System

- Numero: 4
- Prof: Laskar Gabriel
- Date: 37 Octobre 2017

Memoire

La memoire est decouper en block de globalement (4 Ko). Le but est de donner un environnement simple qui va leur etre propre (personne d'autre pourra y acceder).

Espace d'adressage:

> Cela va representer un range d'adresse qui sera dispobible. Ca a un debut et une fin. Dedans il va y avoir des choses alloue ou non. Ces zones sont des pages. (Sur x86 elles font 4 Ko).

- HD $\rightarrow$ 512
- CD $\rightarrow$ 2Ko
- HD (recent) $\rightarrow$ 4Ko

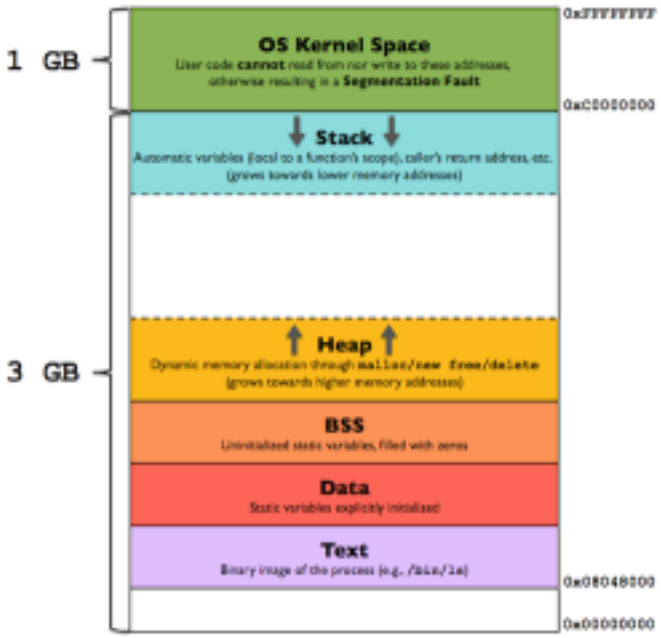

Figure 1: Representation de l'espace memoire d'un programme

Cela n'a rien a voir avec la memoire physique.

Exemple d'une machine 32 bits

- $2^{10} \rightarrow 1024$
- $2^{20} \rightarrow 1M$
- $2^{30} \rightarrow 1G$

Si j'ai des adresse qui font 4b. On a des adresse qui vont de $0 \rightarrow 2^{32}$. Soit 4Go.

L'adresse d'une page est multiple de 4K. Il y a donc 1Mo de pages.

Pour mapper les addresses virtuelles en physique on va avoir un tableau:

- Taille: 1Mo.
- L'indice correspond a la page.
- Le contenu l'adresse physique. 20 bits donc 4 bytes. Il va rester 12bits pour des flags.

```
struct
{
  u20 phys_addr;
  u5  flags(present, RWX, S);
} pages[2^20];
```

Ce tableau prend donc 4M. On doit les reserver a chaque fois pour chaque programme. C'est pas terrible. On peut donc faire mieux, on va utiliser un Btree.

Block Tree est un arbre general de recherche. Un noeud est un tableau de cles. Ca permet de representer des DB et des filesystems.

Du coup la taille d'un noeud fait la taille d'un block (ici 4Ko). Il a une hauteur deux.

L'adresse virtuelle a 10 bits pour le tableau de pages (Page table). Puis 10 bits pour la page et enfin 12 bits pour la donnee dans la page.

Cache TLB:

- Translation
- Lookaside
- Buffer

Gerer les erreurs

Pour report une erreur on veut creer une **Exception**.

Exception:

> Un moyen asynchrone de signaler l'erreur. Envoi un signal et un catcher va la receptionner.

Par defaut le programme quitte, mais on peut la rattraper et la gerer comme on peut. On a une table d'execption pour faire correspondre les erreurs a leur code.

⊛ **QUESTION DU PARTIEL** Sur x86 les PageFault arrive si on veut acceder a une zone NonMappe ou s'il n'ont pas les bonne permissions.

Un programme qui ne fait rien a deja plus de 1000 PageFault.

On va donner la memoire uniquement lorsqu'elle sera utiliser. Du coup l'allocation va "rien faire" c'est l'utilisation qui va vraiment allouer la memoire.

Resume

On a une structure de donne pour faire du mapping qui gere des droits entre les adresse virtuelle et physique.

Algo malloc

Voici la liste des syscalls pour allouer de la memoire:

- mmap
- munmap
- mprotect
- mremap (Linux only)
- maduise

On va uniquement s'interesser a `mmap` et `munmap`. On ne veut pas directement les utiliser du coup on va utiliser un allocateur memoire.

- `calloc` = `malloc(A * B)` + `memset(0)`
  - sauf que `A * B` peut overflow.
- `realloc`(stupide) = `malloc` + `memcpy` + `free`

Du coup `malloc` et `free` sont des cas speciaux de `realloc`.

```c
void *malloc(size_t s)
{
  return realloc(NULL, s);
}
```

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

```
2 {
3   realloc(ptr, 0);
4 }
```

Solution 1

On appelle `mmap` pour chaque `malloc`. Le probleme c qu'on est en $\Theta(n)$. De plus on alloue 4K pour chaque malloc.

Solution 2

Les algos FirstFit, BestFit, WorthFit… Sont identique c'est du $\Theta(n)$.

Solution 3

Si on veut gagner un peu de place, on va maintenir la liste des bloques libres. La recherche reste lineaire.

Solution 4

On prend une zone de 4K avec un peu de donner puis on va avoir des blocks de la meme taille. On va ensuite stocker des metasdonnees dans les blocks free. L'idee est de faire un numberset pour savoir quel block est libre.

Pour les grosse alloc (de plus d'unne page). On pourrait `mmap` betement. Ou alors on peut chainer des pages en utilisant malloc.

Thread safe

On peut utiliser des variables thread locale. En gros on peut faire un allocateur par thread.

Il y a un probleme c'est alloue dans un thread et liberer dans l'autre.

Pour tester on peut utiliser malloc pour compiler nos propre programme.