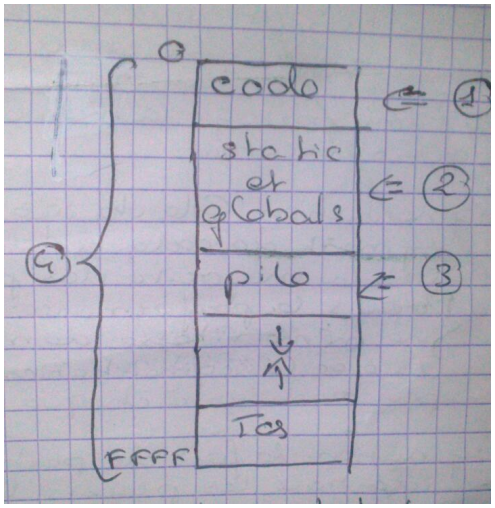
Prise de note Système d'Exploitation

Vue globale :
Programme c quelconque :

```
int a;
chat Tab[100];
float Foo(int x, int y){
   struct * p ;
   while(   ){... }
}
int main(){... }
```



On imagine que la mémoire nous appartiens totalement.
Le compilateur génère le binaire comme si toute la mémoire lui appartient.

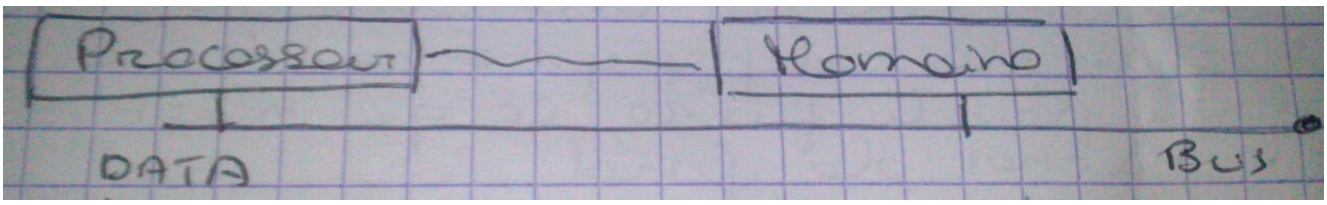
En fait la mémoire est virtuelle, de façon à être vu comme unique par le compilateur.

Pile :
Lorsqu'on appelle une fonction, on ouvre un bloc. A chaque appel on empile variables locales de fonction, paramètre de fonction, et adresse retour de fonction. A la fin de l'exécution d'une fonction, on dépile. Lorsqu'une fonction en appelle une autre, on empile une pile dans la pile initiale.

Tas :
Zone mémoire de toutes les données allouées avec un malloc (tout les buffers du programme)

Le binaire contient tout le tableau : 4

Coté processeur :



Le processeur envoie une adresse physique à la mémoire. Il faut un mécanisme traduisant la mémoire virtuelle à celle physique. Le SE voit cette correspondance de mémoire.

MMU : Memory Management Unit

// ! \\ La gestion mémoire est différente selon où on se place.

Pourquoi une adresse virtuelle ou physique ?
- protection mémoire
- utiliser plus de mémoire que la RAM
- un programme binaire est indépendant de la capacité de la RAM
- la RAM physique est plus compliqué
- les processeurs peuvent partager la même image

Le système partage la mémoire physique entre tout les processeurs qui s'éxécutent.

Segmentation :

Elle oblige à partitionner les données entre segment de code, de donnée, de pile. Mais la majorité des programmes n'utilisaient qu'un segment. Depuis la performance des processeurs à augmenter, pour une même vitesse de mémoire. Or avec la segmentation on fait à chaque fois des accès mémoire dans le système pour traduire à chaque fois virtuel/physique

Pagination :

Le fait de partager la mémoire en bloc de mémoire identique, les pages. Chaque page est de 4 Ko (ou 4Mo).

Il y a 2 types de pages :

- physique de la barrette de RAM : frame
- de la mémoire virtuelle : page virtuelle

Quand un processeur demande de la mémoire, le système une page. Si il demande 2 octect, le système va chercher dans une page déjà utilisé de la place restante, sinon il en ouvre une nouvelle.
Le système fait correspondre frame et page virtuelle.
Il gère allouage de page physique, de page virtuelle, et la défragmentation en mémoire.
L'adresse physique de la page est sur 20b.

Linux gère ses propres frames. Même si aucun logiciel ne l'aide, il a l'avantage d'avoir tous les droits. Il ne se préoccupe pas de page virtuelle.

Le noyau alloue des cadres contigüe, mais à force de trafic de cadre, on fini par avoir une mémoire à trou, fragmentée. Alors on doit défragmenter la mémoire, mais le système se bloque pour le faire, ce qui n'est pas avantageant pour travailler, on préfèrerais que ça se fasse en tache de fond.
Pour résoudre cela, on va renvoyer des adresses contigües ne renvoyant pas vers des cadres contigües. Le problème est que certain périphérique matériel (ex : carte son), doivent écrire sur les adresses physiques. Le KMA (Kernal Memory Allocator) est alors perdu.
Linux lui utilise les zones siamoises, le buddy. On alloue beaucoup, que l'on fragmente hierachiquement. Les allocations contigües sont plus pratiques car le cache charge automatiquement les adresses en bloc continues. Mais il est possible de le faire en non-contigüe.

Le noyau doit gérer la RAM pour lui même et pour les processus. Il utilise le buddy pour les gros besoins (besoin de fram), ou le slab, l'allocation dynamique destinée au noyau.
Donc pour quelque octet on utilise le slab, mais celui-ci utilise le buddy pour avoir des cadres physiques, le slab n'a aucune mémoire.