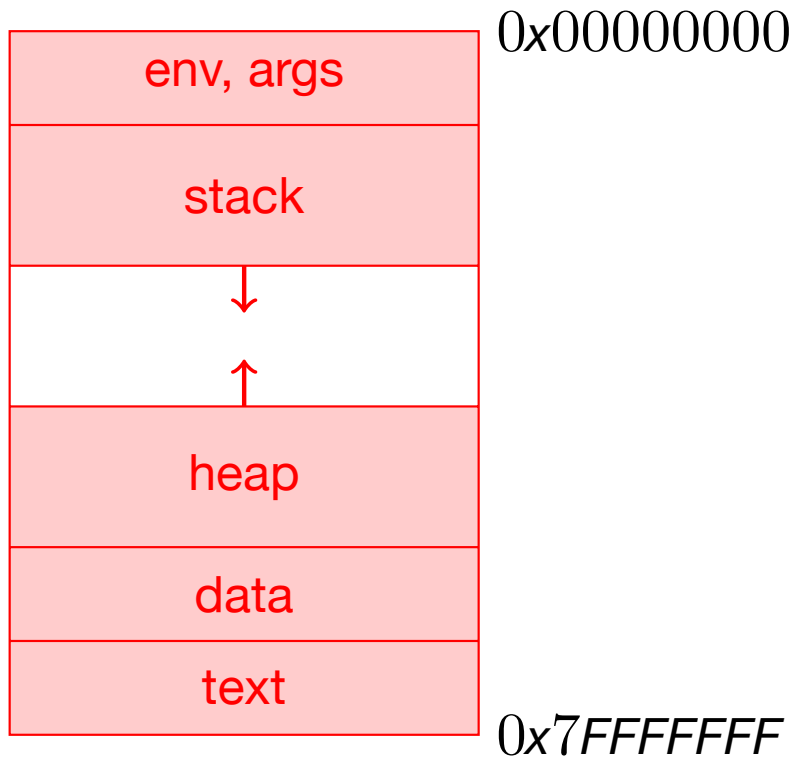
Et sous python

Sous le tapis : Sous python, toutes les opérations de mémoire sont automatiques et il n'existe pas de notion de pointeur.

Remarque : La question de libération se pose néanmoins pour les descripteurs de fichiers ouverts.

Adresses mémoires

uid / gid
euid / egid
ordonnancement
files desciptors
signaux
...



Tous les processus ont une mémoire ayant des adresses entre 0 et le maximum.

Question : Comment faire en sorte qu'ils aient chacun un espace différent ?

Pagination

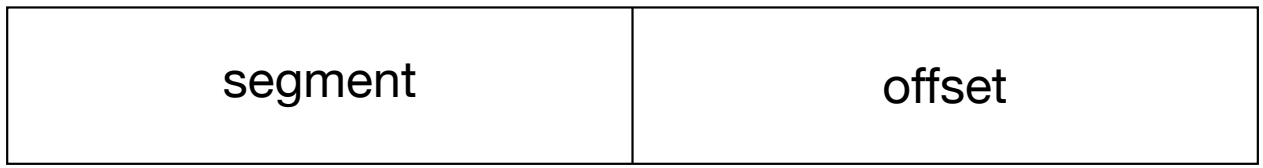
Principe : Pour gérer la mémoire, on divise en blocs élémentaires de taille importante : les pages.

Taille : Actuellement, la taille usuelle des pages est de $4Ko$. On peut l'obtenir à l'aide de la commande `getconf PAGESIZE`.

La taille des pages est un compromis entre l'espace éventuellement gaspillé dans chaque page et l'espace nécessaire pour gérer les pages.

Segmentation

Segmentation : Chaque adresse est divisé en deux parties :



Le segment est transformé à l'aide d'une table des pages en adresse réelle par le système d'exploitation.

Matériel : Ce processus est accéléré par la présence d'une puce dédié à ce travail : la MMU (memory management unit)

Protection

Droits : Comme dans tous les cas précédents, chaque page de mémoire dispose de droits (lecture / écriture / exécution).

No exec : Dans les processeurs modernes, le processeur prend en charge un mécanisme interdisant l'exécution de code si la page n'est pas marquée comme exécutable.

Randomisation : Pour protéger encore d'avantage le système, il est maintenant courant d'introduire une portion d'aléatoire dans la position réelle en mémoire des bibliothèques ou des données afin de rendre plus difficile l'utilisation frauduleuse.

Rôle du système d'exploitation

Allocation / Libération : Le système d'exploitation à la charge d'allouer et de libérer les pages de mémoires à destination des processus.

Partage : Certaines pages peuvent être partagées entre plusieurs processus (ex : bibliothèques).

Mémoire virtuelle : Il est donc courant de distinguer pour un processus :

- la mémoire totale utilisée VSIZE ;
- la mémoire propre utilisé RSIZE ;

Méthode d'allocation

Principe : Le système d'exploitation reçoit des demandes de mémoires / libérations. Il doit alors gérer la position de ces pages dans la mémoire réelle.

Heuristiques : Il existe plusieurs méthodes usuelles d'affectation de la mémoire :

- First fit : on prend le premier espace libre ;
- Best fit : on prend le plus petit espace libre ;
- Worst fit : on prend le plus grand espace libre ;
- Next fit : on prend l'escape libre suivant.

Fragmentation

Contiguïté : on essaie de maintenir le plus possible la contiguïté des données (l'accès à des zone de mémoires contiguës est facile).

Fragmentation : Si les demandes / libérations de mémoires sont étalées dans le temps, il se produit un phénomène de fragmentation : l'espace libre de retrouve divisé en petits blocs répartis sur toute la mémoire.

Conséquences : Ce phénomène ralenti l'exécution de la machine et dans certains cas peut empêcher d'obtenir de la mémoire alors qu'il y aurait théoriquement la place.

3. Cache

Présentation

Problème : Il existe un compromis entre la taille de la mémoire et sa vitesse d'accès.

Idée : Garder "sous le coude" une partie de la mémoire pour l'avoir plus rapidement : **cache**.

Cadre : Dans cette mémoire, on mets à disposition des copies de portions de mémoire. Les emplacement disponibles dans le cache sont appelés cadre

Mise en œuvre

On copie la partie “utile” de la mémoire dans le cache.

Lorsque l’on a besoin de mémoire :

- si la portion voulue est en cache, on l’utilise ;
- sinon, on charge la portion désiré depuis la mémoire et on effectue l’étape précédente (défaut de cache).

Problème : Que faire lorsque tous les cadres sont occupés ?

Algorithmes

Solution : on libère une place dans le cache.

Heuristiques : Pour choisir la cadre à libérer, on peux utiliser les heuristiques suivantes :

- choix au hasard ;
- choix du plus ancien (FIFO) ;
- choix de celui non utilisé depuis le plus longtemps (LRU) ;
- choix du moins fréquemment utilisé (LFU).

Il existe un algorithme optimal qui consiste à libérer le cadre qui sera réutilisé dans le futur le plus lointain. Cet algorithme n'est bien sur pas utilisable dans le cas de l'ordinateur qui ne connaît pas l'utilisation future de la mémoire.

Cohérence

Dirty bit : lorsque l'on choix décrire en mémoire, l'écriture est faite directement dans le cache et n'est pas transmise à la mémoire. On note alors juste que des changements on été effectués (dirty bit).

Synchronisation : Les écritures sont effectuées lors de la libération du cadre.

Cohérence : Ce système permet de gagner en efficacité mais pose des problèmes de cohérence.

Hiérarchie

Efficacité des caches : En pratiques, les caches sont extrêmement efficaces. De nos jours, on n'en rencontre partout et il n'est pas rare d'avoir une chaîne de cache importante entre le processeur et la mémoire la plus lente (ex : disque dur).

Compromis : La taille des caches est principalement limitée par leur coût.

Performance : Dans le cas où l'on souhaite obtenir des performances importantes de calcul, il est important de prendre en compte le fonctionnement du cache et sa taille.

4. Garbage collector

Présentation

Principe de base : Normalement, le programme demande afin d'obtenir de la mémoire et indique quand la mémoire est de nouveau disponible.

Gestion automatique : Dans certains langages (comme python) la mémoire est allouée automatiquement et libérée lorsqu'elle n'est plus utilisée. Le mécanisme dédié à cette deuxième tâche s'appelle le garbage collector (Ramasse-miettes en français).

Comptage de références

Une méthode simple pour déterminer quels données sont utilisées ou non est le principe du comptage de référence.

À la manière du système de fichier, il existe pour chaque valeur en mémoire un compteur qui indique combien de variables pointent vers cette valeur.

Si ce compteur atteint 0, alors la valeur n'est plus d'aucune utilité et peut être libérée.

Comptage de référence (cont.)

Boucle : le principal problème du comptage de référence est la présence possible de boucles. Elles empêchent de récupérer des valeurs pourtant inutiles.

Pour contrer ce problème, le garbage collector dispose souvent d'un mécanisme de recherche des boucles.

Fragmentation : Bien évidement, de nombreuses allocations / libérations de mémoire peuvent entraîner une fragmentation de la mémoire.

Mark and sweep

Une deuxième grande catégorie de méthode pour collecter l'espace libre est le mark and sweep.
Pour cela, on divise l'espace en deux parties. On utilise uniquement une moitié et on effectue l'opération suivante :

- On marque toutes les parties accessibles de la mémoire (mark) ;
- On recopie dans l'autre partie toutes les valeurs marquées en mettant à jour les liens (sweep).

On recommence cette opération périodiquement.

Mark and sweep (cont.)

Avantage : Cette technique permet d'éliminer tous les problèmes de fragmentation.

Inconvénients : En théorie, le collection s'effectue par à-coup et bloque le fonctionnement du programme. Il est néanmoins facile de faire un "mark" incrémental. Il est plus délicat de le faire pour le sweep.
Dans tous les cas, cette technique demande de recopier de grosses quantités de données.

Générations

Remarque : Toutes les variables n'ont pas la même durée de vie.

Corollaire : Il faudrait avoir des fréquence différentes pour les collections selon les types de variables.

Solution : On divise l'espace mémoire en plusieurs zones où on classe les variables selon leur âge. Le GC est d'autant plus fréquent que les variables sont jeunes.

Avantages et inconvénients d'un GC

Avantages :

- La gestion de la mémoire est automatique ;
- On évite la fragmentation ;
- La perte de temps est négligeable.

Inconvénients :

- On introduit potentiellement un lag ;
- La gestion n'est pas optimale ;
- On en peut plus manipuler directement la mémoire ;
- C'est très complexe à mettre en œuvre avec du parallélisme.

Ce genre de mécanisme est présent principalement dans des langages de haut-niveau dans lesquels on n'a pas d'accès direct à la mémoire.

En python

Note : le GC ne fait pas partie du langage python mais dépend de l'interpréteur utilisé.

Tuning : Il est néanmoins possible d'interagir avec le GC par l'intermédiaire du module **gc**.

Implémentation : Sous CPython, il existe un système de comptage par référence muni d'une méthode de recherche de cycle auquel vient s'ajouter un garbage collector de type mark and sweep avec 3 générations.