SYS2

Système d'exploitation

M.Bastreghi (mba)

Haute École Bruxelles Brabant — École Supérieure d'Informatique

Année académique 2020 / 2021

Section Memoire

Memoire

- Introduction
- Aucune abstraction
- Abstraction de la mémoire
- Taille et swap
- Segmentation (mode réel, protégé)

vue idéale

Chaque processus doit disposer d'une mémoire :

- privée
- infiniment grande
- rapide
- non volatile
- réalisée dans une technologie "bon marché"

hyérarchies de mémoires

TYPE	TAILLE	RAPIDE	NON VOLATILE	BON MARCHÉ
cache	MiB	++	NON	–
mémoire vive	GiB	+	NON	-
disques (magnétiques - SSD)	TiB	-	OUI	+

hyérarchies de mémoires

La mémoire vive (RAM) est utilisée pour les instructions et variables d'un programme.

On y accède de mots via des adresses.

Le mot est l'**unité adressable**

La mémoire vive (RAM) est utilisée pour les instructions et variables d'un programme.

On y accède de mots via des adresses.

Le mot est l'**unité adressable**

Chaque "mot" en RAM correspond à une adresse.
En déposant sur le bus d'adresse l'adresse on peut lire ou modifier le mot en RAM.
Le terme **mot** indique l'unité adressable en RAM.

Dans les architectures intel, cela correspond à un octet.

Adresses physiques

Sans abstraction de la mémoire le programme manipule des adresses vraies.

```
MOV [8192], reg
```

Dans ce cas 8192 est une adresse vraie en RAM

Le programme est maître des adresses qu'il utilise en RAM
Deux programmes différents pourraient utiliser la même adresse.
On exécute **un seul programme** à la fois (monopogrammation).

Adresses physiques

Sans abstraction de la mémoire le programme manipule des adresses vraies.

```
MOV [8192], reg
```

Dans ce cas 8192 est une adresse vraie en RAM

Le programme est maître des adresses qu'il utilise en RAM
Deux programmes différents pourraient utiliser la même adresse.
On exécute **un seul programme** à la fois (monopogrammation).

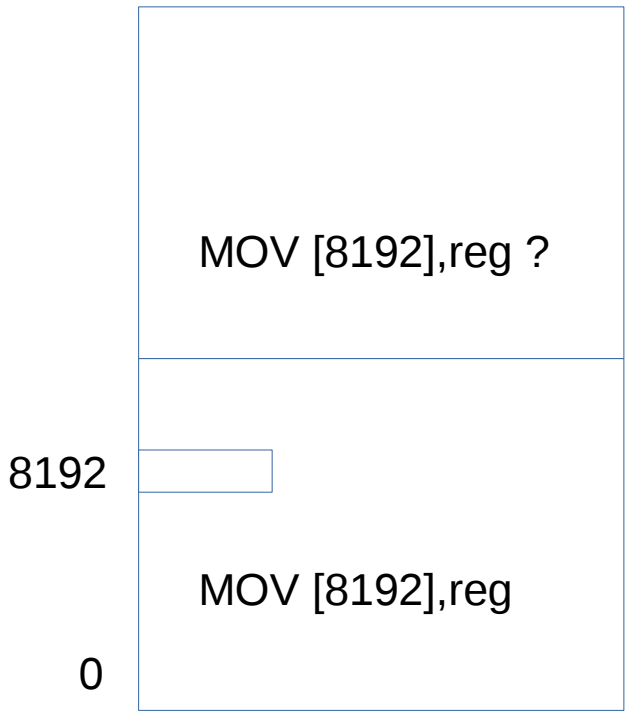
L'adressage physique est direct, il peut être utilisé

Lorsque il y a un seul programme en mémoire destiné à y rester (softwares embarqués, cartes à puce, programmes en ROM) .

Adresses physiques

Deux programmes utilisant une adresse réelle pour une variable

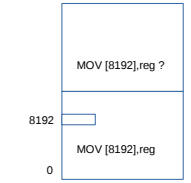
```
MOV [8192], reg
```



Adresses physiques

Deux programmes utilisant une adresse réelle pour une variable

MOV [8192], reg



Il est possible de faire coexister plusieurs programmes en mémoire à condition que chacun utilise une zone de mémoire différente

L'adressage physique ne permet pas la multiprogrammation

Relocation statique

Relocation statique Le programme utilise des adresses relatives ajustées **au chargement**.

c'est la relocation statique

Relocation statique Le programme utilise des adresses relatives ajustées **au chargement**.

c'est la relocation statique

En relocation statique la traduction des adresses relatives en adresses physiques est réalisée une seule fois au **chargement du programme**

Relocation statique

Toute rérérence à la mémoire est corrigée en y ajoutant l'adresse de chargement du programme

Le programme qui s'exécute utilise les **adresses physiques**, que le chargeur aura adaptées.
L'adresse 8192 deviendra 108192 pour le programme chargé en 100000 et 308192 pour le programme chargé en 300000.

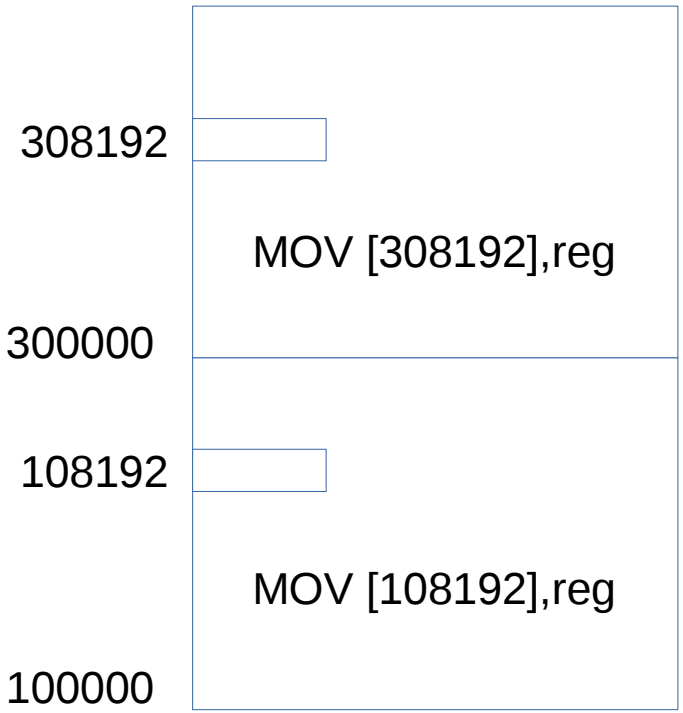
Relocation statique

Toute rérérence à la mémoire est corrigée en y ajoutant l'adresse de chargement du programme

Le programme qui s'exécute utilise les **adresses physiques**, que le chargeur aura adaptées.
L'adresse 8192 deviendra 108192 pour le programme chargé en 100000 et 308192 pour le programme chargé en 300000.

après le chargement du programme toutes les adresses auront été converties en adresses physiques

Relocation statique



Relocation dynamyque

Relocation dynamique Les adresses du programme qui s'exécute sont exprimées par rapport à un espace d'adressage propre au programme

Chaque programme a son/ses propre(s) espace(s) d'adressage

Relocation dynamique Les adresses du programme qui s'exécute sont exprimées par rapport à un espace d'adressage propre au programme

Chaque programme a son/ses propre(s) espace(s) d'adressage

Les adresses ne sont pas modifiées au chargement du programme. Elles sont exprimées relativement à l'espace d'adressage.

Les adresses sur un intel sont sur 32 bits, l'espace d'adressage y est donc limité à 4GiB

Relocation dynamique

$$\boxed{\text{RIP} = ? \; 8192 = ?}$$

Un dispositif **hardware** se charge de la traduction de chaque instruction à la volée **au moment de l'exécution** .

c'est la **relocation dynamique**

Relocation dynamique

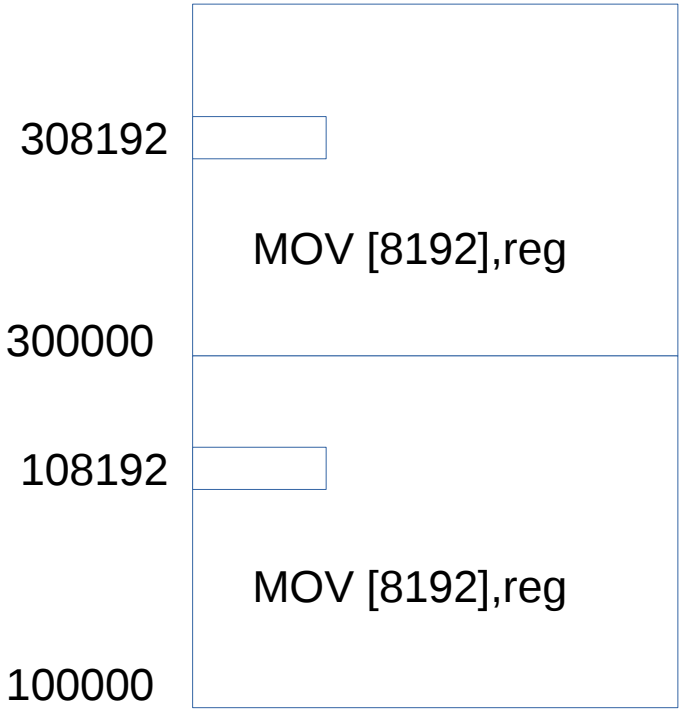
RIP = ? 8192 = ?

Un dispositif **hardware** se charge de la traduction de chaque instruction à la volée **au moment de l'exécution** .

c'est la **relocation dynamique**

sur intel c'est le **MMU** Memory Management Unit qui est responsable de la traduction des adresses à l'exécution

Relocation dynamique



Relocation dynamyque

```
MOV    [8192], reg
```

Exécuter l'instruction nécessite **deux accès** à la RAM et donc deux traductions

1. lire l'instruction depuis la RAM
2. écrire en RAM le contenu de reg

- (-) La correction d'adresses à l'exécution, ralentit l'exécution des programmes.

Relocation dynamyque - mise en oeuvre

registres base et limite

Deux **registres spéciaux** mémorisent l'adresse de chargement **Base** et la taille du programme **Limite**

A chaque accès à la RAM l'adresse relative est validée et convertie :

- adr < limite ? (validation)
- adr = adr+base (traduction)

le dispositif **Hardware** s'en charge

La segmentation en mode protégé sur intel utilise un mécanisme similaire.

Relocation dynamyque - mise en oeuvre

registres base et limite

Deux registres spéciaux mémorisent l'adresse de chargement Base et la taille du programme Limite

A chaque accès à la RAM l'adresse relative est validée et convertie :

- adr < limite ? (validation)
- adr = adr+base (traduction)

le dispositif **Hardware** s'en charge

Relocation dynamyque

- (+) permet la **coexistence de programmes** en mémoire, et la protection de leurs espaces respectifs
- (+) **swapping et déplacement** facilement envisageables (pas de corrections à apporter au programme on modifie le registre de base)
- (-) chaque accès mémoire nécessite **une addition et une comparaison** (lent)

Limites dues à la taille de la RAM

Swapping

La taille de la mémoire physique peut se révéler insuffisante pour accueillir l'ensemble de programmes qui tournent en même temps

Deux approches permettent de faire face à ce nouveau problème :

- swapping de programmes
- swapping de parties de programme

Swapping de programmes

Un programme sort de mémoire au profit d'un autre.

Á noter que tous les programmes n'occupent pas le même espace en RAM.

La RAM se fragmente !

Défragmenter la RAM est coûteux en temps.

Un programme sort de mémoire au profit d'un autre.

Á noter que tous les programmes n'occupent pas le même espace en RAM.

La RAM se fragmente !

Défragmenter la RAM est coûteux en temps.

une mémoire fragmentée présente une multitude d'espaces libres de taille insuffisante à accueilleir un nouveau processus.

Défragmenter la RAM revient à déplacer les processus chargés afin de libérer de l'espace contigu

Swapping de parties de programmes

Un programme ne doit pas résider entièrement en RAM à tout moment de son exécution. C'est cette propriété qui

est exploitée en **pagination** ou mémoire virtuelle

Un programme ne doit pas résider entièrement en RAM à tout moment de son exécution. C'est cette propriété qui est exploitée en **pagination** ou mémoire virtuelle

en pagination les programmes seront divisés en pages de même taille. Seulement une partie des pages des programmes résideront en RAM à un instant donné.

segmentation intel

Segmentation sur INTEL

L'espace d'adressage physique d'un processus peut être séparé en segments (code, données, ...).

La segmentation sur INTEL est une gestion de la mémoire à deux niveaux :

- mode réel
- mode protégé

adressage en mode réel

Mode Réél

Le **mode réel** est le seul mode d'adressage des premiers processeurs 8086.

- Ce mode d'adressage réel subsiste dans les processeurs actuels et est utilisé uniquemen au démarrage.
- Le processeur accéde à 1MiB de RAM en ce mode.

adressage en mode réel

- Une adresse est exprimée via deux registres ou immédiats de 16 bits (segment - offset)
- Ces registres combinés donnent une adresse sur 20 bits
- Un des registres joue le rôle de **registre de base**
- N'utilise pas de registre **limite**
- AB15 :0F34

L'offset sur 16 bits permet d'adresser 64Kib

adressage en mode réel

Adresse physique

- Adresse physique = segment*16 + offset.
- L'adresse AB15 :0F34 devient AB15**0**+0F34 = AC084.
- AC084 est l'adresse physique **sur 20 bits**.

Adressage limité à **1 MiB** de mémoire physique utilisant deux registres de 16 bits

adressages en mode réel

- La base des segments est une adresse multiple de 16 (2^4)
- L'offset de 16 bits permet aux segments une taille de 64Kib
- La valeur des registres est quelconque -> le programme accède à l'entièreté de la mémoire de 1MiB.
- Aucun mécanisme de protection prévu

mode protégé (386)

Mode Protégé

- la manière de traduire les adresses différencie les deux modes
- La traduction d'adresse se fait en plusieurs étapes
- adresse logique -> adresse linéaire -> adresse physique

La dernière étape est nécessaire si on utilise la pagination

Mode Protégé

- la manière de traduire les adresses différencie les deux modes
- La traduction d'adresse se fait en plusieurs étapes
- adresse logique -> adresse linéaire -> adresse physique

La dernière étape est nécessaire si on utilise la pagination

Comme son nom l'indique, ce mode permet de valider les accès à la mémoire et notamment les dépassements de frontière de segment ou des accès illicites (RING)

Le passage au mode protégé se fait en mettant à 1 le bit PE du registre CR0

mode protégé (386)

On part d'une **adresse logique**

- Une **adresse logique** est une adresse dans un segment, elle est composée de deux parties : **registre sélecteur-offset**
- **registres sélecteurs** : CS, DS, SS, ...
  - CS - associé au segment de code
  - DS - associé au segment de données
  - SS - associé au segment de pile
  - ...

adresse logique

Un programme utilise plusieurs segments, leur utilisation est parfois implicite :

- JMP BOUCLE - JMP **CS** :BOUCLE
- MOV EAX,[EBX] - MOV EAX, [**DS** :EBX]
- PUSH EAX - utilise **SS** :ESP

adresse logique

- **sélecteur** de segment : 16 bits (CS, DS, SS,..)
- **offset** dans le segment : 32 bits (offset) .

Chaque segment est un espace d'adressage délimité par une base et une limite (taille)

segments

- L'architecture intel prévoit 6 registres sélecteurs de segment
- CS SS DS ES FS GS (registres de 16 bits)
- Un programmeur peut utiliser plus de segments mais seulement 6 seront disponibles en même temps.

sélecteur de segment

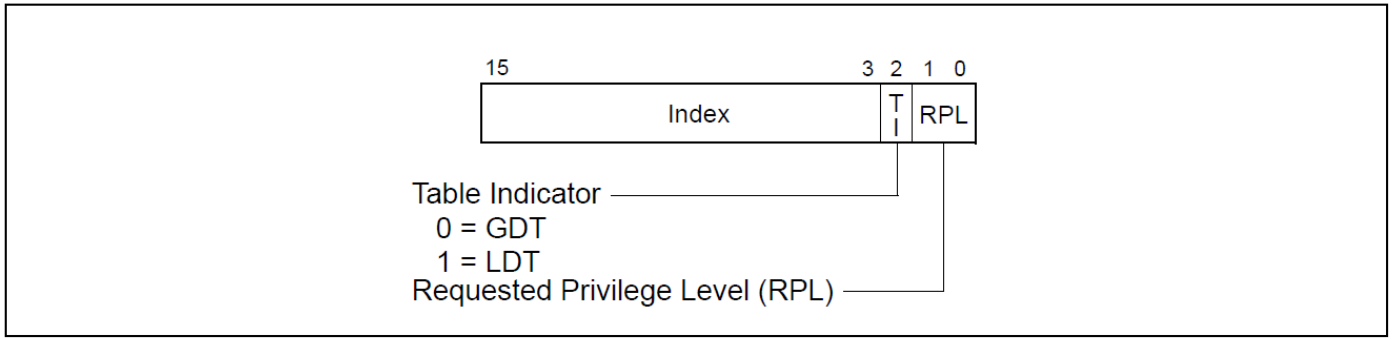


Figure 3-6. Segment Selector

sélecteur de segment

Adresse linéaire = base + offset.

Trouver la base grâce au sélecteur de segment

LDT - GDT

- Deux tables en RAM : GDT et LDT rassemblent les descripteurs de segments (8 bytes).
- **GDT** Global Descriptior Table (partagée par tous les process)
- **LDT** Local Descriptor Table (par process).

Les registres **gdtr** et **ldtr** contiennent les adresses de ces tables

sélecteur de segment

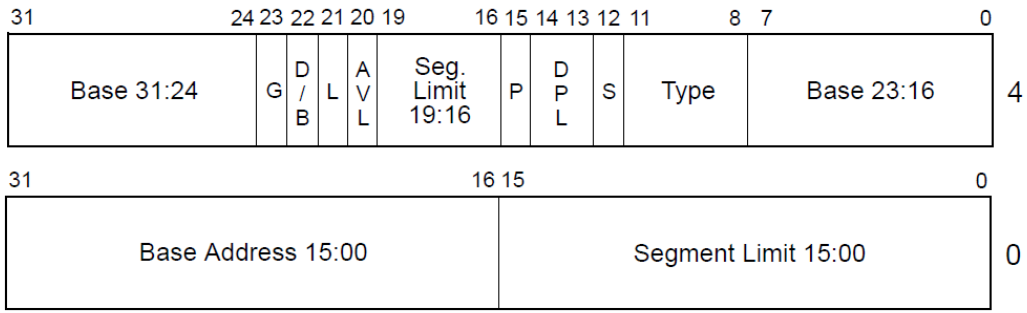
- index - 13 bits
- TI - 0/1 (GDT/LDT).
- RPL ou CPL (pour CS) - 2 bits (Requested/Current Privilege Level)

index sur 13 bits -> maximum 2^{13} descripteurs de segments par table

index -> adresse du descripteur

gdtr/ldtr + index*8

descripteur de segment



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Figure 3-8. Segment Descriptor

descripteur de segment

La position et taille

- Base - 32 bits (0 - 4GiB-1)
- Limite - 20 bits
- G - granularité 0-1 (byte - page)
  - 1MiB (2^{20} bytes)
  - 4GiB (2^{20} pages de 4KiB)

Sans pagination **Base** est une adresse physique

La position et taille

- Base - 32 bits (0 - 4GiB-1)
- Limite - 20 bits
- G - granularité 0-1 (byte - page)
  - 1MiB (2^{20} bytes)
  - 4GiB (2^{20} pages de 4KiB)

Sans pagination **Base** est une adresse physique

En présence de pagination, **Base** est une adresse dans l'espace d'adressage du programme

La taille maximum d'un segment est exprimée par la **limite** et le **bit de granularité**. Elle est de 4GiB maximum.

descripteur de segment

Les droits

- DPL - 2 bits (0-3), Descriptor Privilege Level , niveau de privilège minimum requis pour l'accès (0 plus haut, 3 plus bas).
- type - lecture/écriture

descripteur de segment

Swap

- P - présent
- A - accédé

Traduction d'adresse logique -> adresse linéaire

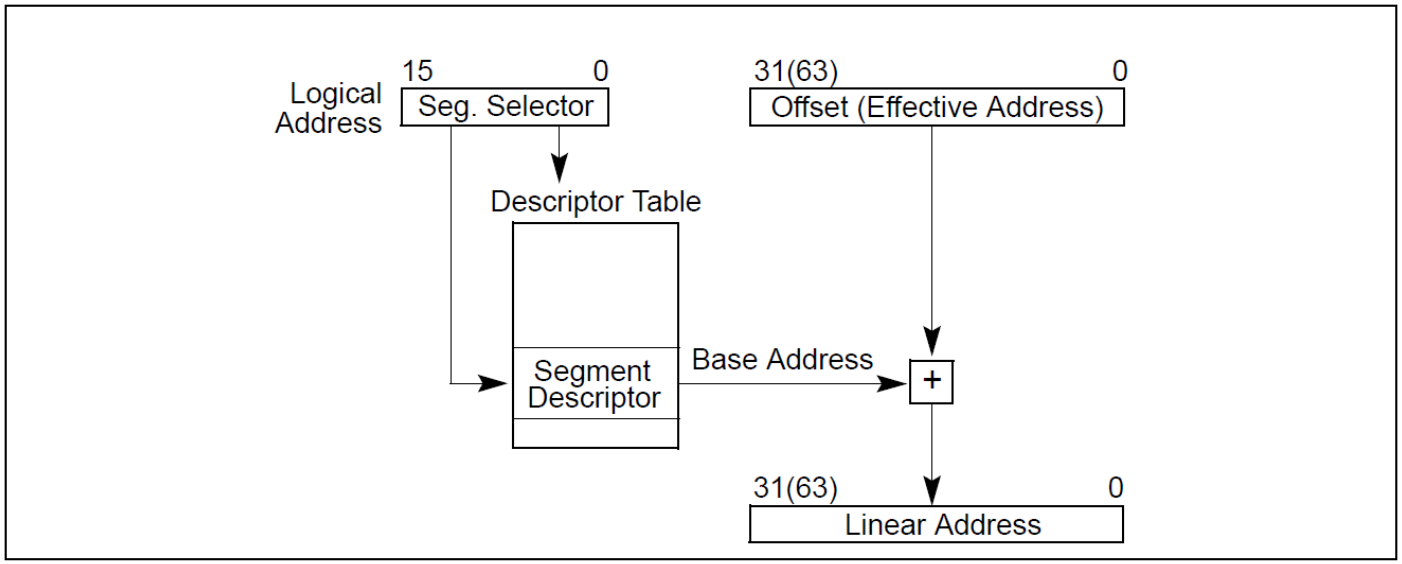


Figure 3-5. Logical Address to Linear Address Translation

segmentation performance

Performance et principe de localité Le **MMU** traduit l'adresse et chaque référence à la RAM requiert une lecture supplémentaire pour le descripteur dans la table !

```
MOV    ebx,[0x1000]
```

3 accès nécessaires !

Diminuer les accès mémoire ?

segmentation performance

Principe de localité :

- mémoriser les derniers descripteurs de segments utilisés (64 bits).
- Un sélecteur de segment a une partie visible (16 bits) et une partie cachée (registre de 8 bytes)
- Le descripteur est lu uniquement à chaque modification du sélecteur

protection

Au niveau de la segmentation :

- niveau de privilège - Rings.
- type d'accès (X,R,W)
- limite

protection

RING

- Les privilèges d'un code qui s'exécute sont marqués dans le sélecteur CS :
- CPL = Code Privilege Level (0 à 3)
- Les privilèges nécessaires pour accéder un segment de données sont dans le descripteur du segment :
- DPL = Descriptor Privilege Level (0 à 3)

protection

- exemples de vérifications :
- Privilèges insuffisants (provoque une exception de type **general protection fault**).
  - Code avec privilège utilisateur 3 (CPL = 3).
  - Données avec un niveau de privilège du noyau 0 (DPL = 0).
  - Et en général, dans les cas CPL > DPL

Dépassement de la limite du segment (provoque une exception de type **segmentation fault**).

Questions ?

images

toutes les images du chapitre segmentation sont issues de
intel 64 and I1-32 Architectures Software Developer's Manual (2011)

remerciements

remerciements à P.Bettens et M.Codutti
pour la mise en page :-) Mba

Crédits

Ces slides sont le support pour la présentation orale de l'activité d'apprentissage **SYS2** à la HE2B-ÉSI

Crédits Crédits

La distribution opensuse
du système d'exploitation GNU Linux.
LaTeX/Beamer comme système d'édition.
GNU make, rubber, pdfnup, ... pour les petites tâches.

Images et icônes

deviantart, flickr, The Noun Project