Systèmes d'Exploitation

Concepts et implémentation appliqués à l'architecture Intel x86

Abdelhamid Hadjeri, Anaïs Gantet

Support de slides fortement inspiré de celui de S. Duverger (tlssec 2015-2021)

TLS-SEC 2024-2025

1. Introduction
2. La phase de démarrage
3. Les modes opératoires
4. La segmentation (en mode privilégié)
   - Segmentation : motivations
   - Segmentation : Définition
   - Segmentation : utilisation
5. Les niveaux de privilèges (en mode privilégié)
6. Les interruptions et exceptions
7. La pagination (en mode privilégié)
8. Vue d'ensemble du mode protégé
9. OS et sécurité bas niveau
10. quiz
11. Conclusion

1/3 du module "Protection des systèmes d'exploitation"

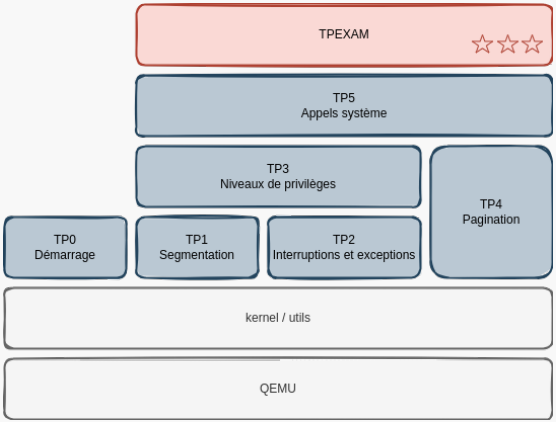
- 3 CM, 3 TP
- Avec cours sur sécurité Linux (E. Lacombe)
- Avec cours sur sécurité Windows (S. Peyrefitte + S. Garrelou ?)

1/3 du module ”Protection des systèmes d’exploitation”

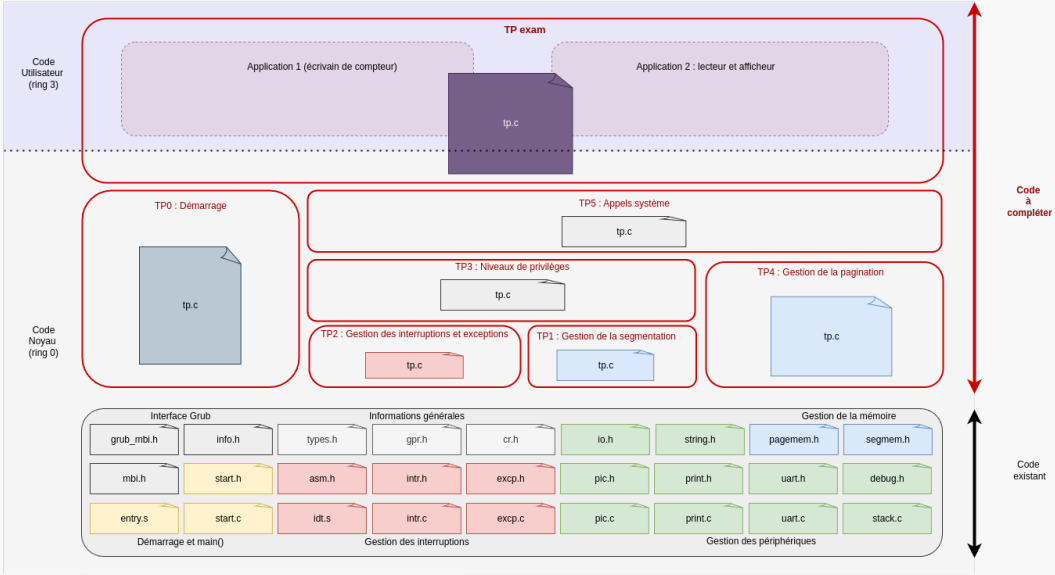
- 3 CM, 3 TP
- Avec cours sur sécurité Linux (E. Lacombe)
- Avec cours sur sécurité Windows (S. Peyrefitte + S. Garrelou ?)

Modalités d’évaluation

- Examen ”sur table”
- Projet de programmation `secos-ng/tp_exam` pour points bonus (max 8) sur la note de l’examen final



Architecture du projet SecOS


Code Utilisateur (ring 3)
Code Noyau (ring 0)
TP exam
Application 1 (écrivain de compteur)
Application 2 : lecteur et afficheur
tp.c
TP0 : Démarrage
tp.c
TP5 : Appels système
tp.c
TP3 : Niveaux de privilèges
tp.c
TP4 : Gestion de la pagination
tp.c
TP2 : Gestion des interruptions et exceptions
tp.c
TP1 : Gestion de la segmentation
tp.c
Code à compléter
Interface Grub
Informations générales
Gestion de la mémoire
grub_mbi.h
info.h
types.h
gpr.h
cr.h
io.h
string.h
pagemem.h
segmem.h
mbi.h
start.h
asm.h
intr.h
excp.h
pic.h
print.h
uart.h
debug.h
entry.s
start.c
idt.s
intr.c
excp.c
pic.c
print.c
uart.c
stack.c
Démarrage et main()
Gestion des interruptions
Gestion des périphériques
Code existant

Objectif

- aborder la programmation des OS
- en particulier du point de vue du CPU

Objectif

- aborder la programmation des OS
- en particulier du point de vue du CPU

Plan d'action

- Modes d'exécution du CPU (protégé essentiellement)
- Mémoire (segmentation, pagination)
- Exceptions/interruptions
- Appels systèmes
- Processus/tâches utilisateurs

D'un développeur d'OS

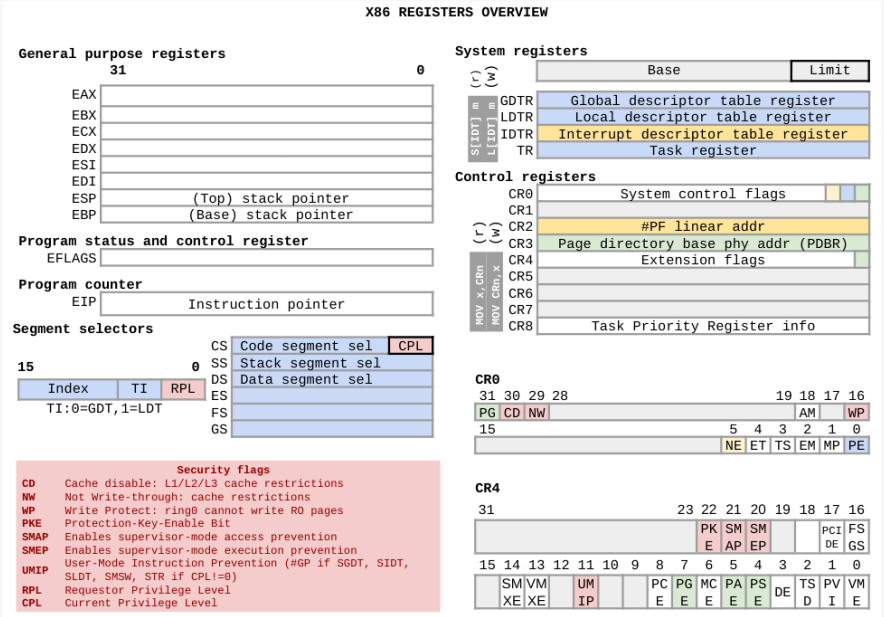
- Sur quels mécanismes matériels s'appuyer pour mettre en place de la ségrégation mémoire ?
  - Choix de la technologie
  - Choix de la stratégie d'allocation mémoire
- Quelles précautions prendre lors de l'implémentation de routines d'interruptions ?
- Choix d'utilisation des niveaux de privilèges ?

D'un développeur d'OS

- Sur quels mécanismes matériels s'appuyer pour mettre en place de la ségrégation mémoire ?
  - Choix de la technologie
  - Choix de la stratégie d'allocation mémoire
- Quelles précautions prendre lors de l'implémentation de routines d'interruptions ?
- Choix d'utilisation des niveaux de privilèges ?

Et en audit ?

- Comment l'OS gère la ségrégation mémoire entre noyau et applications ?
  - Utilisation ou non de la segmentation ?
  - Configuration des tables de segmentation ?
  - Configuration des tables de pagination ?
  - Allocation statique, dynamique ?
  - Zone de mémoires partagées ?
- Comment l'OS gère les signaux d'entrée/sortie ?
- Vulnérabilités dans les routines d'appels système exposés aux applications ?
- Quelle utilisation des niveaux de privilèges ?
- etc.


X86 REGISTERS OVERVIEW
General purpose registers
31
0
EAX
EBX
ECX
EDX
ESI
EDI
ESP
(Top) stack pointer
EBP
(Base) stack pointer
Program status and control register
EFLAGS
Program counter
EIP
Instruction pointer
Segment selectors
15
0
Index
TI
RPL
TI:0=GDT,1=LDT
CS
Code segment sel
CPL
SS
Stack segment sel
DS
Data segment sel
ES
FS
GS
Security flags
CD Cache disable: L1/L2/L3 cache restrictions
NW Not Write-through: cache restrictions
WP Write Protect: ring0 cannot write RO pages
PKE Protection-Key-Enable Bit
SMAP Enables supervisor-mode access prevention
SMEP Enables supervisor-mode execution prevention
UMIP User-Mode Instruction Prevention (#GP if SGDT, SIDT, SLDT, SMSW, STR if CPL!=0)
RPL Requestor Privilege Level
CPL Current Privilege Level
System registers
(r) (w)
Base
Limit
S[IDT] m
L[IDT] m
GDTR
Global descriptor table register
LDTR
Local descriptor table register
IDTR
Interrupt descriptor table register
TR
Task register
Control registers
(r) (w)
MOV x,CRn
MOV CRn,x
CR0
System control flags
CR1
CR2
#PF linear addr
CR3
Page directory base phy addr (PDBR)
CR4
Extension flags
CR5
CR6
CR7
CR8
Task Priority Register info
CR0
31 30 29 28
19 18 17 16
PG CD NW
AM
WP
15
5 4 3 2 1 0
NE ET TS EM MP PE
CR4
31
23 22 21 20 19 18 17 16
PKE SMAP SMEP
PCIDE FSGS
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SMXE VMXE UMIP PCE PGE MCE PAE PSE DE TSD PVI VME

État initial du CPU

- Mode réel, Boot Strap Processor (BSP)
- premières instructions :
  - `0xfffffff0` (`max_addr - max_insn_sz`)
  - la flash ROM du BIOS y est *mappée*
  - puis saute en `0xf0000` (*BIOS code area*)

État initial du CPU

- Mode réel, Boot Strap Processor (BSP)
- premières instructions :
  - `0xfffffff0` (`max_addr - max_insn_sz`)
  - la flash ROM du BIOS y est *mappée*
  - puis saute en `0xf0000` (*BIOS code area*)

Basic Input Output System (B.I.O.S.)

- Responsable de l'initialisation des composants essentiels du système
  - Contrôleur de RAM
  - Table de gestion des interruptions
  - Tables ACPI (gestion de l'énergie entre autre)

Secteur d'amorçage (boot sector)

Le BIOS maintient une liste de périphériques de démarrage

- HDD, disquette, clé USB, ...

Secteur d'amorçage (boot sector)

Le BIOS maintient une liste de périphériques de démarrage

- HDD, disquette, clé USB, ...

Boot Sector

- contenu dans 512 octets, se termine par "\x55\xAA" (marqueur)
- chargé par le BIOS en `0x7c00`
- chargera en mémoire le Boot Loader

Exemple : cas d'un HDD

- Tables de partitions, *boot flag*
- Master Boot Record, premier secteur du disque CHS(0,0,0)
- Volume Boot Record, premier secteur d'une partition bootable

```
# dd if=/dev/sda bs=512 count=1 of=boot_sector
# objdump -D -b binary -m i386 boot_sector
00000000 <.data>:
   0: eb 63                  jmp    0x65
   2: 90                     nop
[...]
  65: fa                     cli
  66: 90                     nop
  67: 90                     nop
  68: f6 c2 80               test   $0x80,%dl
  6b: 74 05                  je     0x72
  6d: f6 c2 70               test   $0x70,%dl
  70: 74 02                  je     0x74
  72: b2 80                  mov    $0x80,%dl
  74: ea 79 7c 00 00 31 c0   ljmp   $0xc031,$0x7c79
[...]
 1fe: 55 aa
```

Boot Loader

- son but est de charger le noyau de l'OS
- simplifier son démarrage (mode protégé, ...)
- plus gros qu'un boot sector (mini-os parfois)
- passerelle entre le noyau et le BIOS :
  - détecte les périphériques de stockage
  - parcourt les systèmes de fichiers
  - récupère les *system map* (ie. taille de la RAM)

```
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000dc000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000affefffe] usable
[    0.000000] BIOS-e820: [mem 0x00000000afff0000-0x00000000afffefff] ACPI data
[    0.000000] BIOS-e820: [mem 0x00000000afffe000-0x00000000afffffff] ACPI NVS
[    0.000000] BIOS-e820: [mem 0x00000000fd000000-0x00000000fd7fffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000ffc00000-0x00000000ffffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000014fffffff] usable
```

- LILO, historique et minimaliste
- u-Boot, dans l'embarqué (ARM, ...)
- GRUB, spécification multi-boot, très courant

Les différents modes

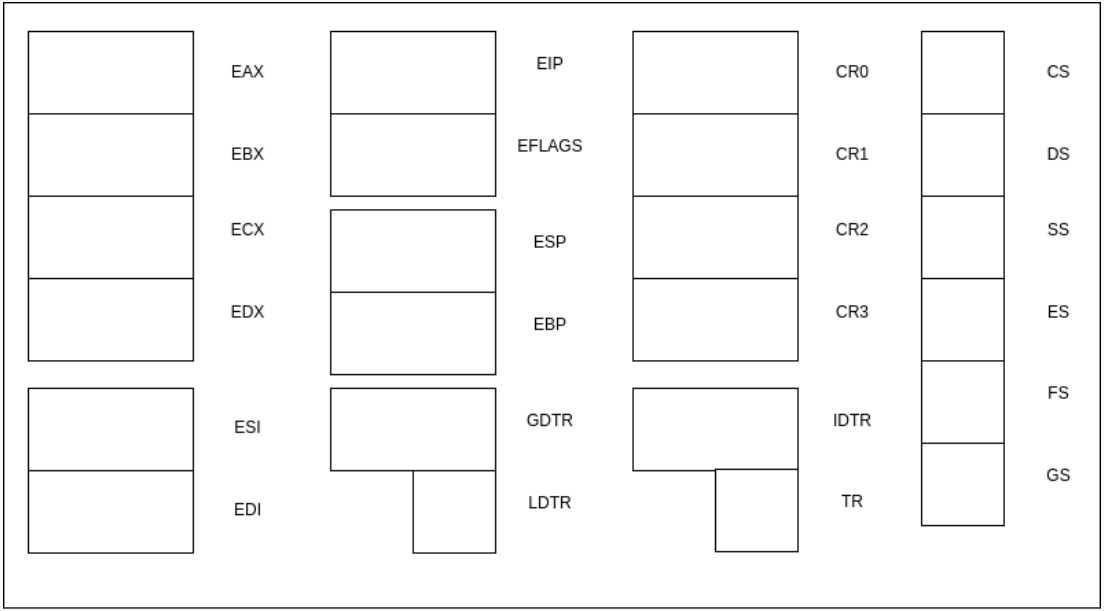
- réel
- virtuel 8086
- system management (SMM)
- protégé
- IA32e (x86-64)
- virtualisé (VMX)

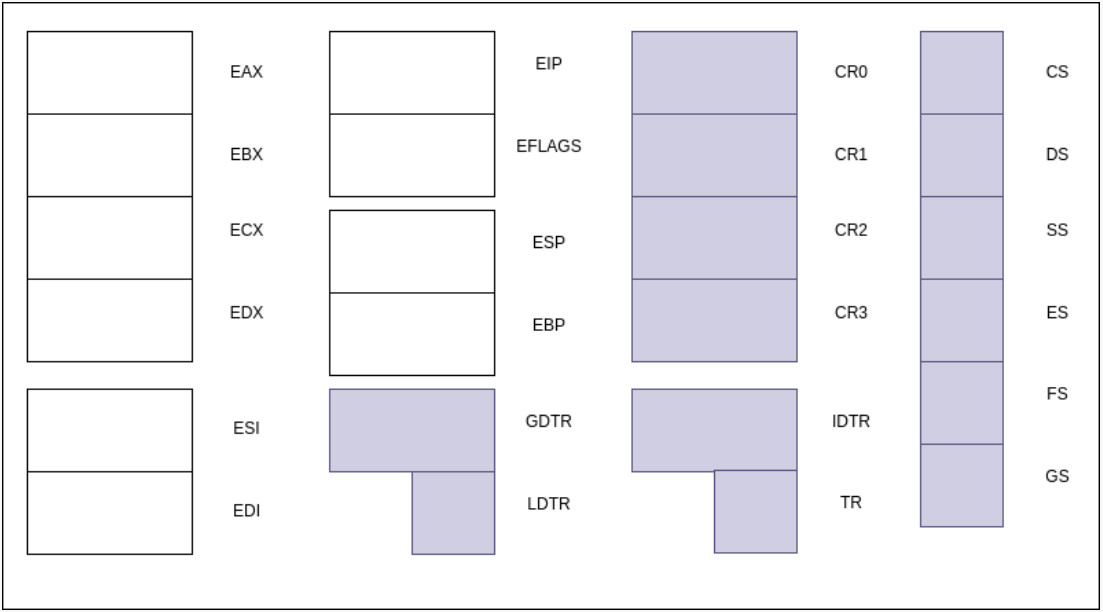
Les différents modes

- réel
- virtuel 8086
- system management (SMM)
- protégé
- IA32e (x86-64)
- virtualisé (VMX)

Leurs conséquences

- le cpu est dans un seul mode à la fois
- transitions de modes peuvent être complexes
- environnement d'exécution est très différent selon le mode
  - quantité de mémoire adressable
  - modes d'adressages (niveau instruction assembleur)
  - gestion des interruptions/exceptions
  - instructions invalides/non autorisées


EAX
EBX
ECX
EDX
ESI
EDI
EIP
EFLAGS
ESP
EBP
GDTR
LDTR
CR0
CR1
CR2
CR3
IDTR
TR
CS
DS
SS
ES
FS
GS


EAX
EBX
ECX
EDX
ESI
EDI
EIP
EFLAGS
ESP
EBP
GDTR
LDTR
CR0
CR1
CR2
CR3
IDTR
TR
CS
DS
SS
ES
FS
GS

Registres généraux

- multi-usage `[r,e]ax, [r,e]bx, [r,e]cx, [r,e]dx, [r,e]si, [r,e]di`
- pour la pile `[r,e]sp, [r,e]bp`
- en 64 bits `r8-r15`
  - selon le mode d'exécution (16,32,64) et/ou l'usage de préfixe
  - ax = 16 bits, eax = 32 bits, rax = 64 bits

Registres de segments (sélecteurs)

cs, ss, ds, es, fs, gs

```
        mov       [ax], 0xdead     DS:ax = 0xdead
        mov   fs:[ax], 0xdead     FS:ax = 0xdead
        push       ax              SS:sp = ax
        jump       ax              saute en CS:ax
        movs                       ES:di = DS:si
```

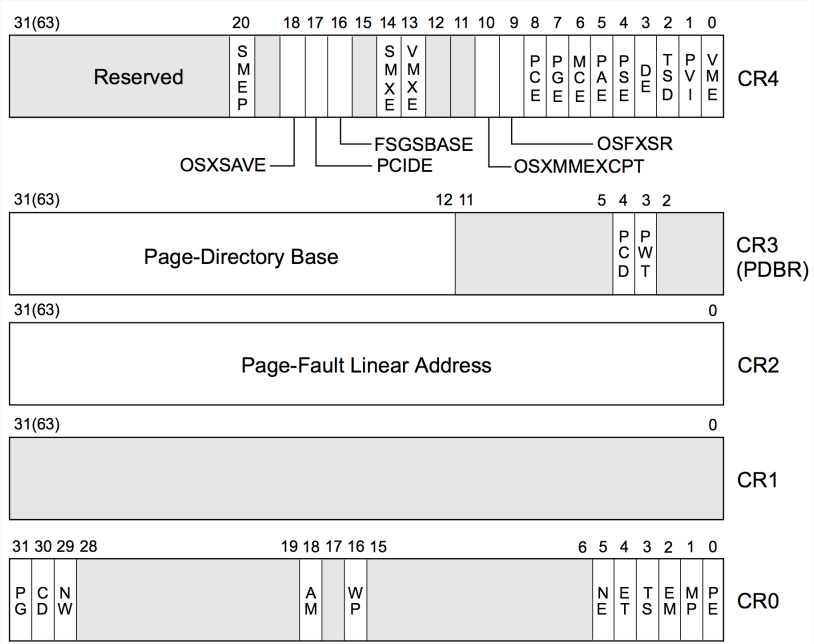
Registres de segments (sélecteurs)

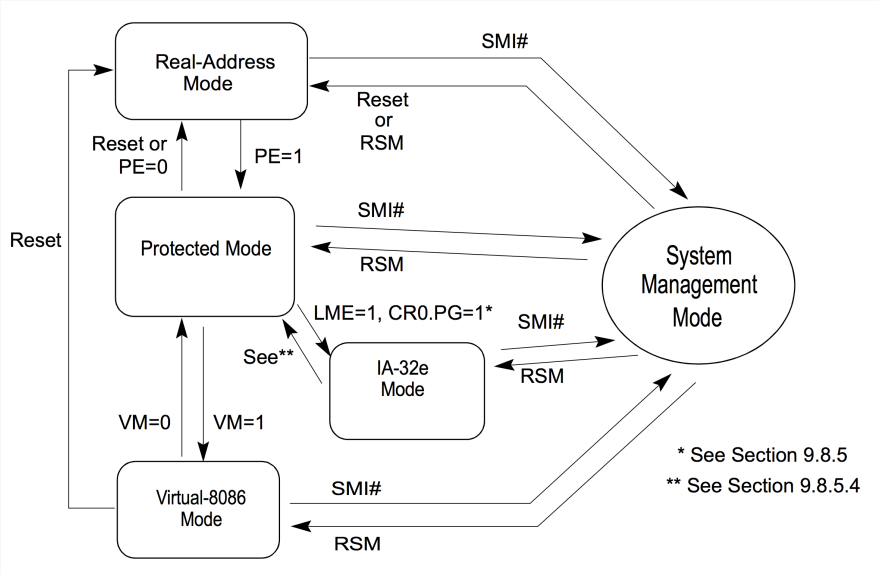
`cs, ss, ds, es, fs, gs`

```
mov       [ax], 0xdead     DS:ax = 0xdead
mov   fs:[ax], 0xdead     FS:ax = 0xdead
push       ax              SS:sp = ax
jump       ax              saute en CS:ax
movs                       ES:di = DS:si
```

Registres systèmes

- controle : `cr0, cr2, cr3, cr4`
  - permettent d'activer des fonctionnalités du cpu
  - dont le passage mode réel/protégé (`cr0.pe`)
  - d'activer la pagination (`cr0.pg, cr3`)
  - ...
- segmentation : `gdtr, idtr, ltr, tr`
- model specific : MSRs


31(63)
20
18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved
SMEP
SMXE
VMXE
PCE
PGE
MCE
PAE
PSE
DE
TSD
PVI
VME
CR4
OSXSAVE
PCIDE
FSGSBASE
OSXMMEXCPT
OSFXSR
31(63)
12 11
5 4 3 2
Page-Directory Base
PCD
PWT
CR3
(PDBR)
31(63)
0
Page-Fault Linear Address
CR2
31(63)
0
CR1
31 30 29 28
19 18 17 16 15
6 5 4 3 2 1 0
PG
CD
NW
AM
WP
NE
ET
TS
EM
MP
PE
CR0
Reserved


Real-Address Mode
SMI#
Reset or RSM
Reset or PE=0
PE=1
Reset
Protected Mode
SMI#
RSM
System Management Mode
LME=1, CR0.PG=1*
SMI#
See**
IA-32e Mode
RSM
VM=0
VM=1
* See Section 9.8.5
** See Section 9.8.5.4
Virtual-8086 Mode
SMI#
RSM

Historique

- premiers x86 : 8086, 80186, 80286 (pseudo mode protégé)
- état initial du cpu
- permet interaction facile avec le BIOS
- mode d'exécution 16 bits, adressage de 20 bits
- pas de protections (niveaux de privilèges)

Historique

- premiers x86 : 8086, 80186, 80286 (pseudo mode protégé)
- état initial du cpu
- permet interaction facile avec le BIOS
- mode d'exécution 16 bits, adressage de 20 bits
- pas de protections (niveaux de privilèges)

Premiers OS de l'IBM/PC compatible

- MS-DOS
- OS/2
- Windows < 3.1

Des segments de 64K

- adressage mémoire en mode 16 bits

```
mov ax, 0x1234
mov [ax], 0xdead
```

Des segments de 64K

- adressage mémoire en mode 16 bits

```
mov ax, 0x1234
mov [ax], 0xdead
```

- registre d'offset sur 16 bits ⇒ 2^{16} = 64KB
- bus d'adressage sur 20 bits ⇒ 2^{20} = 1MB
- usage des registres de segments
- calcul du cpu : $adresse = selecteur * 16 + offset$

```
mov ds, 0x100
mov [0x234x], 0xdead  ---> 0x100<<4 + 0x234 = 0x1234
```

Des segments de 64K

- adressage mémoire en mode 16 bits

```
mov ax, 0x1234
mov [ax], 0xdead
```

- registre d'offset sur 16 bits ⇒ 2^{16} = 64KB
- bus d'adressage sur 20 bits ⇒ 2^{20} = 1MB
- usage des registres de segments
- calcul du cpu : $adresse = selecteur * 16 + offset$

```
mov ds, 0x100
mov [0x234x], 0xdead  ---> 0x100<<4 + 0x234 = 0x1234

mov ds, 0xffff
mov [0xffff], 0xdead  ---> 0xffff<<4 + 0xffff = 0x10ffef

 > 1MB ! wrap around
```

Montre rapidement ses limites

- les OS deviennent plus gros/complexes/gourmands
- arrivée du multimédia
- limitation de la mémoire à 1MB inacceptable
- si une application plante, tout plante !

Montre rapidement ses limites

- les OS deviennent plus gros/complexes/gourmands
- arrivée du multimédia
- limitation de la mémoire à 1MB inacceptable
- si une application plante, tout plante !
- ... passage en mode protégé

System Management Mode

- mode le plus privilégié
- utilisé par les firmwares (BIOS)
- configuré au boot (SMRAM)
- accessible via une System Management Interrupt (SMI)
- généralement difficilement accessible

System Management Mode

- mode le plus privilégié
- utilisé par les firmwares (BIOS)
- configuré au boot (SMRAM)
- accessible via une System Management Interrupt (SMI)
- généralement difficilement accessible

VMX

- gestion de la virtualisation matérielle
- 2 modes d'exécution
  - `vmx-root`, pour l'hyperviseur (VMM)
  - `vmx-nonroot`, pour la(es) machine(s) virtuelle(s) (VM)
- permet de filtrer la plupart des évènements systèmes :
  - instructions sensibles
  - interruptions/exceptions
  - la gestion de la mémoire
  - accès aux périphériques
- proposer une vision contrôlée de la machine

Adressage 32 bits mais pas que

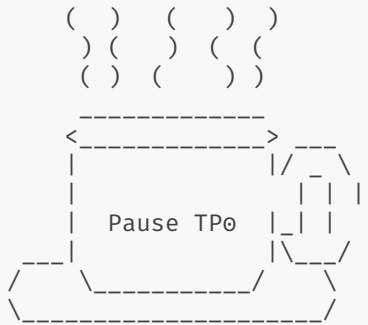
- arrivé avec le 80386 (d'où les `i386` ...)
- support des modes 16 et 32 bits
- émulation du mode réel (v8086)

Adressage 32 bits mais pas que

- arrivé avec le 80386 (d'où les `i386` ...)
- support des modes 16 et 32 bits
- émulation du mode réel (v8086)

Qui dit protégé ... dit

- segmentation, pagination
- niveaux de privilèges
- changement de tâche matériel (*task switch*)


Pause TP0

La segmentation (en mode privilégié)

Segmentation : motivations

Espace mémoire en 32 bits

- Espace linéaire de 32 bits $\Rightarrow 2^{32}$ = 4GB
- Pendant un temps supérieur à la RAM installée
- Segmentation possible de cet espace
- Adressage relatif au début d'un segment (adresse logique)
- Permet une isolation des portions de l'espace linéaire

Espace mémoire en 32 bits

- Espace linéaire de 32 bits $\Rightarrow 2^{32}$ = 4GB
- Pendant un temps supérieur à la RAM installée
- Segmentation possible de cet espace
- Adressage relatif au début d'un segment (adresse logique)
- Permet une isolation des portions de l'espace linéaire

La segmentation

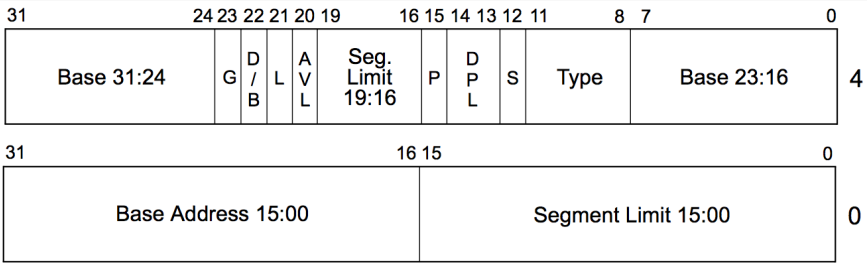
- Extension du principe de segments du mode réel
- Modèles *flat*/*protected flat*/*multi-segment*
- Des segments avec des propriétés : type, taille, droits

La segmentation (en mode privilégié)

Segmentation : Définition

Le descripteur de segment

- base et limite
- type de segment
  - code (`X`,`RX`), data (`R`,`W`,`RW`)
  - système : TSS, Task Gate, Interrupt Gate, Call Gate



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

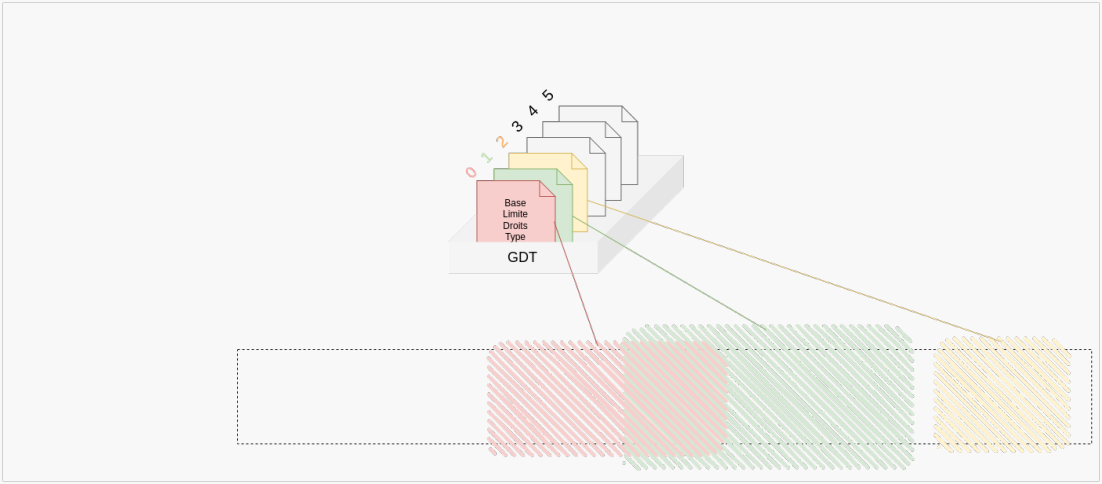
Table 3-1. Code- and Data-Segment Types

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

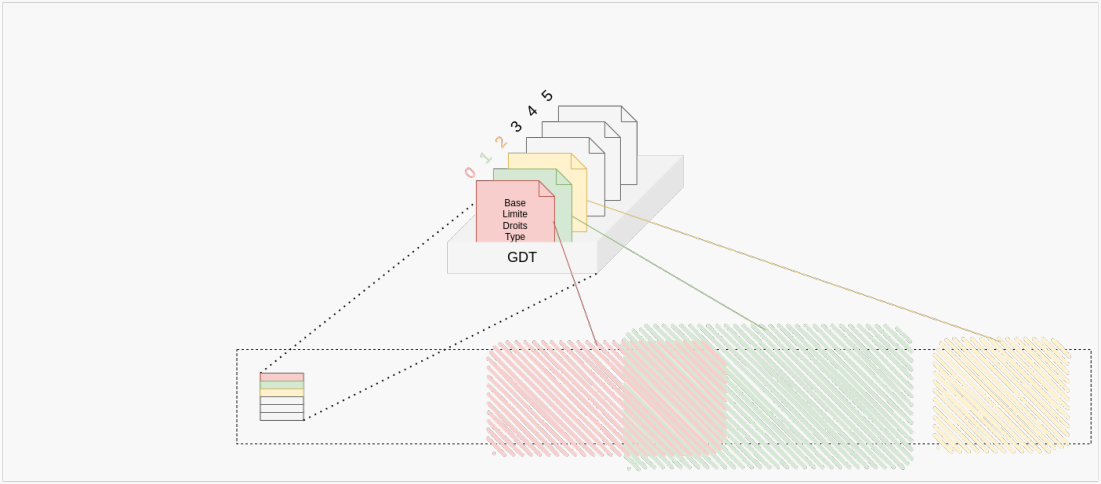
Tables de descripteurs

- Global Descriptor Table (GDT), `gdtr`
  - unique pour chaque coeur d'un cpu
  - 8192 entrées max, première entrée vide
- Local Descriptor Table (LDT), `ldtr`
  - locale à une tâche
  - son propre descripteur se trouve dans la GDT
- Interrupt Descriptor Table (IDT), `idtr`
  - unique pour chaque coeur d'un cpu
  - utilisée pour les exceptions/interruptions

Table de descripteurs de segments



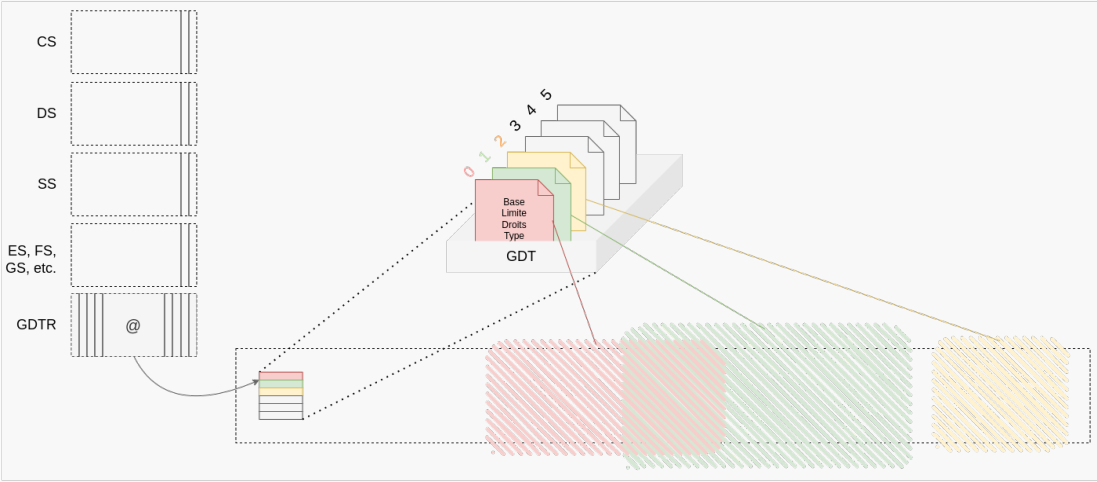
... Stockée en mémoire



La segmentation (en mode privilégié)

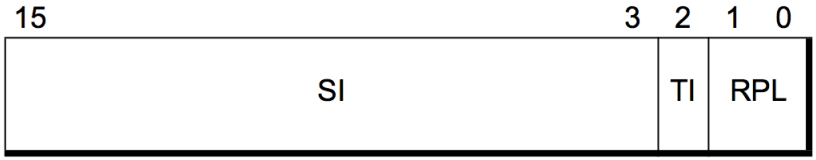
Segmentation : utilisation

Le registre GDTR

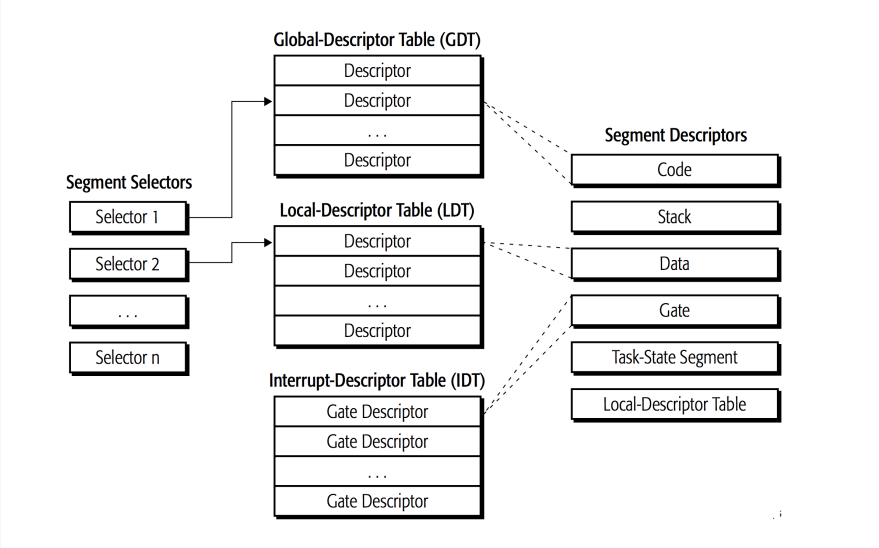


Le sélecteur de segment

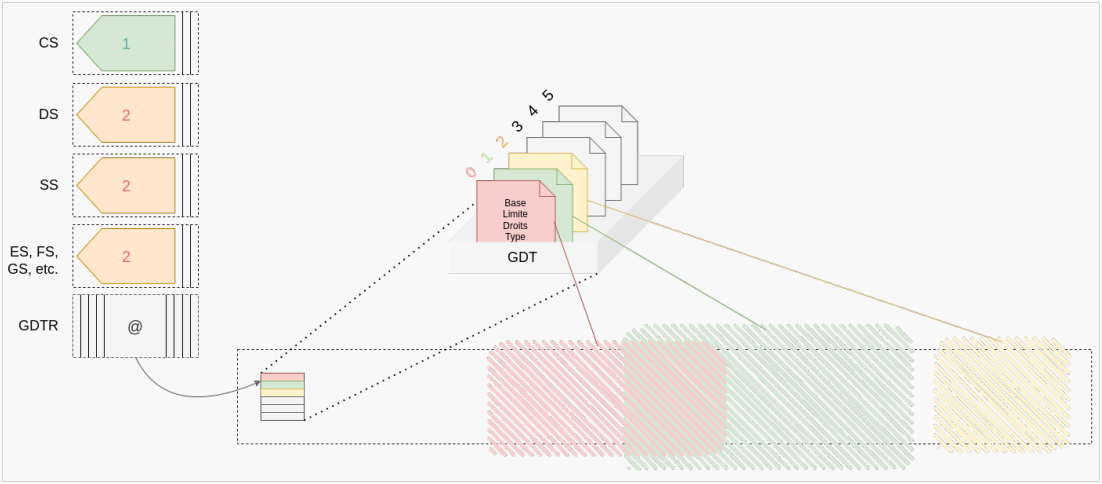
- utilisé par les registres de segment
- permet l'accès à un descripteur
- chaque sélecteur définit
  - un niveau de privilège
  - une table (locale ou globale)
  - un indice de descripteur dans cette table
- partie visible au programmeur
- le cpu charge le descripteur dans des parties cachées du registre



Bits	Mnemonic	Description	R/W
15-3	SI	Selector Index	R/W
2	TI	Table Indicator	R/W
1-0	RPL	Requestor Privilege Level	R/W


Global-Descriptor Table (GDT)
Descriptor
Descriptor
. . .
Descriptor
Segment Selectors
Selector 1
Selector 2
. . .
Selector n
Local-Descriptor Table (LDT)
Descriptor
Descriptor
. . .
Descriptor
Interrupt-Descriptor Table (IDT)
Gate Descriptor
Gate Descriptor
. . .
Gate Descriptor
Segment Descriptors
Code
Stack
Data
Gate
Task-State Segment
Local-Descriptor Table

Mise à jour des sélecteurs de segments



Comment demander au CPU de mettre à jour les sélecteurs ?

Comment demander au CPU de mettre à jour les sélecteurs ?

- Pour les sélecteurs DS, SS, ES, FS, GS
  - Un simple `mov sel_val ds` suffit

Comment demander au CPU de mettre à jour les sélecteurs ?

- Pour les sélecteurs DS, SS, ES, FS, GS
  - Un simple `mov sel_val ds` suffit
- Pour CS
  - L'architecture x86 interdit l'utilisation de `mov` pour sa mise à jour (cf. documentation Intel) :(

MOV instruction
Protected mode
#UD If attempt is made to load the CS register.

Comment demander au CPU de mettre à jour les sélecteurs ?

- Pour les sélecteurs DS, SS, ES, FS, GS
  - Un simple `mov sel_val ds` suffit
- Pour CS
  - L'architecture x86 interdit l'utilisation de `mov` pour sa mise à jour (cf. documentation Intel) :(

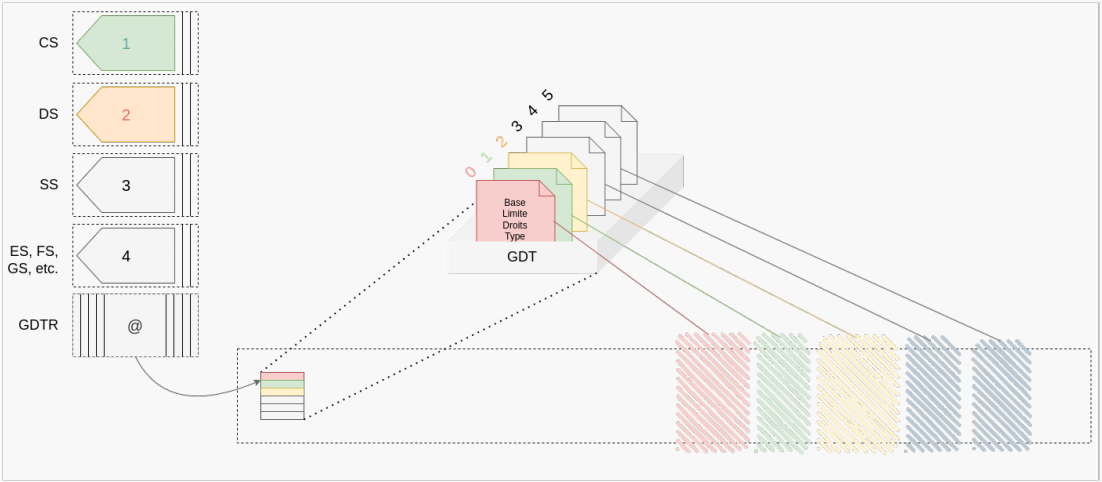
MOV instruction
Protected mode
#UD If attempt is made to load the CS register.

- Solution : `ljump sel_val eip_val` :)

Exemple de `far jump`

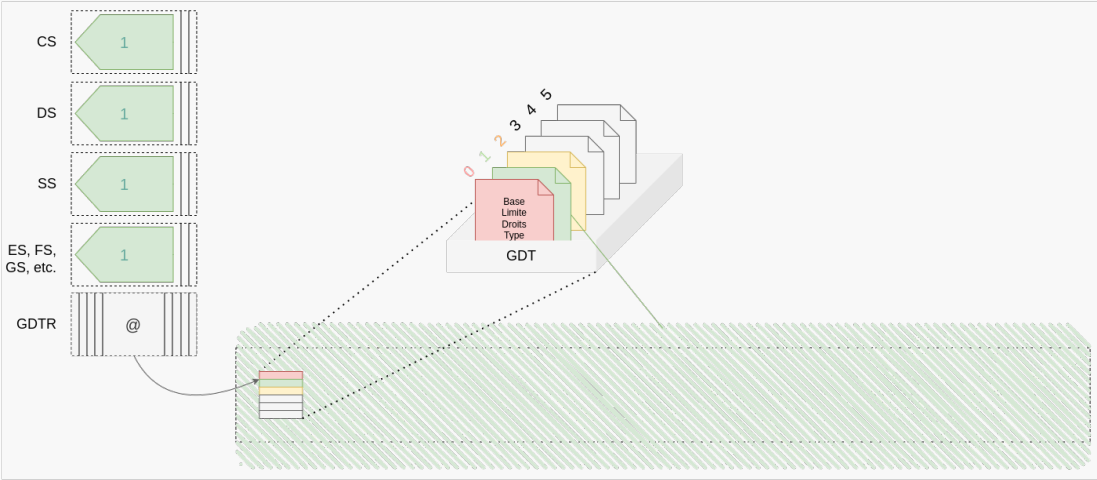
- encodage x86 permet de faire un saut long
- donne le sélecteur et l'offset
- le sélecteur est chargé dans `cs` et l'offset dans `eip`

Le modèle "Multi-segments" : un segment disjoint par sélecteur



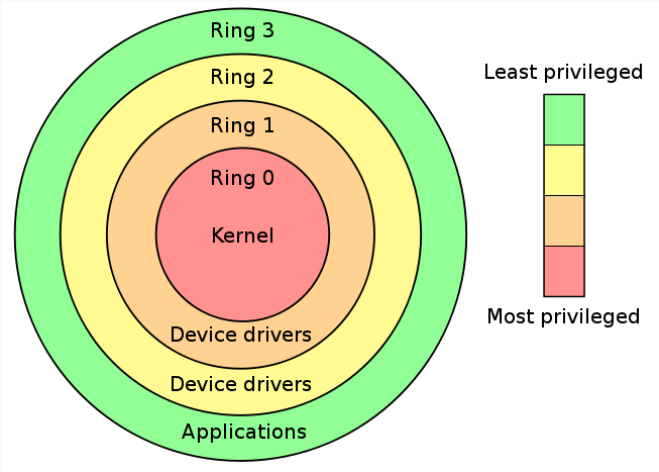
Multi-Segments

Le modèle "Flat" : tous les sélecteurs pointent vers des descripteurs couvrant la totalité de l'espace mémoire adressable



Flat

- mode réel n'avait qu'un seul niveau (*ring 0*)
- mode protégé introduit des anneaux (*ring level*)
- permet d'isoler des composants logiciels de niveau
  - de confidentialité différents
  - d'intégrité différents



Identification du niveau de privilèges

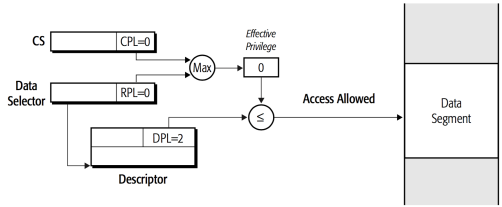
- CPL, Current Privilege Level, contenu dans `cs`
- RPL, Requestor Privilege Level, au niveau du sélecteur
- DPL, Descriptor Privilege Level, au niveau du descripteur
- potentiellement très complexe
  - interprétation différente selon le type de segment
  - transferts inter-segments (Gate)

Identification du niveau de privilèges

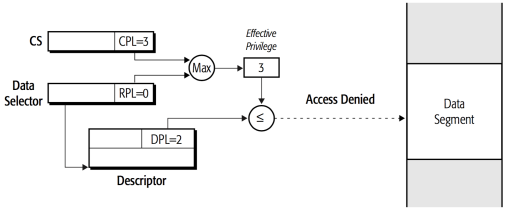
- CPL, Current Privilege Level, contenu dans `cs`
- RPL, Requestor Privilege Level, au niveau du sélecteur
- DPL, Descriptor Privilege Level, au niveau du descripteur
- potentiellement très complexe
  - interprétation différente selon le type de segment
  - transferts inter-segments (Gate)

Changements de niveau de privilèges

- on ne peut accéder à un segment de données plus privilégié
- on ne peut charger un descripteur de code qu'à niveau de privilège équivalent
  - aussi bien plus faible que plus privilégié
  - passer par des *call gate* ou *interrupt gate*
  - cas classique des appels systèmes (ring 3 vers ring 0)
  - historiquement on passait par une interruption (ie. `int 0x80`)



Example 2: Privilege Check Passes



Example 1: Privilege Check Fails

Des descripteurs sécurisants

- les niveaux de privilèges isolent les composants : Qui ?
- les bases et limites aussi à leur manière : Où/Combien ?
- les attributs également (read, write, ...) : Quoi/Comment ?

Des descripteurs sécurisants

- les niveaux de privilèges isolent les composants : Qui ?
- les bases et limites aussi à leur manière : Où/Combien ?
- les attributs également (read, write, ...) : Quoi/Comment ?

Du point de vue de la sécurité

- simulation d'une architecture harvard (contre von Neumann)
- imaginez un descripteur par buffer ? anti-overflow
- Linux Kernel PaX protection : SEGMEXEC
  `https://pax.grsecurity.net/docs/segmexec.txt`

Des descripteurs sécurisants

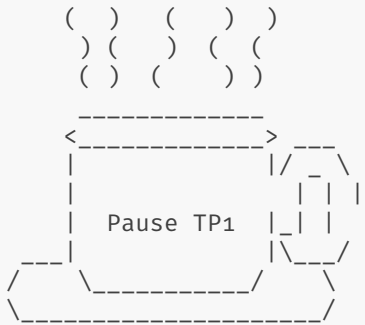
- les niveaux de privilèges isolent les composants : Qui ?
- les bases et limites aussi à leur manière : Où/Combien ?
- les attributs également (read, write, ...) : Quoi/Comment ?

Du point de vue de la sécurité

- simulation d'une architecture harvard (contre von Neumann)
- imaginez un descripteur par buffer ? anti-overflow
- Linux Kernel PaX protection : SEGMEXEC
  `https://pax.grsecurity.net/docs/segmexec.txt`

Et pourtant ...

- les OS modernes n'utilisent pas la segmentation (modèle flat)
- quasiment disparue en 64 bits
- basent leur sécurité sur la pagination


Pause TP1

Intérêt

- éviter l'attente active et bloquer le CPU
- apporter une composante événementielle à l'OS
- être réveillé à la demande
  - appui sur une touche clavier
  - arrivée d'un paquet réseau
  - écoulement d'un quantum de temps

Les sources d'exceptions

- générées par le CPU, en cas d'erreurs (#GP, #NP)
- générées par du code, (#BP)
- différentes natures
  - *fault*, on la gère et on re-exécute l'instruction
  - *trap*, on la gère et on continue après l'instruction
  - *abort*, non récupérable
- priorités entre exceptions générées simultanément
- code d'erreur accompagne parfois les exceptions

Les sources d'interruptions

- générées par les périphériques
- ou par du code (`int 0x80`)
- différentes natures
  - *irq*, pour les périphériques
  - *smi*, par le mode SMM
  - *nmi*, par des périphériques souvent liés à l'ACPI
  - *ipi*, entre coeurs d'un CPU, entre CPUs (*smp*)
- certaines peuvent être masquées (via l'instruction `cli`)
- une fois masquées, le CPU est in-interruptible
- sauf dans le cas des *nmi* (Non Maskable Interrupts)

General Protection Fault :#GP

- survient dans de nombreux cas
  - erreurs liées à la segmentation (taille, droits, pvl)
  - configuration invalide des registres de controle, MSRs
- du type *fault*
- fournit un code d'erreur
  - sélecteur de segment fautif, si erreur de segmentation
  - ou 0 pour les autres erreurs

General Protection Fault :#GP

- survient dans de nombreux cas
  - erreurs liées à la segmentation (taille, droits, pvl)
  - configuration invalide des registres de controle, MSRs
- du type *fault*
- fournit un code d'erreur
  - sélecteur de segment fautif, si erreur de segmentation
  - ou 0 pour les autres erreurs

Break Point :#BP

- survient lorsque l'instruction `int3` est exécutée
- du type *trap*
- pas de code d'erreur

Invalid Opcode :#UD

- survient lorsqu'une instruction est invalide
- du type *fault*
- pas de code d'erreur

Invalid Opcode :#UD

- survient lorsqu'une instruction est invalide
- du type *fault*
- pas de code d'erreur

Divide Error :#DE

- survient lorsque l'on effectue une division par zéro
- du type *fault*
- pas de code d'erreur

Concerne les erreurs liées à la pagination

- page non présente (`pte.p/pde.p = 0`)
- accès read/write
- accès user/supervisor

Concerne les erreurs liées à la pagination

- page non présente (`pte.p/pde.p = 0`)
- accès read/write
- accès user/supervisor

Code d'erreur et CR2

- code d'erreur indique la nature de la faute
- le registre `cr2` contient l'adresse fautive
- parfois différente de l'endroit où la faute est générée

```
0x804801e    mov  [0x1234], eax
```

- `cr2 = 0x1234`, faute générée en `0x804801e`

Fonctionnement

- qui dit interruption, dit reprise d'exécution
- nécessaire de sauvegarder l'état du CPU au moment de l'interruption
- on va parler
  - de pile d'interruption
  - d'Interrupt Gate (descripteur de segment d'interruption)
  - d'appels systèmes (implémentation historique)

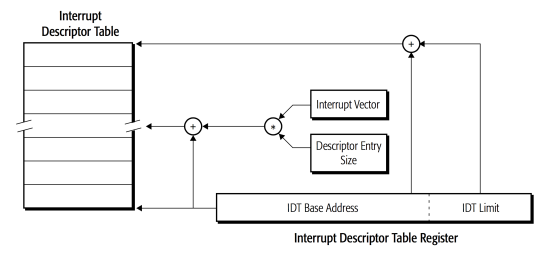
Fonctionnement

- qui dit interruption, dit reprise d'exécution
- nécessaire de sauvegarder l'état du CPU au moment de l'interruption
- on va parler
  - de pile d'interruption
  - d'Interrupt Gate (descripteur de segment d'interruption)
  - d'appels systèmes (implémentation historique)

Comme pour la segmentation

- les gestionnaires (*handlers*) sont placés dans une table
- on parle d'Interrupt Descriptor Table (IDT)
- ressemble à une GDT, avec des descripteurs spécifiques
- localisée grace au registre `idtr`

- les descripteurs référencent des segments de code et un *handler*
- indiquent également le DPL nécessaire pour y accéder (généralement ring 0)
- sauf le *handler* d'appels systèmes qui sera en ring 3 (linux 0x80)



Gate Descriptor (32-bit):

63 48	47	46 45	44	43 40	39 32
Offset	**P**	**DPL**	0	**Gate Type**	Reserved
31 16		1 0		3 0	

31 16	15 0
Segment Selector	**Offset**
15 0	15 0

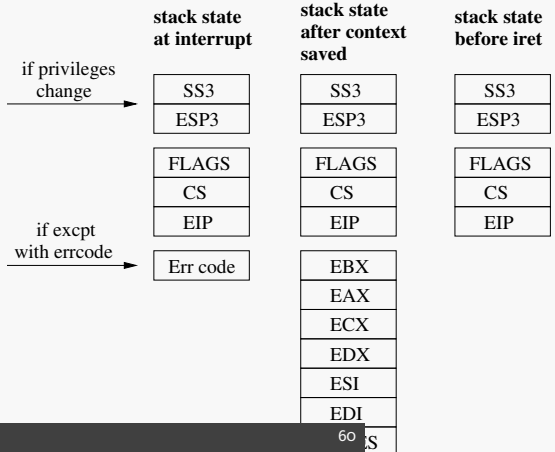
- **Offset:** A 32-bit value, split in two parts. It represents the address of the entry point of the **Interrupt Service Routine**.
- **Selector:** A **Segment Selector** with multiple fields which must point to a valid code segment in your **GDT**.
- **Gate Type:** A 4-bit value which defines the type of gate this **Interrupt Descriptor** represents. There are five valid type values:
  - **0b0101** or **0x5**: Task Gate, note that in this case, the **Offset** value is unused and should be set to zero.
  - **0b0110** or **0x6**: 16-bit Interrupt Gate
  - **0b0111** or **0x7**: 16-bit Trap Gate
  - **0b1110** or **0xE**: 32-bit Interrupt Gate
  - **0b1111** or **0xF**: 32-bit Trap Gate
- **DPL:** A 2-bit value which defines the **CPU Privilege Levels** which are allowed to access this interrupt via the **INT** instruction. Hardwa
- **P:** Present bit. Must be set (**1**) for the descriptor to be valid.

Le changement de contexte

- un contexte indique un état du CPU à un instant précis
- cela implique
  - le niveau de privilèges courant
  - la valeur des registres généraux (`eax, ebx, ... esp`)
  - le registre d'état (`EFLAGS`)
- dans le cas de l'arrivée d'une interruption/exception, on peut potentiellement changer de niveau de privilèges (ring 3 ⇒ ring 0)
- le CPU sauvegarde pour nous les informations du niveau de provenance
- le noyau fait le reste

En trois étapes

- au moment de l'interruption, le CPU sauvegarde le minimum et détermine s'il y a eu changement de niveau de privilèges
- avant de traiter l'interruption, l'OS sauvegarde ce qu'il pense être nécessaire de restaurer
- au retour de l'interruption, l'OS utilise une instruction (`iret`) spécifique



Comment le CPU sait où sauvegarder ?

- dans le cas d'une transition ring 3 ⇒ ring 0
  - le CPU cherche à changer de pile (registre `esp`)
  - sauver les informations du ring 3 dans le ring de destination
  - où trouver la nouvelle valeur à mettre dans `esp` ?

Comment le CPU sait où sauvegarder ?

- dans le cas d'une transition ring 3 ⇒ ring 0
  - le CPU cherche à changer de pile (registre `esp`)
  - sauver les informations du ring 3 dans le ring de destination
  - où trouver la nouvelle valeur à mettre dans `esp` ?
- grâce au TSS (Task State Segment)

Comment le CPU sait où sauvegarder ?

- dans le cas d'une transition ring 3 ⇒ ring 0
  - le CPU cherche à changer de pile (registre `esp`)
  - sauver les informations du ring 3 dans le ring de destination
  - où trouver la nouvelle valeur à mettre dans `esp` ?
- grâce au TSS (Task State Segment)

Changement de tâche matériel

- descripteur de segment spécifique dans la GDT
- dont l'index est chargé dans le registre `tr`
- mis à jour par l'OS à chaque changement de tâche
- en particulier son champ `esp0`

Comment le CPU sait où sauvegarder ?

- dans le cas d'une transition ring 3 ⇒ ring 0
  - le CPU cherche à changer de pile (registre `esp`)
  - sauver les informations du ring 3 dans le ring de destination
  - où trouver la nouvelle valeur à mettre dans `esp` ?
- grâce au TSS (Task State Segment)

Changement de tâche matériel

- descripteur de segment spécifique dans la GDT
- dont l'index est chargé dans le registre `tr`
- mis à jour par l'OS à chaque changement de tâche
- en particulier son champ `esp0`
- consulté par le CPU à chaque changement de privilèges
- initialement utilisé pour effectuer des changements de tâches matériel
- pas utilisé dans les OS modernes

Historiquement

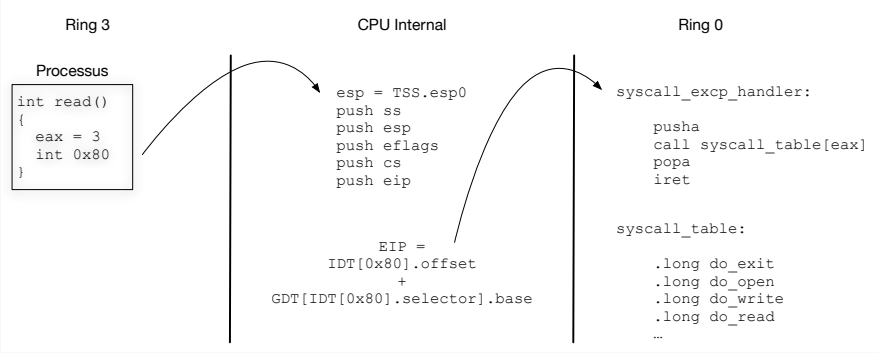
- géré grâce à une interruption accessible depuis le ring 3
- Interrupt Gate avec DPL=3 dans l'IDT
- les programmes utilisent `int N`
- où N correspond à l'indice dans l'IDT du descripteur

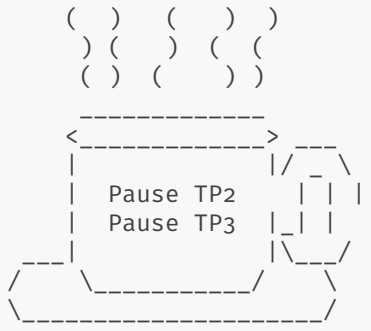
Historiquement

- géré grâce à une interruption accessible depuis le ring 3
- Interrupt Gate avec DPL=3 dans l'IDT
- les programmes utilisent `int N`
- où N correspond à l'indice dans l'IDT du descripteur

De nos jours

- on utilise les *fast system call*
- en passant par les instructions `sysenter, sysexit`
- l'OS configure des MSRs
  - `MSR_SYSENTER_CS`
  - `MSR_SYSENTER_EIP`
  - `MSR_SYSENTER_ESP`


Ring 3
CPU Internal
Ring 0
Processus
int read()
{
eax = 3
int 0x80
}
esp = TSS.esp0
push ss
push esp
push eflags
push cs
push eip
EIP =
IDT[0x80].offset
+
GDT[IDT[0x80].selector].base
syscall_excp_handler:
pusha
call syscall_table[eax]
popa
iret
syscall_table:
.long do_exit
.long do_open
.long do_write
.long do_read
…


Pause TP2
Pause TP3

La pagination

- l'adresse linéaire n'est plus l'adresse physique finale (RAM)
- elle devient une adresse virtuelle
- traduite en adresse physique via des tables par la MMU
- un peu de jargon
  - on parle d'espace d'adressage (*address space*)
  - de *mapping* mémoire
  - page tables/directory, memory management unit (MMU)
  - page table entries (pte)

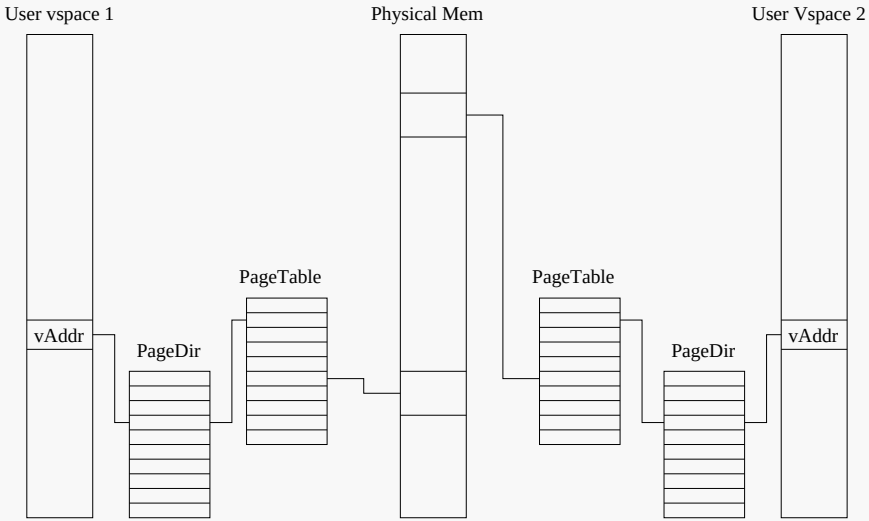
La pagination

- l'adresse linéaire n'est plus l'adresse physique finale (RAM)
- elle devient une adresse virtuelle
- traduite en adresse physique via des tables par la MMU
- un peu de jargon
  - on parle d'espace d'adressage (*address space*)
  - de *mapping* mémoire
  - page tables/directory, memory management unit (MMU)
  - page table entries (pte)

Intérêt

- optimiser la consommation mémoire
  - demand paging (ie. la pile)
  - shared memory (ie. des *threads*)
- charger plusieurs fois le meme programme au meme instant
  - adresses virtuelles identiques (ie. fichier exécutable)
  - adresses physiques distinctes

Espaces d'adressage de 2 exécutables



Répertoire et tables

- un répertoire de pages (*page directory*, PGD)
- une ou plusieurs table(s) de pages (*page tables*, PTB)
- le répertoire et les tables contiennent uniquement des adresses physiques !
- les entrées du répertoire et des tables sont assez similaires

Répertoire et tables

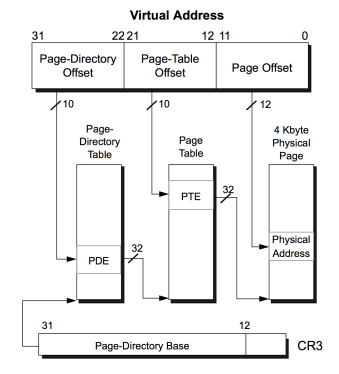
- un répertoire de pages (*page directory*, PGD)
- une ou plusieurs table(s) de pages (*page tables*, PTB)
- le répertoire et les tables contiennent uniquement des adresses physiques !
- les entrées du répertoire et des tables sont assez similaires

Dans le détail

- le PGD voit son adresse physique stockée dans `cr3`
- chaque table contient au plus 1024 entrées de 4 octets
- une entrée de table (*PTE*) définie une page de 4KB
- une table de pages couvre donc $1024 * 4KB = 4MB$ d'espace virtuel
- une entrée de répertoire (*PDE*) peut définir
  - une table de pages de 1024 entrées
  - une page de 4MB (*bit pse*)
- un répertoire de pages couvre donc $1024 * 4MB = 4GB$ d'espace virtuel

Traduction d'une adresse virtuelle en adresse physique

- adresse de 32 bits découpée en 3 parties
- 10 bits d'index dans le PGD (2^{10} = 1024)
- 10 bits d'index dans la PTB
- 12 bits d'*offset* dans la page (2^{12} = 4096)



Détail des entrées de PTE/PDE

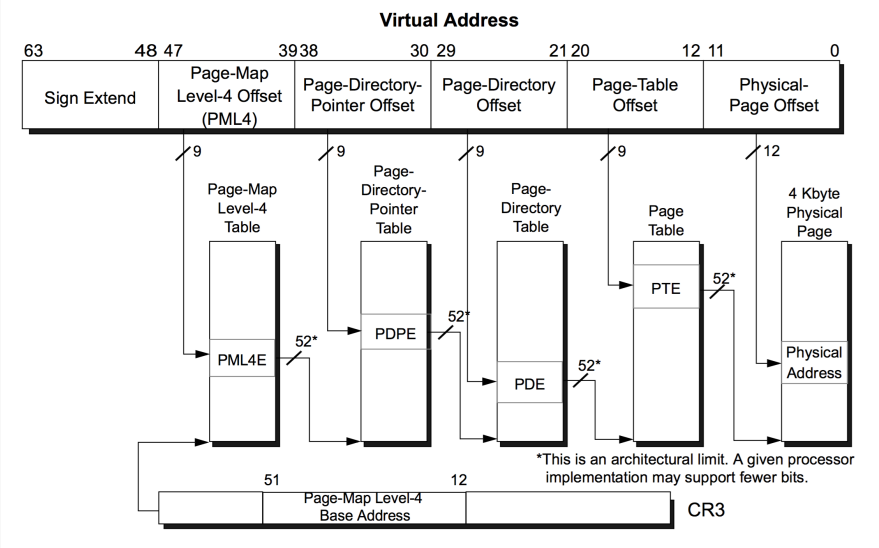
- Present, génère #PF (*page fault*) si 0
- User (ring 3), Supervisor (Ring 0,1,2)
- PSE (page de 4MB ou table de pages)
- G (global), mapping persistant
- bits de gestion des caches (PAT, PCD, PWT)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]																				Ignored							PCD	PWT	Ignored			CR3
Bits 31:22 of address of 4MB page frame									Reserved (must be 0)					Bits 39:32 of address[2]				PAT	Ignored			G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 4MB page
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored																															0	PDE: not present
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																															0	PTE: not present

De nombreuses variations

- taille des adresses virtuelles/physiques
- taille des pages adressables

Mode		Physical-Address Extensions (CR4.PAE)	Page-Size Extensions (CR4.PSE)	Page-Directory Pointer Offset	Page-Directory Page Size	Resulting Physical-Page Size	Maximum Virtual Address	Maximum Physical Address
Long Mode	64-Bit Mode	Enabled	–	PDPE.PS=0	PDE.PS=0	4 Kbyte	64-bit	52-bit
	Compatibility Mode				PDE.PS=1	2 Mbyte		
				PDPE.PS=1	–	1 Gbyte		
Legacy Mode		Enabled	–	PDPE.PS=0	PDE.PS=0	4 Kbyte	32-bit	52-bit
					PDE.PS=1	2 Mbyte		52-bit
		Disabled	Disabled		–	4 Kbyte		32-bit
			Enabled		PDE.PS=0	4 Kbyte		32-bit
					PDE.PS=1	4 Mbyte		40-bit


Virtual Address
63
48 47
39 38
30 29
21 20
12 11
0
Sign Extend
Page-Map Level-4 Offset (PML4)
Page-Directory-Pointer Offset
Page-Directory Offset
Page-Table Offset
Physical-Page Offset
9
9
9
9
12
Page-Map Level-4 Table
Page-Directory-Pointer Table
Page-Directory Table
Page Table
4 Kbyte Physical Page
PML4E
PDPE
PDE
PTE
Physical Address
52*
52*
52*
52*
*This is an architectural limit. A given processor implementation may support fewer bits.
51
12
Page-Map Level-4 Base Address
CR3

Les Translation Lookaside Buffers (TLBs)

- traduction couteuse
- chaque traduction est conservée
  - dans des caches spécifiques (les TLBs)
  - temporairement, une écriture dans *cr3* les invalide
  - on parle alors de *TLB flush*
- les entrées Global sont conservées (sauf si écriture dans *cr4*)

Les Translation Lookaside Buffers (TLBs)

- traduction couteuse
- chaque traduction est conservée
  - dans des caches spécifiques (les TLBs)
  - temporairement, une écriture dans *cr3* les invalide
  - on parle alors de *TLB flush*
- les entrées Global sont conservées (sauf si écriture dans *cr4*)

Dans la pratique, que fait un OS ?

- le noyau expose sa mémoire
  - dans tous les espaces d'adressage
  - en mode *supervisor*, bit U=0
  - en mode global, bit G=1
- chaque tache possède son propre espace d'adressage
- maintenu par le noyau uniquement (sécurité)
- chaque changement de tache implique un changement d'espace d'adressage
- et une perte des caches de traduction (*write cr3*)
- sauf des entrées globales

Le problème

- pagination activée ⇒ toute adresse est virtuelle
- or les tables contiennent des adresses physiques (ie. celles des autres tables)
- comment lire/modifier une table quand on ne connait que son adresse physique ?

Le problème

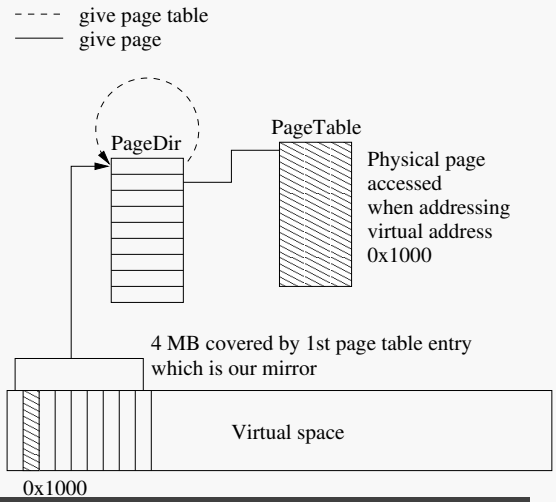
- pagination activée ⇒ toute adresse est virtuelle
- or les tables contiennent des adresses physiques (ie. celles des autres tables)
- comment lire/modifier une table quand on ne connait que son adresse physique ?

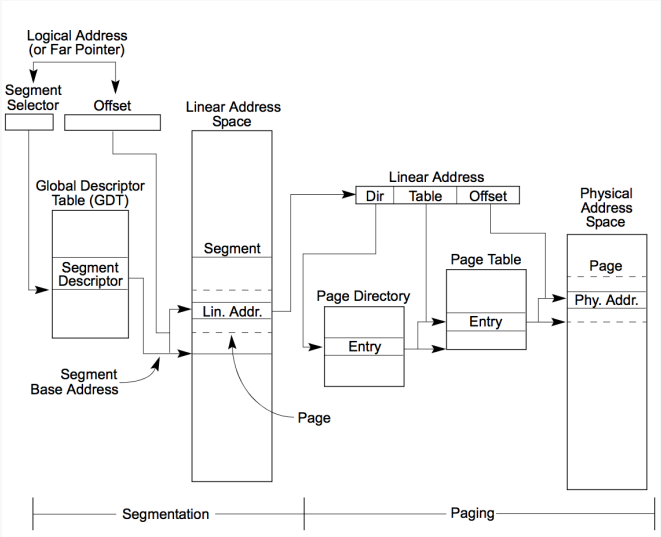
Plusieurs solutions

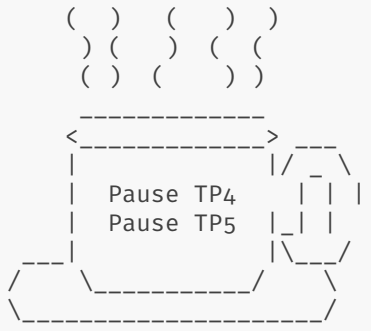
- *identity mapping* ... misérable
- correspondance linéaire virtuelle/physique (noyau linux, *PAGE_OFFSET*)
- un peu plus métaphysique ... le *self-mapping*

- une entrée du PGD est utilisée comme *self-map index*
- l'adresse de la table de pages à cet index est le PGD lui-meme
- conséquence : annulation d'un niveau d'indirection
- les pages finales accédées seront les tables de pages
- inconvénient : on perd 4MB d'espace virtuel

- une entrée du PGD est utilisée comme *self-map index*
- l'adresse de la table de pages à cet index est le PGD lui-meme
- conséquence : annulation d'un niveau d'indirection
- les pages finales accédées seront les tables de pages
- inconvénient : on perd 4MB d'espace virtuel




Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Segment
Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical
Address
Space
Page
Phy. Addr.
Segmentation
Paging


Pause TP4
Pause TP5

Le contexte

- considérons un processus (A) s'exécutant en ring 3 du mode protégé
- celui-ci est interrompu par l'apparition d'une interruption d'horloge (*irq 0*)
- le noyau (K) en ring 0 traite l'interruption
- et décide d'ordonnancer un autre processus (B)

Le contexte

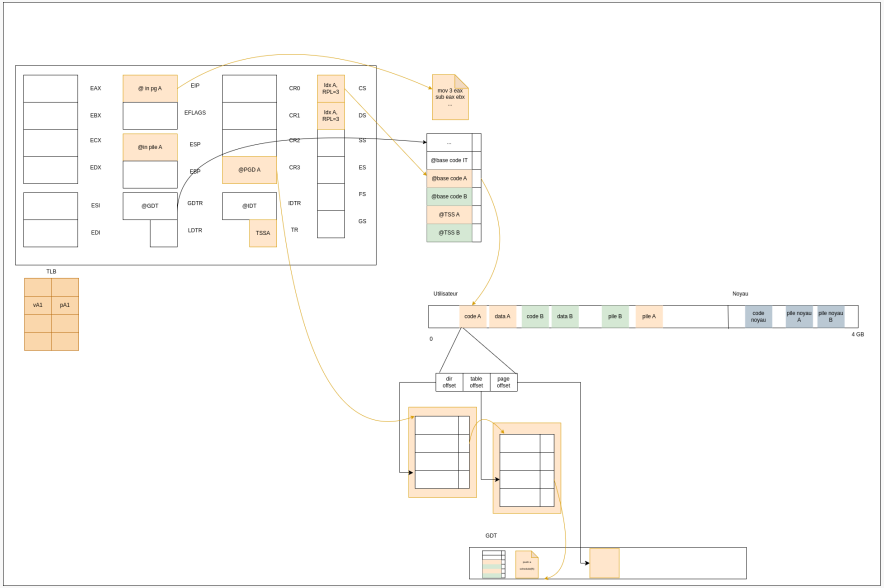
- considérons un processus (A) s'exécutant en ring 3 du mode protégé
- celui-ci est interrompu par l'apparition d'une interruption d'horloge (*irq 0*)
- le noyau (K) en ring 0 traite l'interruption
- et décide d'ordonnancer un autre processus (B)

Cette vue d'ensemble regroupe

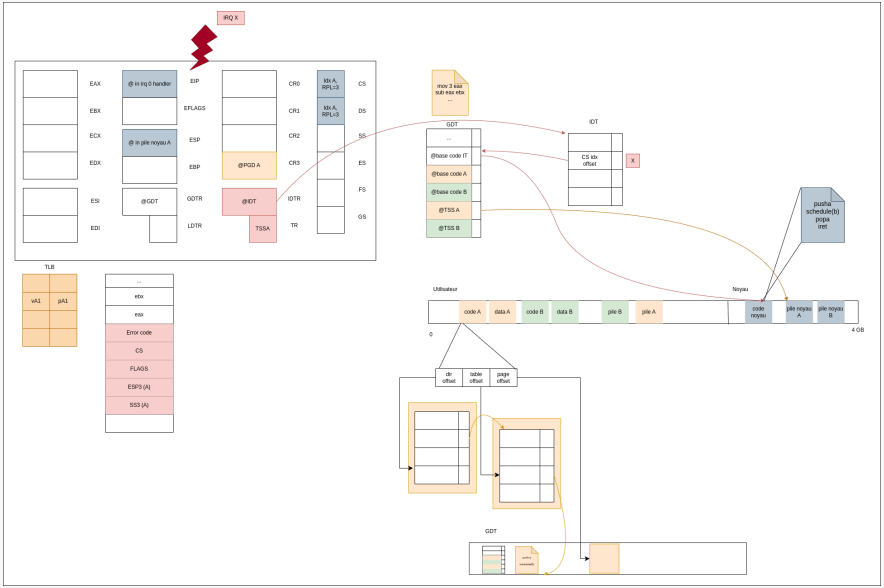
- la mémoire physique
- les espaces d'adressage des processus A et B
- les mappings virtuels/physiques correspondant
- ainsi que la mécanique du mode protégé, à savoir
  - traitement d'une interruption
  - consultation des tables IDT/GDT, du TSS
  - changement de niveau de privilèges
  - mise à jour du répertoire de pages et des TLBs
  - retour d'interruption/reprise d'un processus

Exemple d'ordonnancement : Tâche A en cours d'exécution

Exemple d'ordonnancement : Tâche A en cours d'exécution

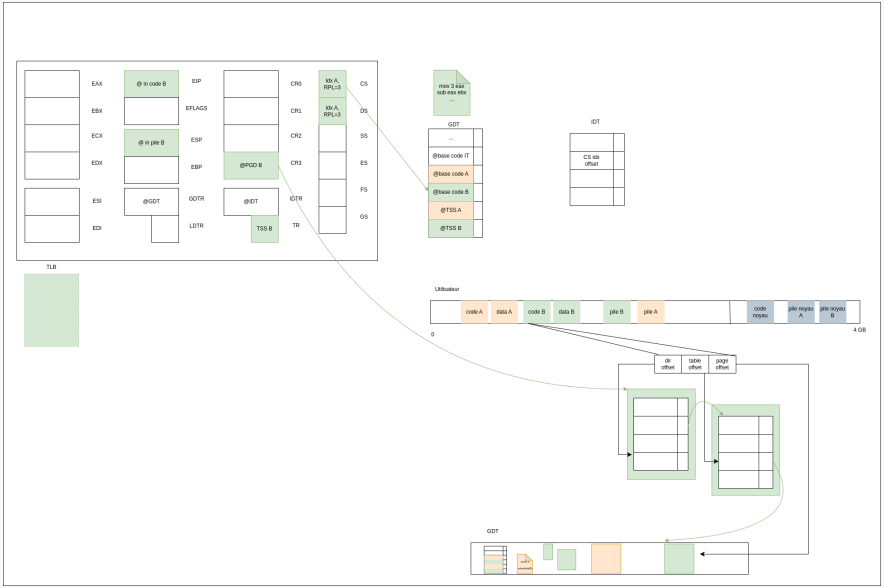


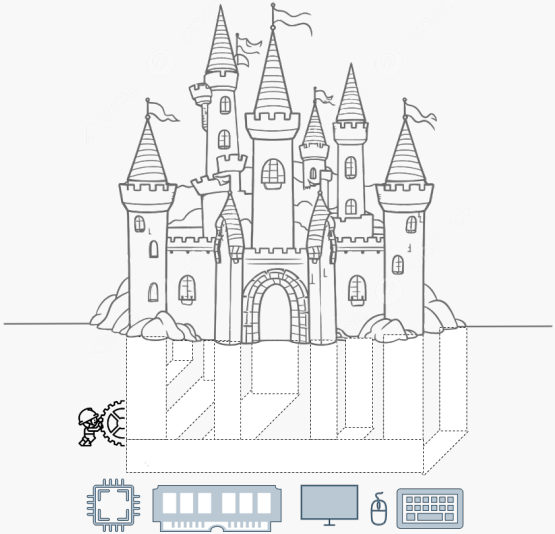
Exemple d'ordonnancement : Traitement arrivée interruption par CPU/Noyau

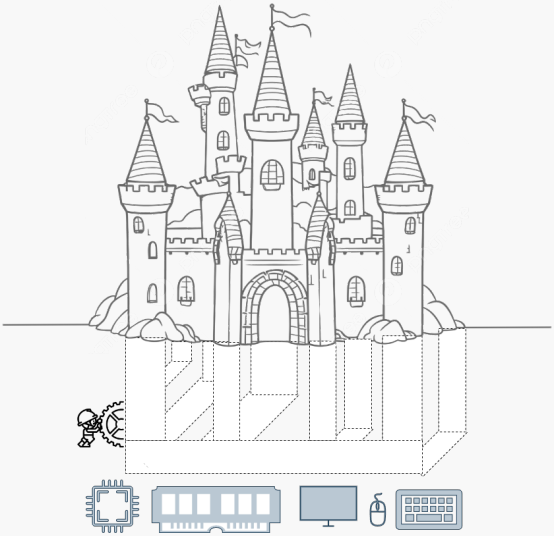

IRQ X
EAX
EBX
ECX
EDX
ESI
EDI
@ in irq 0 handler
@ in pile noyau A
@GDT
EIP
EFLAGS
ESP
EBP
GDTR
LDTR
CR0
CR1
CR2
CR3
@PGD A
@IDT
IDTR
TSSA
TR
klx A, RPL=3
klx A, RPL=3
CS
DS
SS
ES
FS
GS
mov 3 eax
sub eax ebx
...
GDT
@base code IT
@base code A
@base code B
@TSS A
@TSS B
IDT
CS idx offset
x
pusha
schedule(b)
popa
iret
TLB
vA1
pA1
ebx
eax
Error code
CS
FLAGS
ESP3 (A)
SS3 (A)
Utilisateur
Noyau
code A
data A
code B
data B
pile B
pile A
code noyau
pile noyau A
pile noyau B
4 GB
0
dir offset
table offset
page offset
GDT

Exemple d'ordonnancement : Tâche B prête à s'exécuter

Exemple d'ordonnancement : Tâche B prête à s'exécuter

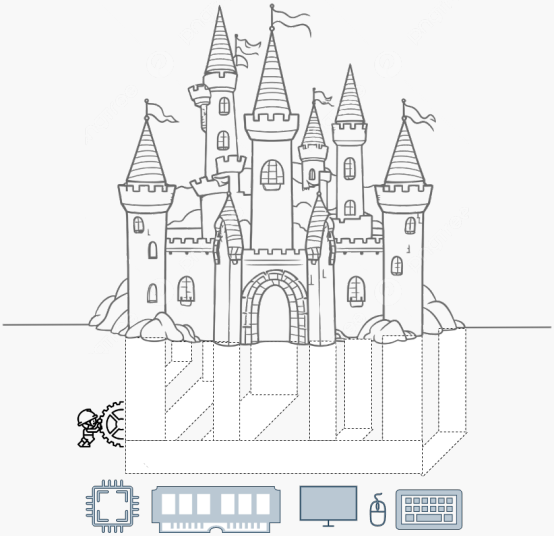






Grandes lignes :

- Isolation
  - Exemple : entre applications et noyau, entre les applications entre elles, etc.
- Limitation
- Gestion des erreurs

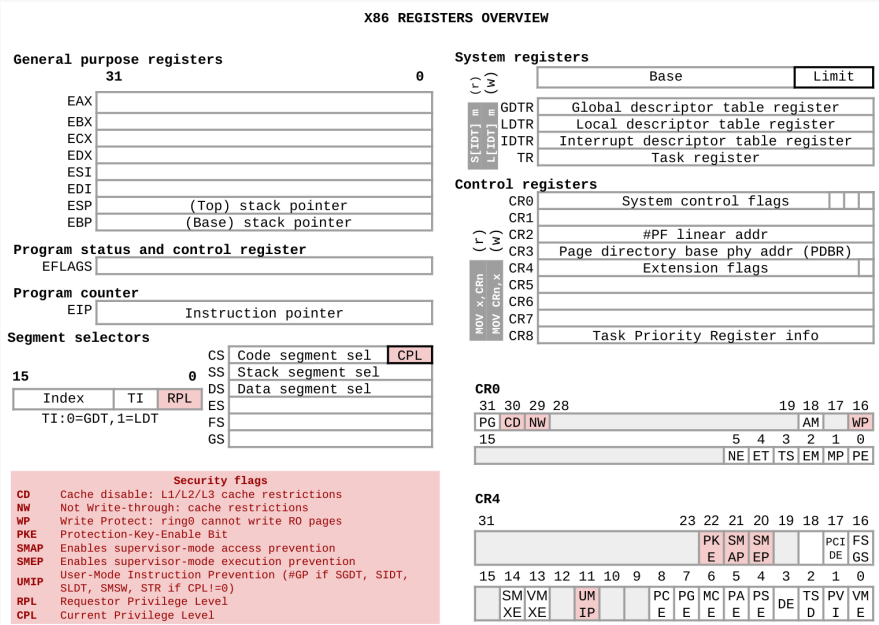


Grandes lignes :

- Isolation
  - Exemple : entre applications et noyau, entre les applications entre elles, etc.
- Limitation
- Gestion des erreurs

Concepts en vrac :

- Authentification
- DEP (data execution protection)
- ASLR
- Droits RWX sur les pages mémoires
- Vérification d'intégrité
- Sandboxing


X86 REGISTERS OVERVIEW
General purpose registers
31
0
EAX
EBX
ECX
EDX
ESI
EDI
ESP
(Top) stack pointer
EBP
(Base) stack pointer
Program status and control register
EFLAGS
Program counter
EIP
Instruction pointer
Segment selectors
15
0
Index
TI
RPL
TI:0=GDT,1=LDT
CS
Code segment sel
CPL
SS
Stack segment sel
DS
Data segment sel
ES
FS
GS
System registers
(r) (w)
Base
Limit
S[IDT] m
L[IDT] m
GDTR
Global descriptor table register
LDTR
Local descriptor table register
IDTR
Interrupt descriptor table register
TR
Task register
Control registers
(r) (w)
MOV x,CRn
MOV CRn,x
CR0
System control flags
CR1
CR2
#PF linear addr
CR3
Page directory base phy addr (PDBR)
CR4
Extension flags
CR5
CR6
CR7
CR8
Task Priority Register info
CR0
31 30 29 28
19 18 17 16
PG CD NW
AM
WP
15
5 4 3 2 1 0
NE ET TS EM MP PE
CR4
31
23 22 21 20 19 18 17 16
PKE SMAP SMEP
PCIDE
FSGS
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SMXE VMXE UMIP PCE PGE MCE PAE PSE DE TSD PVI VME
Security flags
CD Cache disable: L1/L2/L3 cache restrictions
NW Not Write-through: cache restrictions
WP Write Protect: ring0 cannot write RO pages
PKE Protection-Key-Enable Bit
SMAP Enables supervisor-mode access prevention
SMEP Enables supervisor-mode execution prevention
UMIP User-Mode Instruction Prevention (#GP if SGDT, SIDT, SLDT, SMSW, STR if CPL!=0)
RPL Requestor Privilege Level
CPL Current Privilege Level

1. **Vérifier le mode d'exécution du CPU**

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : `CR0.PE`

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : `CR0.PE`
2. **Si protégé, vérifier si mode paginé activé aussi**

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : `CR0.PE`
2. **Si protégé, vérifier si mode paginé activé aussi**
   - Registre concerné : `CR0.PG`

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : `CR0.PE`
2. **Si protégé, vérifier si mode paginé activé aussi**
   - Registre concerné : `CR0.PG`
3. **Comprendre comment est configurée la segmentation :**

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : CR0.PE
2. **Si protégé, vérifier si mode paginé activé aussi**
   - Registre concerné : CR0.PG
3. **Comprendre comment est configurée la segmentation :**
   - 3.1 Où est stockée la GDT ?
     - Registre concerné : GDTR
   - 3.2 Quels types de descripteurs contient la GDT ? Et avec quels droits ?
     - Objet concerné : le champ "type" des descripteurs
   - 3.3 Définit-elle des niveaux de privilèges différents ?
     - Objet concerné : le champ "DPL" des descripteurs
   - 3.4 Définit-elle un modèle FLAT ?
     - Objet concerné : le champ "base" et "limit" des descripteurs

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : CR0.PE
2. **Si protégé, vérifier si mode paginé activé aussi**
   - Registre concerné : CR0.PG
3. **Comprendre comment est configurée la segmentation :**
   - 3.1 Où est stockée la GDT ?
     - Registre concerné : GDTR
   - 3.2 Quels types de descripteurs contient la GDT ? Et avec quels droits ?
     - Objet concerné : le champ "type" des descripteurs
   - 3.3 Définit-elle des niveaux de privilèges différents ?
     - Objet concerné : le champ "DPL" des descripteurs
   - 3.4 Définit-elle un modèle FLAT ?
     - Objet concerné : le champ "base" et "limit" des descripteurs
4. **Comprendre comment est utilisée la segmentation au fil de l'exécution du système**

1. **Vérifier le mode d'exécution du CPU**
   - Registre concerné : CR0.PE
2. **Si protégé, vérifier si mode paginé activé aussi**
   - Registre concerné : CR0.PG
3. **Comprendre comment est configurée la segmentation :**
   - 3.1 Où est stockée la GDT ?
     - Registre concerné : GDTR
   - 3.2 Quels types de descripteurs contient la GDT ? Et avec quels droits ?
     - Objet concerné : le champ "type" des descripteurs
   - 3.3 Définit-elle des niveaux de privilèges différents ?
     - Objet concerné : le champ "DPL" des descripteurs
   - 3.4 Définit-elle un modèle FLAT ?
     - Objet concerné : le champ "base" et "limit" des descripteurs
4. **Comprendre comment est utilisée la segmentation au fil de l'exécution du système**
   - Registres concernés : les sélecteurs de segment (CS, DS, SS, GS, FS, ES)

5. **Si activée, comprendre comment est configurée la pagination**

5. **Si activée, comprendre comment est configurée la pagination**
   - 5.1 Où sont stockés les PGD ? les PTB ?
     - Registre concerné : CR3
   - 5.2 Y a-t-il utilisation de pages de 4Mb ? 4Kb ?
     - Objet concerné : champ ”g” des entrées de PGD
   - 5.3 Quelle correspondance entre adressage virtuel et adressage physique ?
     - Objet concerné : champ ”address” des entrées de PTB
   - 5.4 Quels droits sur quelles plages d'adresse ? Quelles restrictions d'accès aux objets (physiques) noyau à une application ?
     - Objets concernés : champs ”U/S” et ”R/W” des entrées de PGD

5. **Si activée, comprendre comment est configurée la pagination**
   - 5.1 Où sont stockés les PGD ? les PTB ?
     - Registre concerné : CR3
   - 5.2 Y a-t-il utilisation de pages de 4Mb ? 4Kb ?
     - Objet concerné : champ "g" des entrées de PGD
   - 5.3 Quelle correspondance entre adressage virtuel et adressage physique ?
     - Objet concerné : champ "address" des entrées de PTB
   - 5.4 Quels droits sur quelles plages d'adresse ? Quelles restrictions d'accès aux objets (physiques) noyau à une application ?
     - Objets concernés : champs "U/S" et "R/W" des entrées de PGD
6. **Comprendre comment est utilisée la pagination**

5. **Si activée, comprendre comment est configurée la pagination**
   - 5.1 Où sont stockés les PGD ? les PTB ?
     - Registre concerné : CR3
   - 5.2 Y a-t-il utilisation de pages de 4Mb ? 4Kb ?
     - Objet concerné : champ ”g” des entrées de PGD
   - 5.3 Quelle correspondance entre adressage virtuel et adressage physique ?
     - Objet concerné : champ ”address” des entrées de PTB
   - 5.4 Quels droits sur quelles plages d'adresse ? Quelles restrictions d'accès aux objets (physiques) noyau à une application ?
     - Objets concernés : champs ”U/S” et ”R/W” des entrées de PGD
6. **Comprendre comment est utilisée la pagination**
   - 6.1 Quelle configuration noyau initiale ?
   - 6.2 Y a-t-il chargement d'un nouveau set de tables au basculement d'exécution entre applications ?
   - 6.3 Les entrées de tables sont-elles mises à jour ? si oui quand et avec quels droits ?
   - 6.4 Qu'en est-il de la mise à jour des TLB ?
     - Instructions dédiées aux caches (flush, invlg, etc.)

5. **Si activée, comprendre comment est configurée la pagination**
   - 5.1 Où sont stockés les PGD ? les PTB ?
     - Registre concerné : CR3
   - 5.2 Y a-t-il utilisation de pages de 4Mb ? 4Kb ?
     - Objet concerné : champ "g" des entrées de PGD
   - 5.3 Quelle correspondance entre adressage virtuel et adressage physique ?
     - Objet concerné : champ "address" des entrées de PTB
   - 5.4 Quels droits sur quelles plages d'adresse ? Quelles restrictions d'accès aux objets (physiques) noyau à une application ?
     - Objets concernés : champs "U/S" et "R/W" des entrées de PGD
6. **Comprendre comment est utilisée la pagination**
   - 6.1 Quelle configuration noyau initiale ?
   - 6.2 Y a-t-il chargement d'un nouveau set de tables au basculement d'exécution entre applications ?
   - 6.3 Les entrées de tables sont-elles mises à jour ? si oui quand et avec quels droits ?
   - 6.4 Qu'en est-il de la mise à jour des TLB ?
     - Instructions dédiées aux caches (flush, invlg, etc.)

Note : A cette étape, l'auditeur a une bonne vision de la cartographie mémoire imposée par le noyau au système entier.

7. **Comprendre comment est configurée la gestion des interruptions**

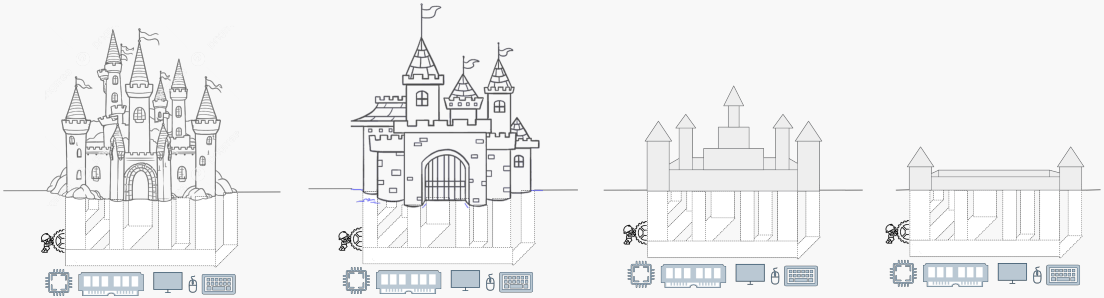
7. **Comprendre comment est configurée la gestion des interruptions**
   - 7.1 Où est stockée la table des descripteurs d'interruption ?
     - Registre concerné : IDTR
   - 7.2 Une application peut-elle modifiée ce qui est stocké à cette adresse ?
   - 7.3 Tous les descripteurs pointent-ils vers un handler commun ?
   - 7.4 Quels droits sont-ils requis pour exécuter les handlers ?
     - Objet concerné : DPL des descripteurs
   - 7.5 Comment sont implémentés les handlers ?
     - Bonne sauvegarde des registres ?
     - Bonne restauration des registres avant IRET ?

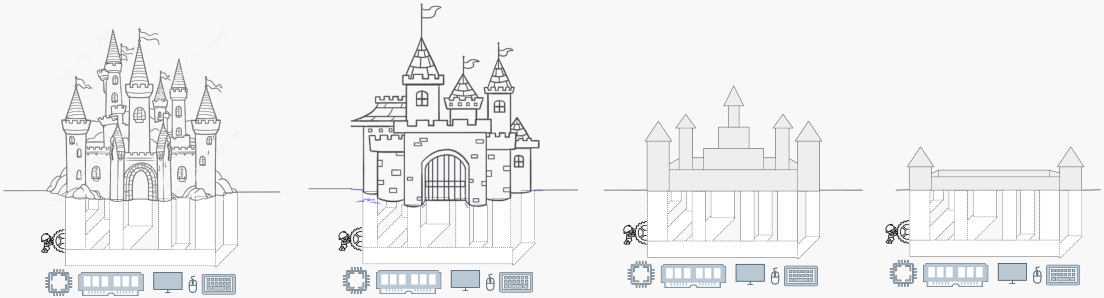
7. **Comprendre comment est configurée la gestion des interruptions**
   - 7.1 Où est stockée la table des descripteurs d'interruption ?
     - Registre concerné : IDTR
   - 7.2 Une application peut-elle modifiée ce qui est stocké à cette adresse ?
   - 7.3 Tous les descripteurs pointent-ils vers un handler commun ?
   - 7.4 Quels droits sont-ils requis pour exécuter les handlers ?
     - Objet concerné : DPL des descripteurs
   - 7.5 Comment sont implémentés les handlers ?
     - Bonne sauvegarde des registres ?
     - Bonne restauration des registres avant IRET ?
8. **Comprendre comment sont implémentés les appels systèmes**
   - 8.1 Via l'IDT ? Via la GDT ?
     - Objet concerné : Descripteurs avec DPL = 3
   - 8.2 Combien y en a-t-il ?
   - 8.3 Pour chaque appel système, comment sont vérifiés les paramètres provenant des applications ?

7. **Comprendre comment est configurée la gestion des interruptions**
   - 7.1 Où est stockée la table des descripteurs d'interruption ?
     - Registre concerné : IDTR
   - 7.2 Une application peut-elle modifiée ce qui est stocké à cette adresse ?
   - 7.3 Tous les descripteurs pointent-ils vers un handler commun ?
   - 7.4 Quels droits sont-ils requis pour exécuter les handlers ?
     - Objet concerné : DPL des descripteurs
   - 7.5 Comment sont implémentés les handlers ?
     - Bonne sauvegarde des registres ?
     - Bonne restauration des registres avant IRET ?
8. **Comprendre comment sont implémentés les appels systèmes**
   - 8.1 Via l'IDT ? Via la GDT ?
     - Objet concerné : Descripteurs avec DPL = 3
   - 8.2 Combien y en a-t-il ?
   - 8.3 Pour chaque appel système, comment sont vérifiés les paramètres provenant des applications ?

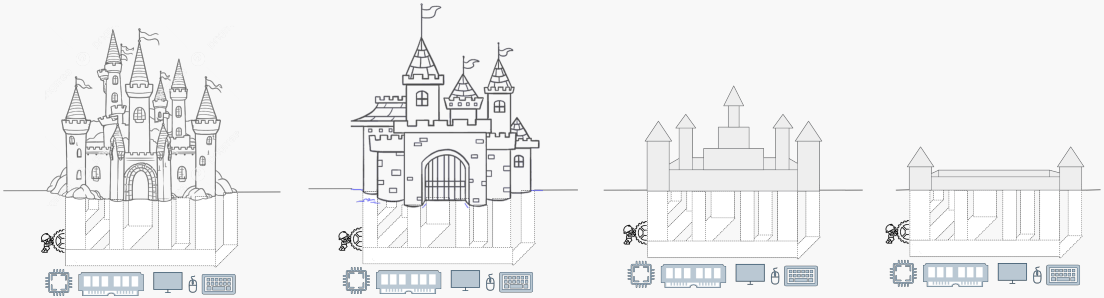
Note : appels systèmes == vecteur d'attaque courant pour élever ses privilèges

Si une vulnérabilité est identifiée dans l'implémentation d'un appel système, la réussite de son exploitation dépendra de la configuration mémore exposée par le noyau.





Tout noyau doit savoir profiter au maximum des mesures de sécurité proposées par le matériel !



Tout noyau doit savoir profiter au maximum des mesures de sécurité proposées par le matériel !

Tout détail mal utilisé/implémenté peut être fatal pour l'ensemble de la sécurité du système !

- Au sein d'un système d'exploitation,
  1. Le noyau possède des privilèges plus élevés que les applications.
  2. Le noyau possède des privilèges moins élevés que les applications.

- Au sein d'un système d'exploitation,
  1. Le noyau possède des privilèges plus élevés que les applications.
  2. Le noyau possède des privilèges moins élevés que les applications.
- Concevoir une architecture de micronoyau
  1. Augmente la surface d'attaque du noyau par rapport à un noyau monolithique.
  2. Ralentit les performances en termes de rapidité d'exécution.

- Le mode par défaut de démarrage du CPU est le mode :
  1. SMM
  2. Protégé
  3. v8086
  4. Réel
  5. VMX
  6. ECX

- Le mode par défaut de démarrage du CPU est le mode :
  1. SMM
  2. Protégé
  3. v8086
  4. Réel
  5. VMX
  6. ECX
- Quel registre contient le niveau de privilège courant ?
  1. EIP
  2. CR0
  3. EFLAGS
  4. CS
  5. GDTR
  6. ESP
  7. TR

- La segmentation et la pagination sont des fonctionalités
  1. Proposées par le matériel et uniquement gérées par le matériel.
  2. Purement logicielles.
  3. Supportées par le matériel et configurables par le noyau.
  4. Qui existent en mode réel comme en mode protégé.

- La segmentation et la pagination sont des fonctionalités
  1. Proposées par le matériel et uniquement gérées par le matériel.
  2. Purement logicielles.
  3. Supportées par le matériel et configurables par le noyau.
  4. Qui existent en mode réel comme en mode protégé.
- En mode protégé,
  1. La segmentation est activée par défaut.
  2. La pagination est activée par défaut.

- Aujourd'hui, la segmentation est principalement utilisée pour :
  1. Mettre en place des niveaux de privilèges.
  2. Donner à chaque application un espace d'adressage de 4GB.
  3. Apporter de la ségrégation d'accès mémoire entre applications.

- Aujourd'hui, la segmentation est principalement utilisée pour :
  1. Mettre en place des niveaux de privilèges.
  2. Donner à chaque application un espace d'adressage de 4GB.
  3. Apporter de la ségrégation d'accès mémoire entre applications.
- Dans le cas d'un modèle non *flat*, avec un segment de données de taille 64, que se passe-t-il si un programme tente d'écrire à l'adresse de base du segment + 65 ?
  1. Si le programme est en ring 0, il va pouvoir écrire à l'adresse demandée.
  2. Que le programme soit en ring 0 ou en ring 3, le CPU va lever une exception de type #GP.
  3. Si le programme est en ring 3, le CPU va lever une exception de type #PF.

- Les exceptions
  1. Sont une sous-partie des interruptions levées en cas de détection d'erreur par le CPU.
  2. Sont une sous-partie des interruptions levées en cas de détection d'erreur par le noyau.
  3. Ont pour numéro d'interruption un nombre supérieur à 32.

- Les exceptions
  1. Sont une sous-partie des interruptions levées en cas de détection d'erreur par le CPU.
  2. Sont une sous-partie des interruptions levées en cas de détection d'erreur par le noyau.
  3. Ont pour numéro d'interruption un nombre supérieur à 32.
- Lorsque l'on accède au registre CR0 en ring 3,
  1. Il ne se passe rien de particulier.
  2. Le noyau lève une erreur.
  3. Le CPU lève une erreur.

- Quel(s) registre(s) le CPU consulte-t-il pour traiter l'arrivée d'une interruption matérielle ?
  1. CR0
  2. CR3
  3. EAX
  4. EFLAGS
  5. GDTR
  6. IDTR
  7. TR

- Quel(s) registre(s) le CPU consulte-t-il pour traiter l'arrivée d'une interruption matérielle ?
  1. CR0
  2. CR3
  3. EAX
  4. EFLAGS
  5. GDTR
  6. IDTR
  7. TR
- Lors de l'arrivée d'une interruption matérielle, le CPU sauvegarde lui-même les registres suivants :
  1. CR0
  2. CR3
  3. CS
  4. SS
  5. EFLAGS
  6. EIP
  7. ESP
  8. IDTR
  9. TR
  10. Les registres généraux (EAX, etc.)

- Sous Linux, de combien de PGD(s) dispose un processus ?
  1. Au moins 2, pour ségréger espace noyau et espace utilisateur.
  2. Un par fonctions.
  3. Un seul.

- Sous Linux, de combien de PGD(s) dispose un processus ?
  1. Au moins 2, pour ségréger espace noyau et espace utilisateur.
  2. Un par fonctions.
  3. Un seul.
- Combien d'entrées de GDT sont nécessaires au minimum pour qu'un noyau puisse gérer des tâches utilisateur ?
  1. 2
  2. 4
  3. 5
  4. 6

- Les TLB
  1. Servent à accélérer la traduction d'adresse.
  2. Sont conservés au rechargement du registre CR3.
  3. Stockent la traduction d'adresse virtuelle en adresse physique.
  4. Stockent la traduction d'adresse physique en adresse virtuelle.

- Les TLB
  1. Servent à accélérer la traduction d'adresse.
  2. Sont conservés au rechargement du registre CR3.
  3. Stockent la traduction d'adresse virtuelle en adresse physique.
  4. Stockent la traduction d'adresse physique en adresse virtuelle.
- Pour mettre en place de la mémoire partagée entre deux tâches, il faut
  1. Utiliser le même PGD.
  2. Utiliser les mêmes adresses dans le champ addr des PTE.
  3. Utiliser les mêmes index de PTE dans les tables de page.

Architecture matérielle

- Les manuels Intel
  `https://www-ssl.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`
- Les manuels AMD et les BIOS & Kernel Developer's Guide
  `http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/`
- What every programmer should know about memory, Ulrich Drepper
  `http://www.akkadia.org/drepper/cpumemory.pdf`
- Protected Mode Software Architecture, Tom Shanley

Concepts

- Les systèmes d'exploitation - conception et mise en oeuvre, Andrew Tanenbaum
- Operating System Concepts, 8th Edition, Avi Silberschatz, Peter Baer Galvin, Greg Gagne
  `http://codex.cs.yale.edu/avi/os-book/OS8/os8c/slide-dir/`

Implémentation

- Understanding the Linux Kernel, 3rd Edition, Bovet & Cesati
- Windows Internals, 6th Edition, Russinovich & Ionescu
- The Design and Implementation of the 4.4 BSD Operating System, McKusick
- UNIX Internals : The New Frontiers, Vahalia Uresh
- Lion's commentary on Unix 6th Edition
- MIT xv6 OS
  `http://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf`
- Créez votre OS (Linux magazine)
  `http://sos.enix.org/fr/SOSDownload`

Votre avis compte

- remarques ?
- améliorations ?
- manquements ?
- suggestions ?
- questions ?