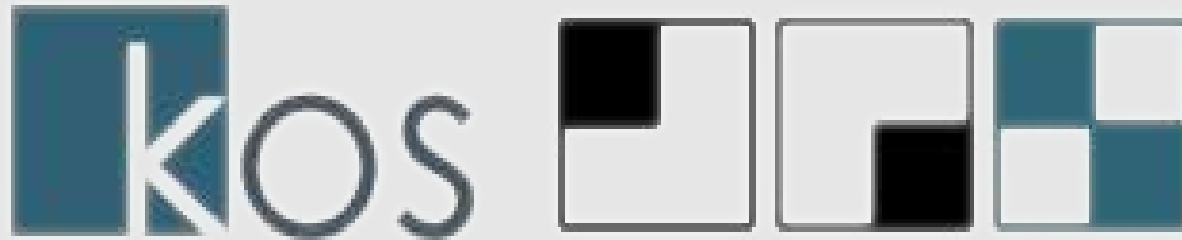


KOS - Kid Operating System

Ou comment réinventer la roue...



David Decotigny - Thomas Petazzoni

9 juillet 2004

- 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>**

- **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

Fonctionnalités « originales » de KOS

- Modulaire
- Gestion mémoire
- Accès et gestion des ressources

État actuel

Perspectives

KOS : un système modulaire

- Noyau de KOS « multi-modules »
- 25-30 modules
 - scheduler
 - vmm
 - pmm
 - arch/mm
 - arch/task
 - klavier
 - fs/devfs
 - fs/fat
 - ...
- Édition de liens réalisée au boot
- **Avantages**
 - Interfaces claires entre les sous-systèmes
 - Séparation code portable / non-portable
 - Chargement / déchargement (non implémenté)

KOS : un système modulaire

stdlib.ro :

```
printf(...)  
void memcpy (...) { }
```

stdio.ro :

```
int printf(...) { }
```

string.ro

```
memcpy(...)
```


KOS : un système modulaire

stdlib.ro :

```
printf(...)  
void memcpy (...) { }
```

stdio.ro :

```
int printf(...) { }
```

string.ro

```
memcpy(...)
```

boot
(loader kos)

printf (ref)

memcpy (def)

printf (def)

memcpy (ref)

Mémoire

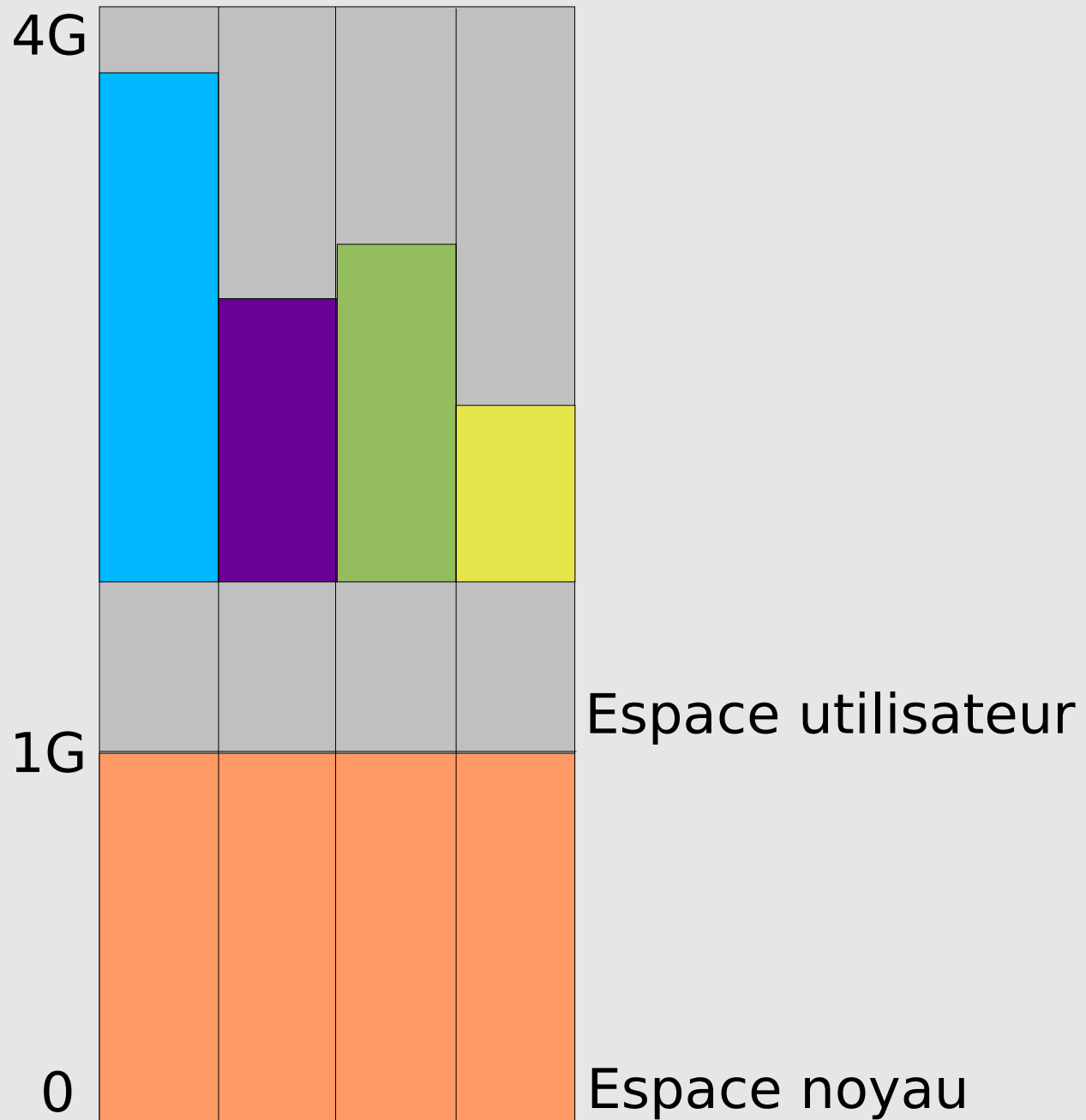
```
#include <loader/mod.h>

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;
}

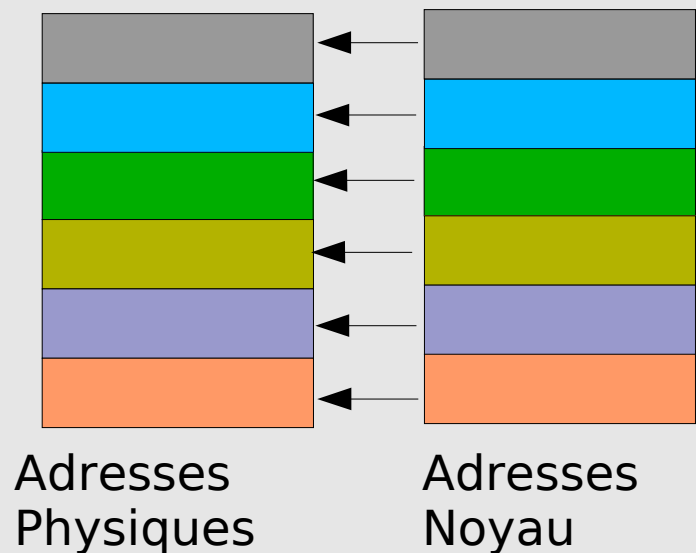
DECLARE_INIT_SYMBOL(post_init_module_level3,
    POST_INIT_LEVEL3);
EXPORT_FUNCTION(hello_word);
```

Espace noyau / espace utilisateur

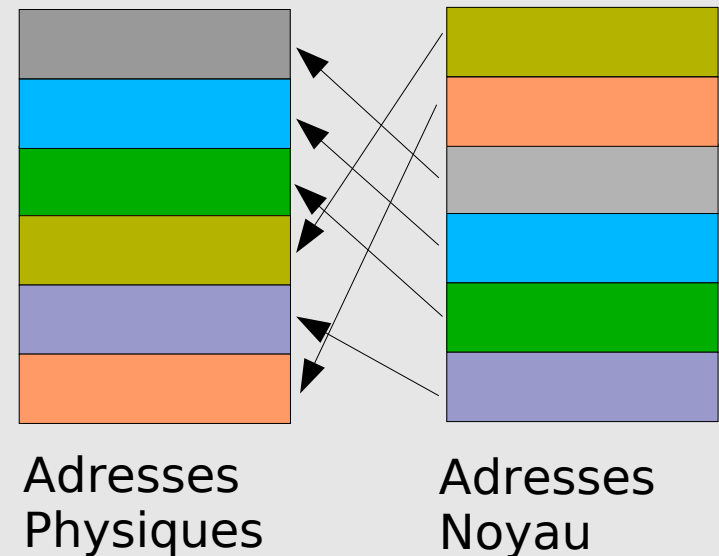


Gestion de l'espace noyau

- Dans Linux : 3 zones
 - DMA
 - « normale »
 - Highmem
- Dans KOS : pas de découpage de la mémoire physique en zones
 - Possible grace à l'absence d'*identity mapping*



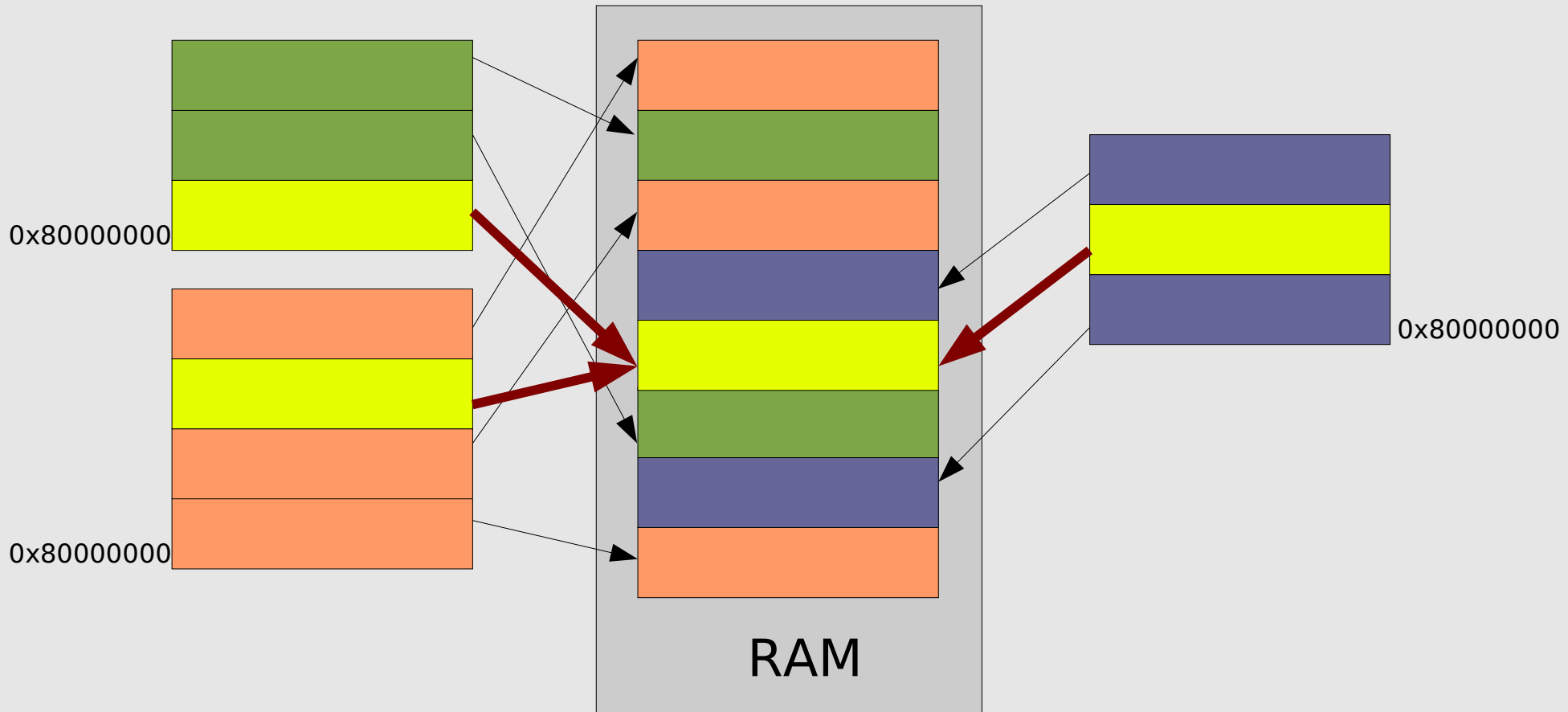
Linux



KOS

Gestion de l'espace utilisateur

- Comme l'espace noyau, repose fortement sur la pagination
- Différence : partage de pages possible entre processus



Difficulté principale : *swapping*

- connaître à quelles adresses virtuelles de quel(s) processus correspond telle page physique

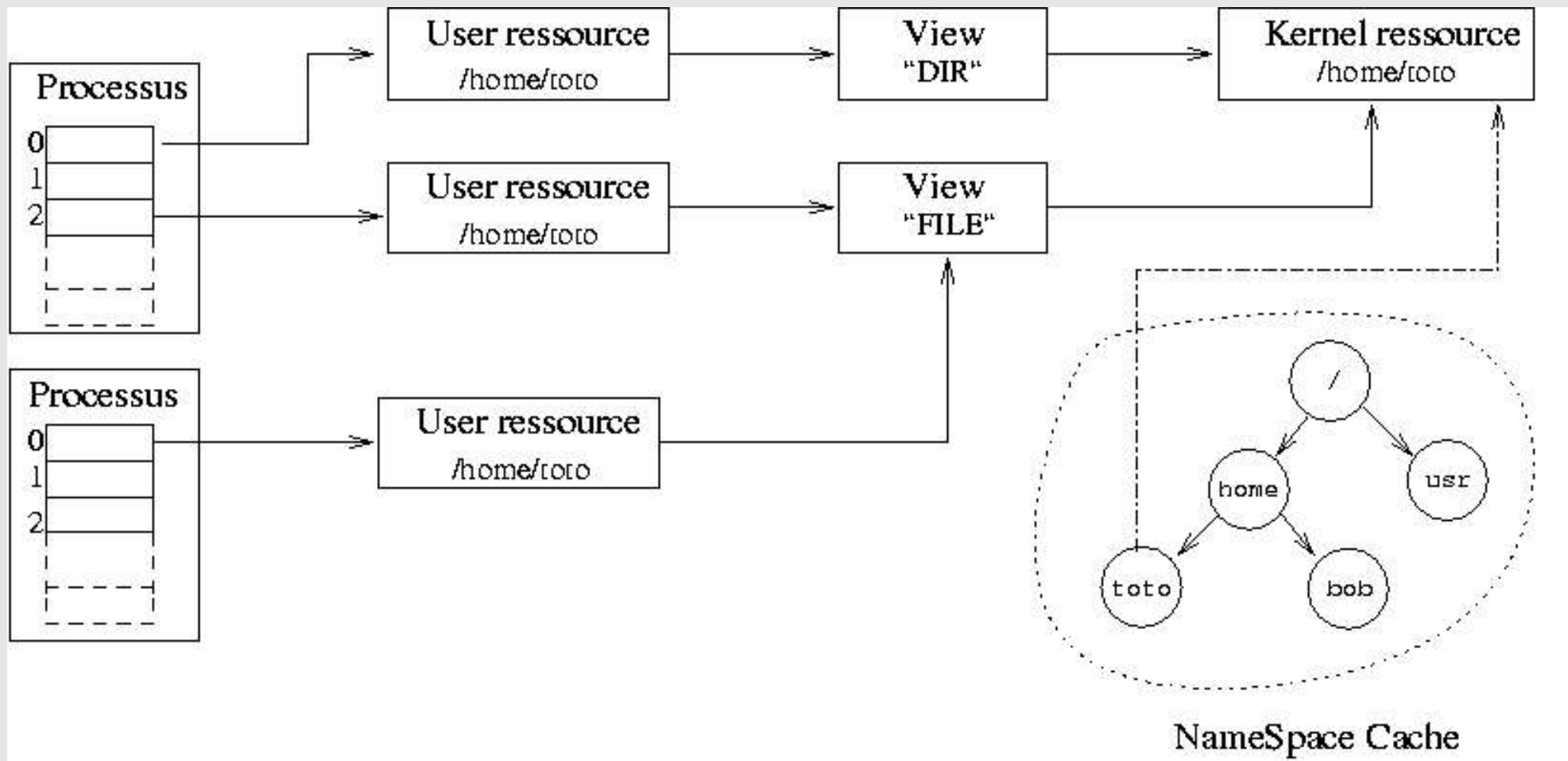
Solution : reverse mapping

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

Fonctionnement général



Bibliothèques utiles : libcharfile, libblockfile, libfilemap.

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>
  ...
</interface>
```


Liste des interfaces reconnues par le système :

```
struct interface interface_array[] = {  
...  
[INTERFACE_BLOCK_ID] = { "block" ,  
                        INTERFACE_BLOCK_NB_OPS,  
                        NB_ARGS_ARRAY(4, 4, 3, 3) },  
...  
};
```

Définition « C » de l'interface :

```
struct __kos_interface_block {  
    result_t (*read)(struct ures* ur, char* buffer, count_t block_start, count_t*  
inout_block_count);  
    result_t (*write)(struct ures* ur, const char* buffer, count_t block_start,  
count_t* inout_block_count);  
    ...  
};
```

Définition des *stubs*

```
#define KOS_INTERFACE_BLOCK_ID 1
#define KOS_INTERFACE_BLOCK_READ 0
extern result_t __kos_sys_block_read (int /* rd */, char* buffer, count_t
block_start, count_t* inout_block_count);
```

Implémentation des *stubs*

```
result_t __kos_sys_block_read (int __rd__, char* buffer, count_t block_start,
count_t* inout_block_count)
{
    return __kos_do_syscall3 (__rd__, KOS_INTERFACE_BLOCK_READ,
        (unsigned long)buffer,
        (unsigned long)block_start,
        (unsigned long)inout_block_count);
}
```

Implantation d'une méthode

```
static result_t
_ide_block_read(struct ures *ures, char *buffer,
                count_t block_start,
                count_t *inout_block_count)
{
    ...
}
```

Implantation d'une interface par un driver

Déclaration des méthodes :

```
struct INTERFACE_BLOCK _ide_blockops =  
{  
    .read = _ide_block_read,  
    .write = _ide_block_write,  
    ...  
};
```

Enregistrement des méthodes dans le système :

```
result_t _ide_register_disk()  
{  
    struct ures *block_ures;  
    struct kres *kres = kmalloc(sizeof(struct kres));  
    struct view *view = kmalloc(sizeof(struct view));  
    ...  
    view->iid      = INTERFACE_BLOCK_ID;  
    view->ops      = INTERFACE_OPS(& _ide_blockops);  
    view->view_data = device;  
  
    result = kres_add_view(kres, view);
```

Par le noyau :

```
result = open("/dev/disk/ide0", INTERFACE_BLOCK_ID, & hd);  
...  
block_count = 1;  
result = BLOCK_OPS(hd->view->ops)->read(hd, buffer, 0,  
                                         & block_count);  
...
```

Par les applications :

```
fd = open (filename, KOS_INTERFACE_BLOCK_ID);  
...  
sz = read(fd, buffer, sizeof(buffer));
```

- Système **modulaire** (chargement par GRUB + loader)
- **Chargeur** ELF
- Gestion mémoire **physique**
- Gestion mémoire **virtuelle**
- Gestion des **interruptions**
- **Primitives de synchronisation pour le noyau**
- Outils de **débuggage**
- Allocateur mémoire pour le noyau de type **Slab**
- 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

Appels système 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
- Pilote de carte réseau NE2000 et implantation d'une petite pile réseau

Depuis Juin 2004

- Parution d'articles (11) dans Linux Magazine France

Caractéristiques de SOS :

- Reprend l'expérience et quelques caractéristiques de KOS
- Monoprocasseur x86 seulement
- Noyau monolithique non préemptif
- Multitache préemptif mono-utilisateur
- Gestion mémoire physique et noyau comme Kos
- Pilotes basiques (écran, clavier, IDE)
- Systèmes de fichiers basiques
- Applications utilisateur ELF, quelques syscalls dont mmap

- 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/lld/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/rmll2004/>