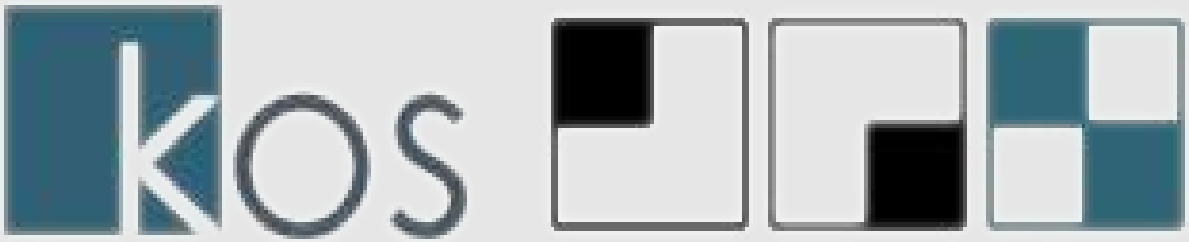
Fonctionnement des systèmes d'exploitation

-

KOS - Kid Operating System



Thomas Petazzoni

25 avril 2004

Fonctionnement d'un OS

- Concepts fondamentaux
  - Ressources d'exécution
  - Fichiers
- Dans le noyau Linux
  - Gestion mémoire
  - Gestion des processus
  - Fichiers
  - Pilotes de périphériques

Kid Operating System

- Présentation et objectifs
- Particularités
- Avenir

Ressources d'exécution : Threads

Pour exécuter un programme :
- instructions
- données

Application = flots d'instructions

Processeur = un flot d'instruction

Multitâche = illusion d'exécution en parallèle

Thread = flot d'instruction

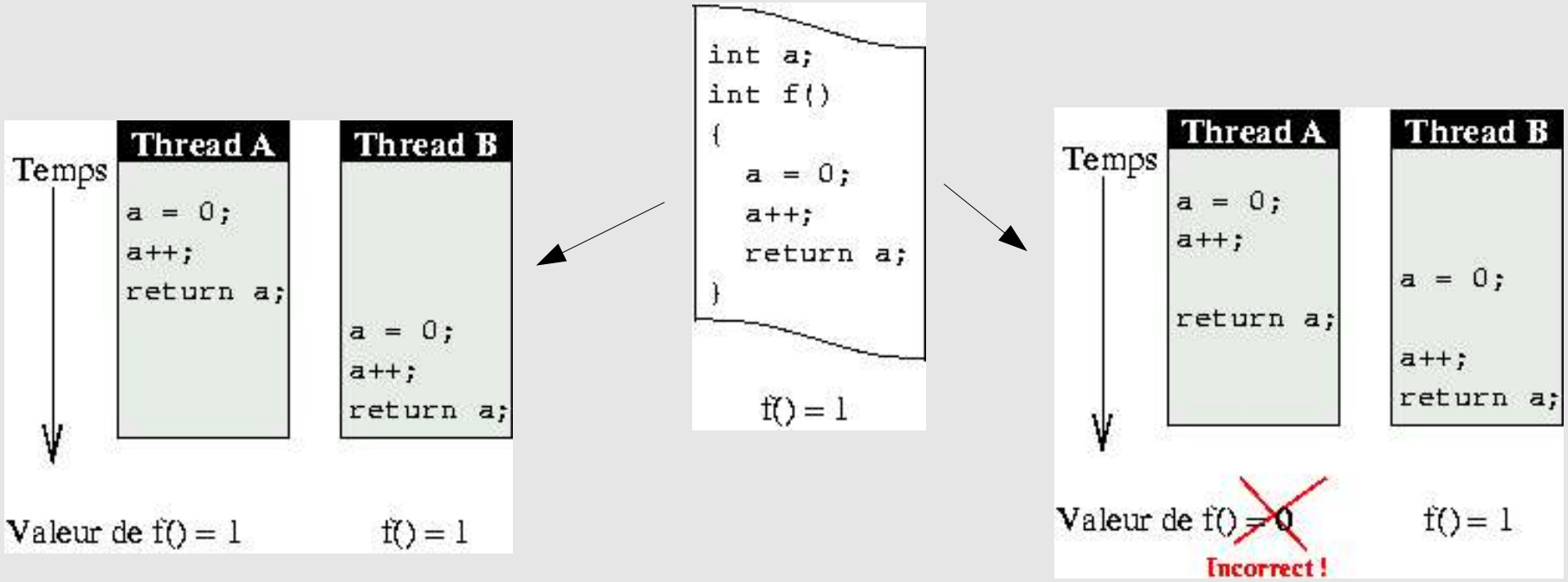
Pour passer d'un flot à un autre : **changement de contexte**

Gestion des threads : **ordonnanceur**

État du thread : *prêt* ou *bloqué*, dépend des ressources **demandées** et **disponibles**

Sources de blocage :
- matérielles
- logiques
- virtuelles

Mécanisme de synchronisation logicielle pour protéger les ressources matérielles et logiques.



Correct

Incorrect

Solution : synchroniser

- mutex
- sémaphores
- conditions
- files de messages

Gestion de la mémoire virtuelle 1/2

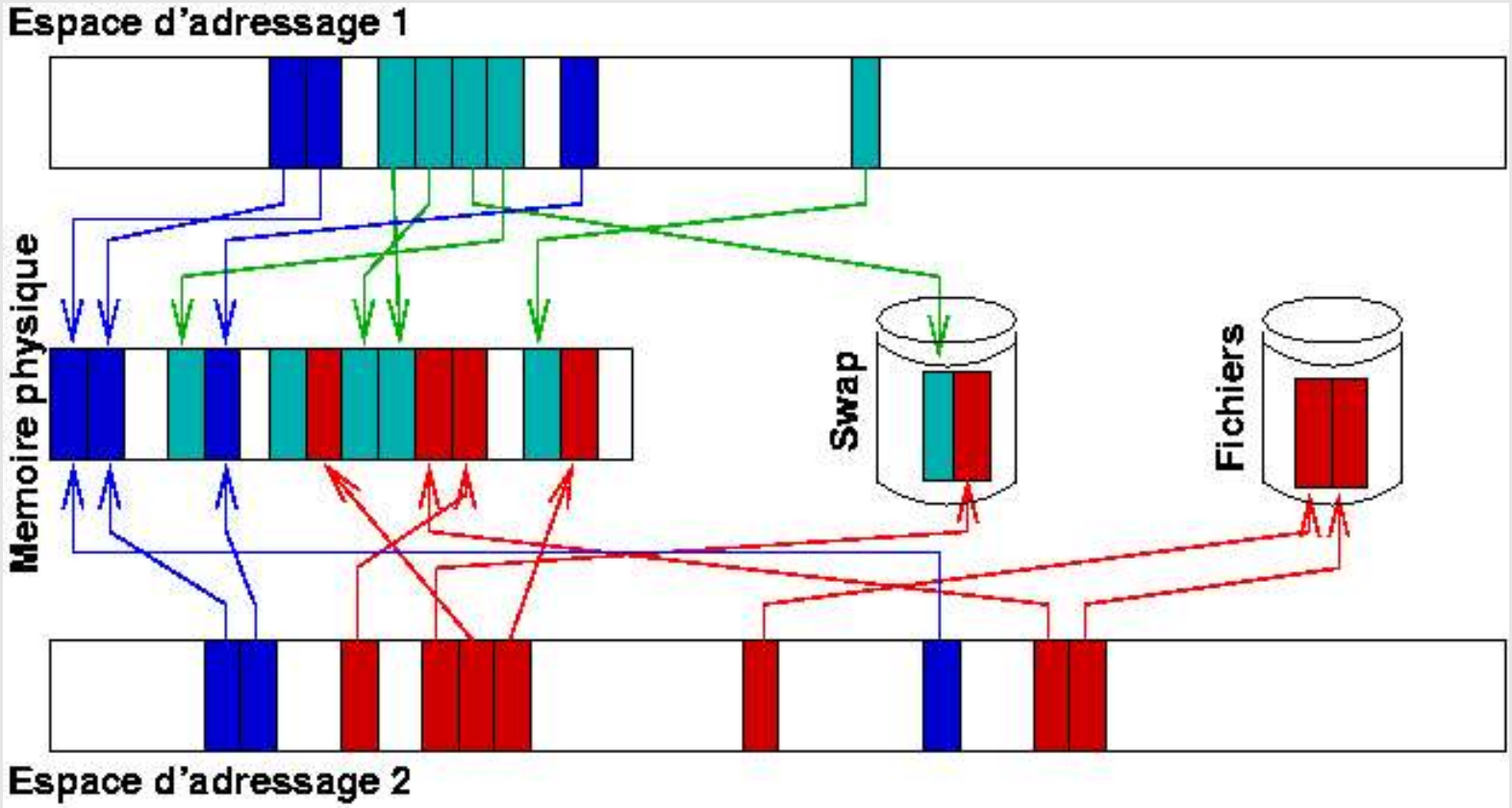
En mémoire :
- Instructions
- Données

Pour faire du multitâche robuste : deux « types » d'adresses
- Adresses **physiques**
- Adresses « **effectives** », relatives à l'**espace d'adressage** courant.

Conversion **effective -> physique :**
- En fonction de la configuration de l'espace d'adressage
- Réalisée par la MMU

Avantages :
- Adresses fixes
- Cloisonnement
- Partage possible de mémoire
- Taille mémoire effective >> taille mémoire physique

Gestion de la mémoire virtuelle 2/2



Application = configuration de l'espace d'adressage + ensemble de threads

Sous Unix : processus

Fichier = représentation des ressources du système accessibles à l'utilisateur.

Sous Unix : accès aux fichiers par une interface unique :

- read
- write
- seek
- stat
- ...

Interfaces étendues :

- sockets : accept, bind, ...
- répertoires : readdir, ...
- périphériques : ioctl

Linux : mémoire physique

Mémoire physique : mémoire matériellement présente dans l'ordinateur, RAM.

Sous Linux, chaque page physique est représentée par une structure *include/linux/mm.h:struct page*

Allocateur de pages physiques : *mm/page_alloc.c*

Appel au swapper *kswapd* si mémoire physique insuffisante.

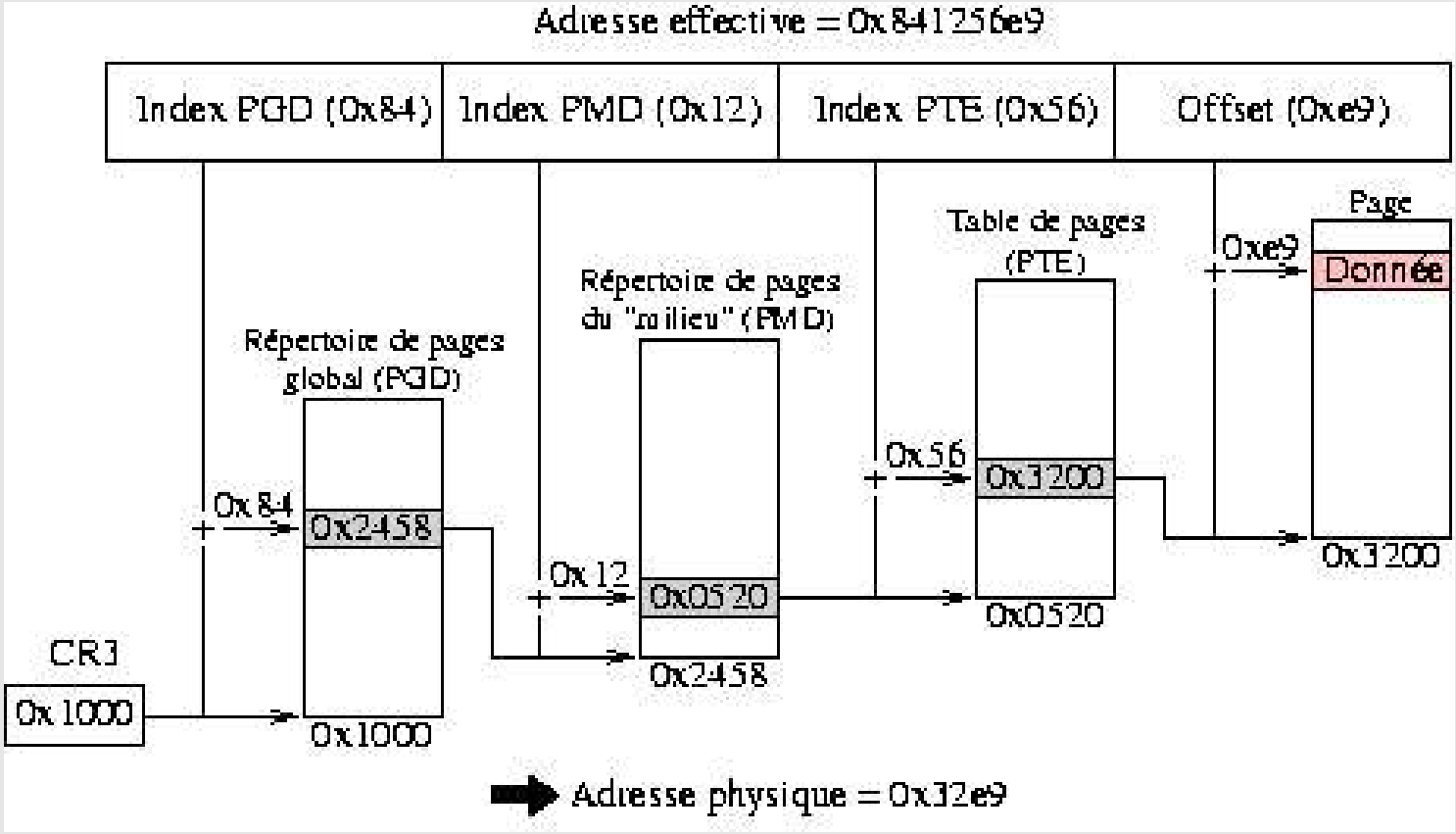
Système *slab allocator* pour allouer des objets de petites tailles.

Découpage en 3 zones :

- DMA (< 16 Mo)
- Normale (< 1 Go)
- HighMem (> 1 Go)

Adresses sur 32 bits : 4 Go par espace d'adressage.

Sous Linux, pagination à 3 niveaux.
Sur x86, pagination à 2 niveaux.



Linux : découpage de l'espace d'adressage

Espace d'adressage découpé en 2 parties :

- 0 -> PAGE_OFFSET (3 Go) : code et données du programme, bibliothèques partagées, tas et pile. **Partie différente d'un processus à l'autre.**
- PAGE_OFFSET -> fin : code et données du noyau. **Partie identique dans tous les espaces d'adressage.**

Partie consacrée au noyau identique à la mémoire physique : *identity mapping*.

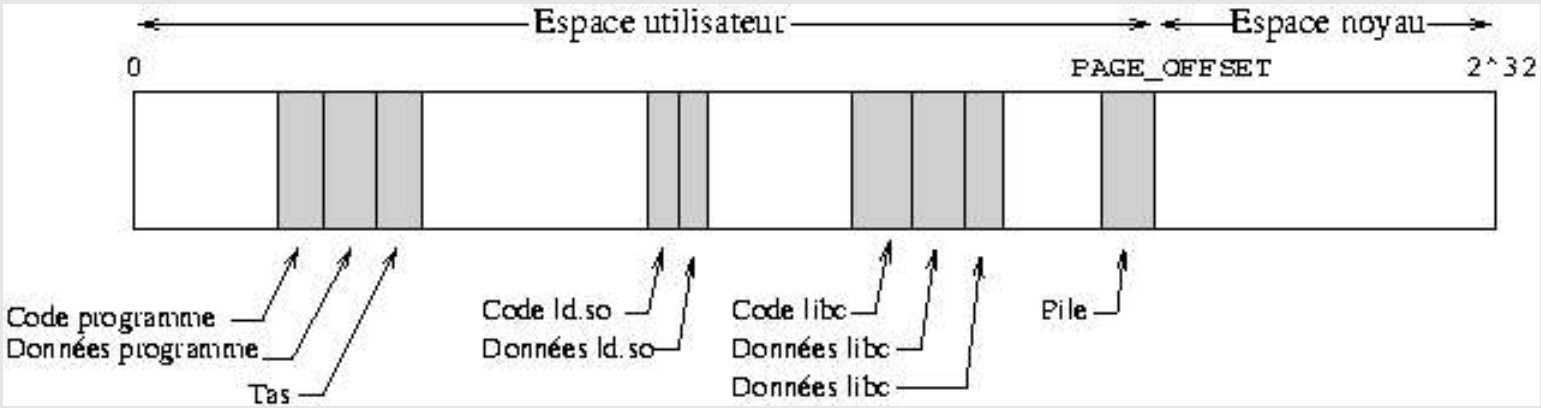
PAGE_OFFSET + x en virtuel => x en physique.

Mécanisme Highmem si RAM > 1 Go.

Linux : régions virtuelles

Espace d'adressage : *include/linux/sched.h:mm_struct*
Découpé en régions virtuelles.

```
thomas@crazy:~$ cat /proc/self/maps
08048000-0804c000 r-xp 00000000 03:02 12136      /bin/cat
0804c000-0804d000 rw-p 00003000 03:02 12136      /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:02 26269      /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:02 26269      /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
4001f000-4014d000 r-xp 00000000 03:02 24198      /lib/libc-2.3.2.so
4014d000-40156000 rw-p 0012d000 03:02 24198      /lib/libc-2.3.2.so
40156000-40159000 rw-p 00000000 00:00 0
bffff000-c0000000 rwxp 00000000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0
```



Linux : régions virtuelles

Une région virtuelle peut être sauvegardée sur disque (*backing store)* :

- Région virtuelle correspondant à un fichier (*file mapping)*
- Sinon région **anonyme**, sauvegarde dans le *swap*

Différents comportements sur modification :

- MAP_SHARED : modification partagées par tous les processus qui ont le même mapping.
- MAP_PRIVATE : modification provoque la copie de la page d'origine. Mécanisme de COW, *Copy On Write*.

Sous Linux, une région virtuelle est représentée par une structure *include/linux/mm.h:vm_area_struct.*

Gestion des régions :

- au **fork** / **exec**
- ensuite, **mmap** / **munmap** / **mremap** / **brk**

Linux : défaut de page

Défaut de page :

- Pas de page physique pour l'adresse virtuelle accédée
- Accès non autorisé

=> exécution d'une routine de résolution dans le noyau

Traitement :

- Page hors région => SIGSEGV
- Page en lecture seule accédée en écriture
  - Si région en lecture/écriture => COW
  - Sinon => SIGSEGV
- Sinon
  - *demand paging*

Deux types de *threads :*

- Threads **noyau** : mode privilégié, dans n'importe quel espace d'adressage. Ex: keventd, kswapd, ksoftirqd_CPU0.

- Threads **utilisateur** : mode utilisateur, dans un espace d'adressage donné. Représenté par *include/linux/sched.h:task_struct.*

Linux : threads

Création par **fork()** ou **clone()** *kernel/fork.c:do_fork()*

Fork : nouveau thread dans un nouvel espace d'adressage par duplication.

Clone : nouveau thread partageant certains éléments avec le contexte d'exécution appelant.

La bibliothèque Linux Threads (pthread_*) utilise **clone()**.

Suite à **clone()** ou **fork()**, les espaces d'adressage sont identiques : utilisation de **exec()** pour réinitialiser l'espace d'adressage avec l'image d'un nouveau programme fs/exec.c:do_execve().

Terminaison d'un thread : **exit()**, *kernel/exit.c:exit()*

Linux : ordonnancement

Changement de contexte T1 => T2

- sauvegarde contexte T1
- restauration contexte T2

Sous Linux, le contexte est sauvegardé sur la pile (macros SAVE_ALL et RESTORE_ALL dans *asm/i386/entry.S*).

Ordonnanceur à temps partagé : répartition équitable du temps, selon des priorités.

L'ordonnanceur est dans *kernel/sched.c:schedule()*.

Appellé périodiquement, ou sur blocage.

Linux : appel système

Applications utilisateur => noyau : **appel système**.

Sous Linux : interruption logicielle 0x80

Numéro de fonction demandée dans *eax.*

Table des appels systèmes :
arch/i386/entry.S:syscall_table

Le traitement des appels se fait sur une autre pile. Chaque thread a donc 2 piles :

- une **pile utilisateur**, pour l'exécution de l'application
- une **pile noyau**, pour l'exécution des appels systèmes et la sauvegarde du contexte

Linux : synchro dans le noyau

Opérations atomiques

- Évitement des accès concurrents : impossible d'être interrompu.
- Deux types :
  - Sur les bits (test/set/change/clear)
  - Sur les variables (set/sub/inc_and_test)

Spinlocks

- Désactivation des interruptions
- Opération exécutée en un seul tenant
- En SMP, attente active
- Impossible de bloquer

Sémaphores

- Compteur
- Pas d'attente active
- Possibilité de bloquer

Linux : synchro pour l'utilisateur

Sémaphores
- Même principe que les sémaphores noyau
- semget(), semctl(), semop() (ipc/sem.c)

Files de messages
- Mécanisme de synchro et de communication
- msgget(), msgctl(), msgsnd() (ipc/msg.c)

Fichiers
- Moyen de communication (*pipe* et *fifo*)
- Moyen de synchro (*fcntl)*

Signaux
- Communication basique : déclenchement d'une routine dans d'autres processus.

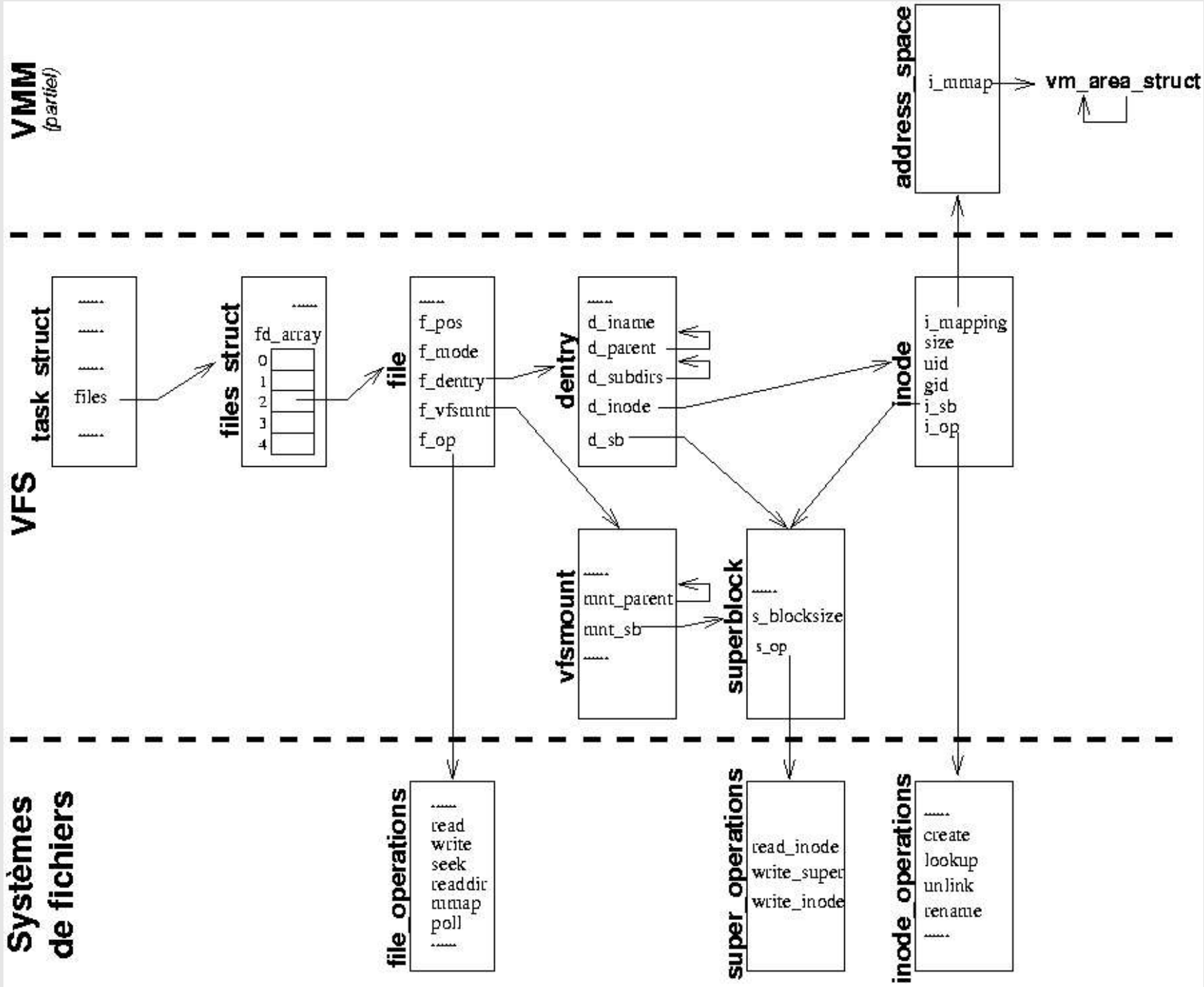
Mémoire partagée
- Région virtuelle identique dans plusieurs espaces d'adressage.

VFS : Virtual File System

- cohérence de l'espace de nommage global
- interface de programmation unique
- intégration avec les autres sous-systèmes, en particulier mémoire virtuelle

Un schéma vaut mieux qu'un long discours...

Linux : fichiers



Pilote de périphérique : représenter une ressource matérielle sous la forme d'une ressource logique.

Certains pilotes accessibles directement par l'utilisateur, d'autres non.

Types de périphériques :

- bloc
- caractère

Mais également, des périphériques internes, comme les cartes réseaux, utilisées via des piles de protocoles.

Pas de détails sur chaque périphérique, un simple détour sur l'interaction avec le matériel.

Linux : interaction avec le matériel

Trois modes d'accès :

- Ports d'entrées sorties
  - Petites zones pour dialoguer avec le périphérique
  - *inb / outb*
- Périphériques mappés
  - Une partie de la mémoire physique est “déroutée”
  - Pour dialoguer, écrire et lire dans la mémoire physique
- DMA : Direct Memory Access
  - Transfert de données direct entre périphérique et mémoire, sans utiliser le processeur.

Linux : interaction avec le matériel

Signalement d'événements : interruptions

- Exceptions : interne au processeur
  - Division par zéro
  - Défaut de page
  - Erreur de protection
- IRQ : du matériel
  - Timer
  - Clavier
  - Disque
  - Carte réseau
  - ..
- Appel système : du logiciel

Report du traitement :
- tasklets et softirqs

Kid Operating System - KOS

- Lancé en juin 1998
- Développé « from scratch »
- Jusqu'à 10 développeurs
- A l'heure actuelle, 3 développeurs actifs
  - David Decotigny
  - Julien Munier
  - Thomas Petazzoni
- **http://kos.enix.org**

KOS : Objectifs

- **Pour les développeurs : apprendre**
  - à programmer
  - à travailler de manière collaborative et distribuée
  - comprendre l'architecture x86
  - comprendre le fonctionnement d'un OS
  - debuggage et tests
  - **s'amuser**
- **Pour les autres**
  - Bibliographie
  - Documentations
  - Code source (GPL)
- Aucune application pratique prévue

Typique d'un projet Logiciel Libre :

- CVS
- Web
- Mailing lists
- Rencontres physiques

Méthode « on the fly »

- Conception pour le court terme
- Essai d'avoir quelque chose de satisfaisant d'un point de vue implémentation
- Améliorations itératives

KOS : un système modulaire

- Noyau de KOS découpé en modules, jusqu'au coeur
- 25-30 modules
  - scheduler
  - vmm
  - pmm
  - arch/mm
  - arch/task
  - klavier
  - fs/devfs
  - fs/fat
  - ...
- Modules reliés au boot

- **Avantages**
  - Interfaces claires entre les sous-systèmes
  - Séparation code portable / non-portable
  - Chargement/déchargement (non implémenté)

KOS : exemple de module

```
#include <loader/mod.h>

int hello_word(void)
{
   printk(``Hello World'');
   return 0;
}

__init_text static int
  post_init_module_level3 (kernel_parameter_t *kp)
{
   UNUSED(kp);
   printk(``Hello World init ... Ok'');
   return 0;
}

DECLARE_INIT_SYMBOL(post_init_module_level3,
                    POST_INIT_LEVEL3);
EXPORT_FUNCTION(hello_word);
```

KOS : interface utilisateur/noyau

- Fonctionnalité originale pour la gestion des ressources :

Karm

- Remplacement du **ioctl** Unix
  - `int ioctl(int d, int request, ...);`
- Ouverture d'une ressource selon une interface
  - `int open(char *path, unsigned interface);`
- Appel système
  - Numéro de ressource
  - Numéro de méthode
  - Paramètres

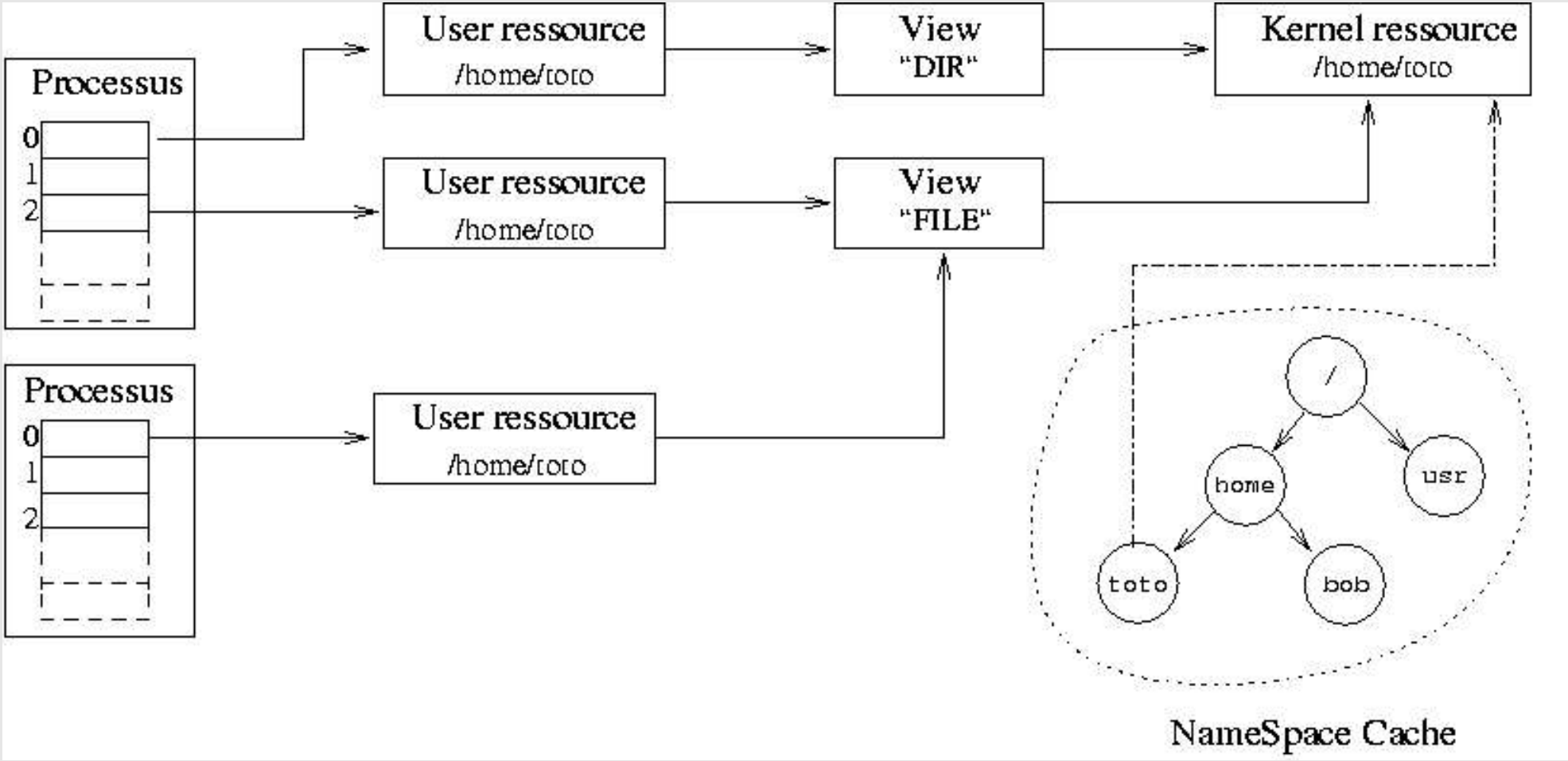
KOS : définition des interfaces

Interfaces définies en XML pour générer :

- définitions coté noyau
- définitions et stubs coté utilisateur

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<interface name="block">
 <method name="read">
  <arg type="struct ures*" name="ur"/>
  <arg type="char*" name="buffer"/>
  <arg type="count_t" name="block_start"/>
  <arg type="count_t*" name="inout_block_count"/>
  </method>
 <method name="write">
  <arg type="struct ures*" name="ur"/>
  <arg type="const char*" name="buffer"/>
  <arg type="count_t" name="block_start"/>
  <arg type="count_t*" name="inout_block_count"/>
 </method>
```

KOS : fonctionnement général



Bibliothèques utiles : libcharfile, libblockfile, libfilemap.

KOS : état actuel

- Système modulaire (chargement par Grub + loader)
- Gestion mémoire physique
- Gestion mapping mémoire
- Gestion mémoire virtuelle
- Allocateur mémoire pour le noyau
- Gestion des teams, threads noyaux et utilisateur
- Système karm avec appel système
- Pilotes de périphériques : disque, partition, console, clavier, série
- Systèmes de fichiers : FAT + devfs
- Gestion des interruptions
- Primitives de synchronisation pour le noyau
- Chargeur ELF
- Outils de débuggage

KOS : état actuel

Appels systèmes disponibles

- fork
- exec
- brk
- getpid
- getppid
- open
- close
- read/write minimaux

Résultat :

- Une mini bibliothèque C faite maison
- Un programme ELF chargé depuis le disque :
  - ouvre un fichier
  - se fork()
  - exécute un autre programme
  - créé des threads utilisateurs
  - alloue de la mémoire sur le tas

- Revoir toute la synchronisation dans le noyau
- Nouveaux appels systèmes
- Portage GNU libc

Bibliographie

- KOS. **Kid Operating System**
  http://kos.enix.org
- Tigran Aivazian. **Linux Kernel 2.4 Internals.**
  http://www.mc.man.ac.uk/LDP/LDP/lki/lki.html
- Andries Brouwer. **A small trail through the Linux kernel**
  http://www.win.tue.nl/~aeb/linux/vfs/trail.html
- Jonathan Corbet. **Porting device drivers to the 2.5 kernel**
  http://lwn.net/Articles/driver- porting/
- Mel Gorman. **Understanding the Linux Virtual Memory Manager**
  http://www.csn.ul.ie/~mel/projects/vm/
- Hans-Peter Messmer. **The Indispensable PC Hardware Book**
  Number ISBN 0201403994, Addison-Wesley
- Alessandro Rubini and Jonathan Corbet. **Linux Device Drivers**
  http://www.xml.com/ldd/chapter/book/index.html
- Andrew Tanenbaum. **Systemes d'exploitation**.
  Number ISBN 2100045547. InterEditions / Prentice Hall
- Uresh Vahalia. **UNIX Internals : The New Frontiers**.
  Number ISSN 0131019082. Prentice Hall.

Inutile de noter : http://thomas.enix.org/pub/conf/libreast2004/

Merci à :

- David Decotigny
- Julien Munier
- Les développeurs de Bochs, Grub et Linux
- organisateurs de Libr'east

Thomas Petazzoni

- étudiant ingénieur en Génie Informatique à l'UTBM
- **à la recherche d'un stage de fin d'études**

Questions ?