# Kid Operating System

Thomas Petazzoni

`http://kos.enix.org`

# Outline

- ✔ Why this presentation ?
- ✔ What is *KOS* ?
- ✔ What is *KOS* for ?
- ✔ How do we work ?
- ✔ Main features
- ✔ Status and future work

# Why this presentation ?

- ☞ *Present* our project
- ☞ *Share* our little experience
- ☞ Get some *feedback*
- ☞ And maybe find some people to help us (development, documentation ...) !

# What is *KOS* ?

- ☞ *K* $\longrightarrow$ historical origin
- ☞ *O*perating *S*ystem
- ☞ Distributed under the *GNU GPL*

# What is *KOS* for ? (1/2)

- For the project members : *learn*
  - Programming
  - Distributed and *collaborative* work
  - Acquisition of an extensive *knowledge* of an architecture (IA32)
  - Good expertise of *OS design and realisation*
  - Debugging and testing

# What is *KOS* for ? (2/2)

- For those curious about OS programming
  - Bibliography
  - Documentation about details of the implementation
  - Source code (GPL)
- The goal is not to create the most perfect OS !

# How do we work ? (1/2)

- Technical organisation (currently 3 main developers disseminated accross France)
  - CVS
  - Web → documentation + CVSWeb + LXR
  - Mailing lists
- Typical open source project organisation.

# How do we work ? (2/2)

- Methodology : "*on the fly*" programming
  - Conception for the short term
  - Strive for producing a well designed implementation
  - Close to *extreme programming* methods

# Main features – Loader

*Modular structure* : a loader and many modules

- Boot sequence
  - GNU Grub loads loader and modules, then runs the loader
  - Modules are *linked* together by the loader (elf32, ar)
  - Kernel *initializes* modules
  - Kernel starts

# Main features – Loader

- Loader goals
  - Enforce *clean interfaces* between functionnalities of the kernel
  - Distinguish portable mechanisms from non portable ones
  - Ease development of modules

*Kos Loader* useful for other operating systems !

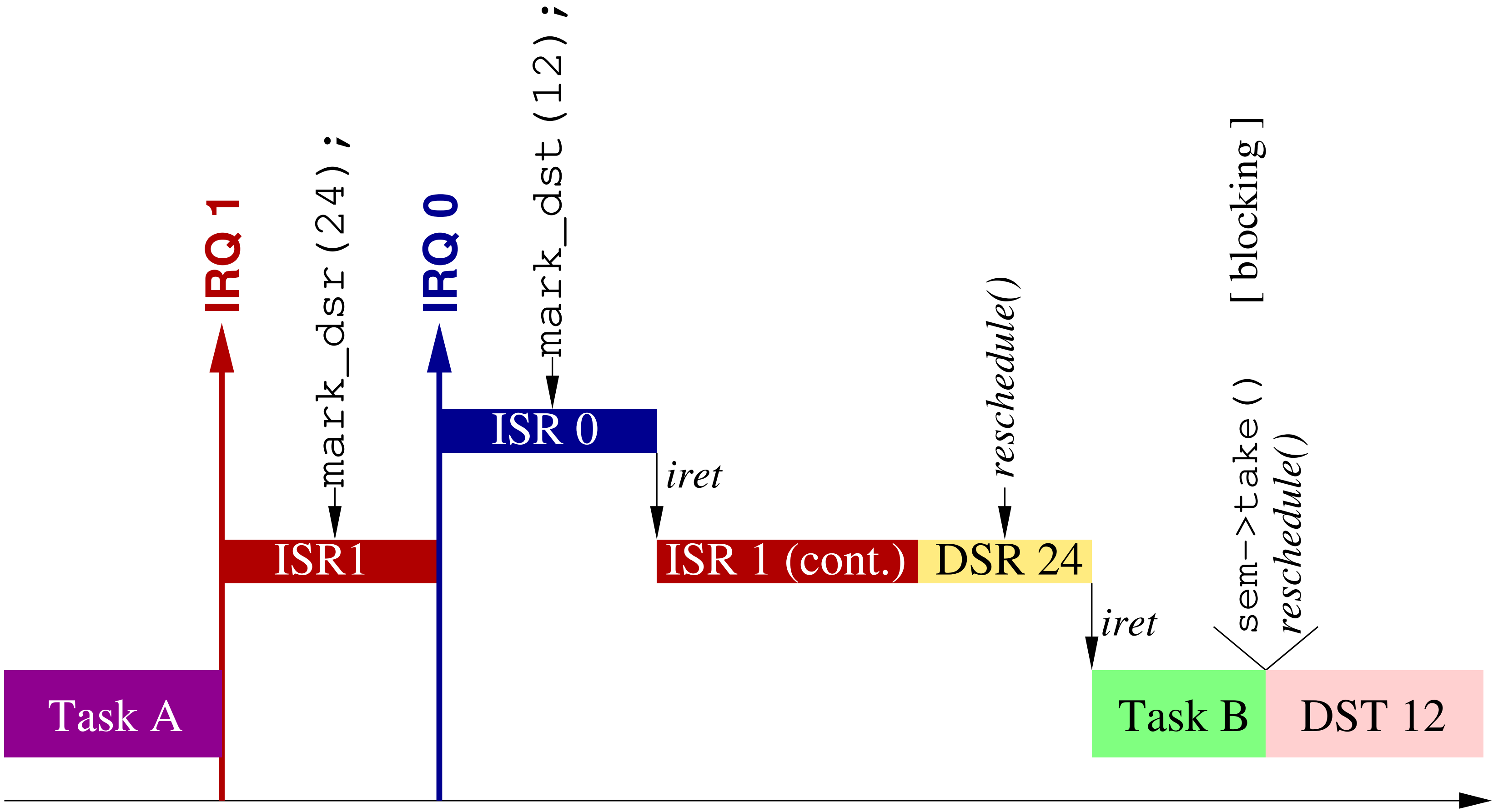
# Main features – A dummy module

```
#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);
```

# Main features – Multi-levels interrupts

- ✓ *Wish* : maximal lattitude while programming interrupt handlers
- ✓ *Approach* : 3 levels scheme
  1. *ISR* : *non interruptable* service handler
  2. *DSR* : *lock-free & deferred* interruptable service handler
  3. *DST* : blockable interruptable service *thread*

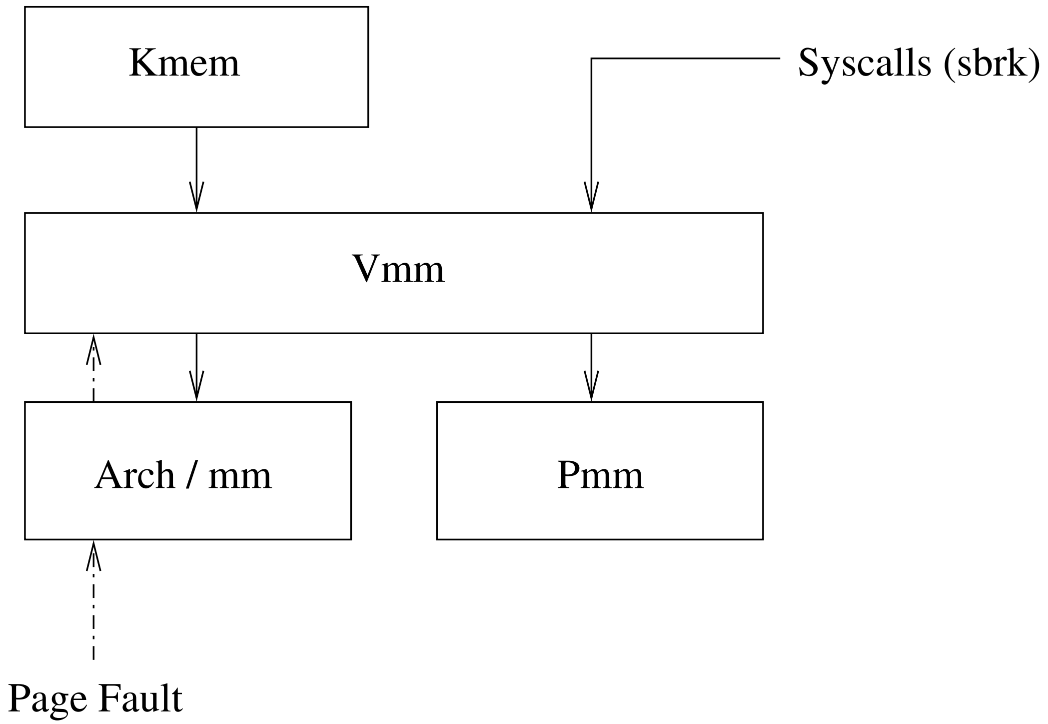
# Main features – Multi-levels interrupts



# Main features – Virtual memory

- Ideas close to the SVR4 VM Layer
  - Physical memory considered as a *cache* for virtual memory
  - *Object-oriented* approach suitable for portability

# Main features – Virtual memory




- Clean interfaces

  *arch/mm* : arch-dependent code *pmm* : physical memory management

  *vmm* : virtual memory management *kmem* : kernel memory management

# Main features – Virtual Memory

- ✓ vmap and *rmap*
- ✓ no identity mapping
- ✓ ability to move any pages, even kernel code !
- ✓ use of *slabs* for kernel memory allocation

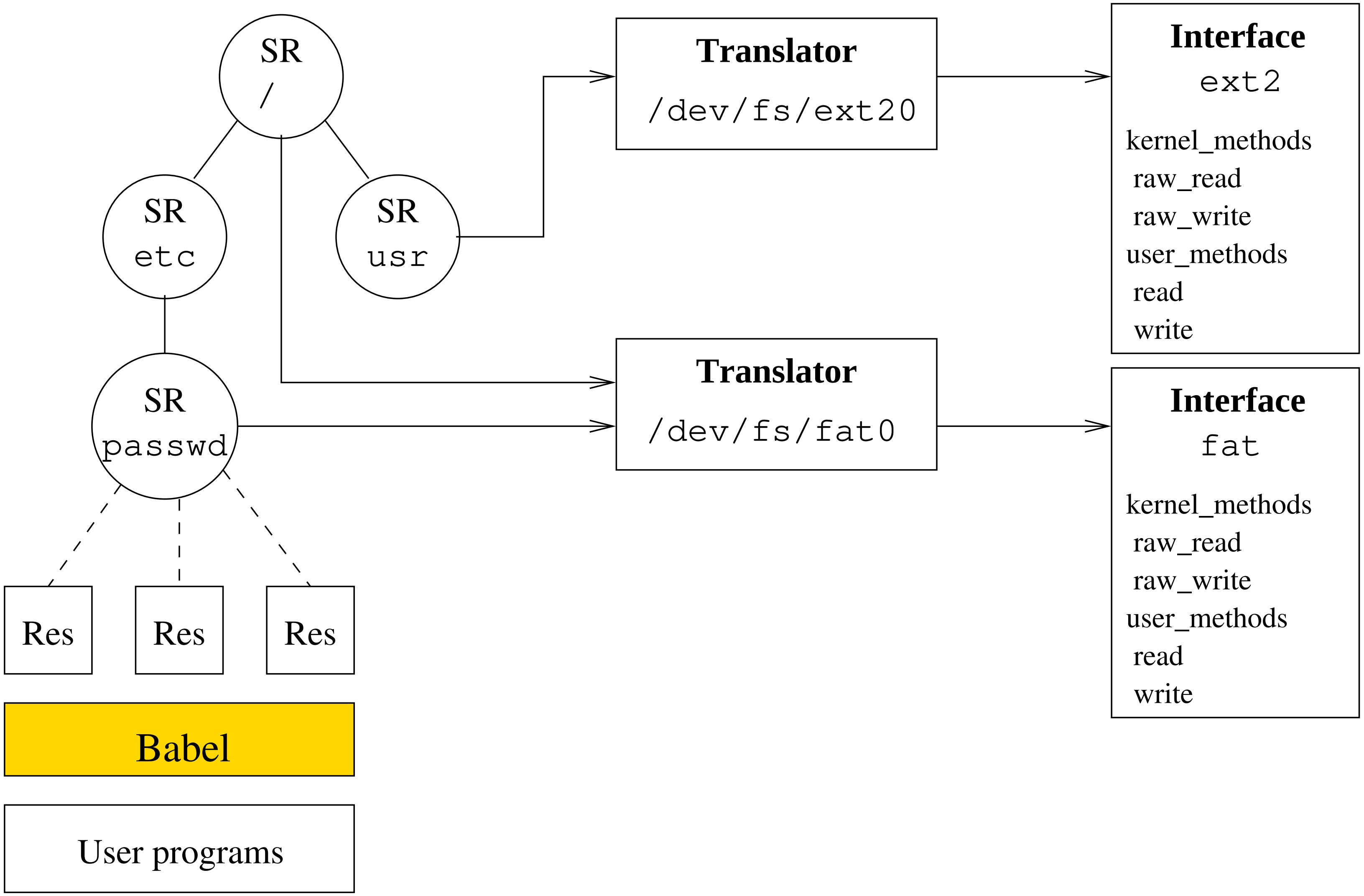
# Main features – *Babel*

- ✓ *Wish* : cleanly extend classical open/close/read/write/*ioctl* VFS interface.
- ✓ *Approach* :
  - Each driver exports an interface made of *methods* for user libraries and *methods* for kernel
  - The *Babel tower* is in charge of *registering and managing* these interfaces
  - An interface generator for the programmer

# Main features – Babel – Current status

- ✓ Different paths explored :
  - ➥ Object oriented interfaces in C : *interface*, *translator*, *shadow resource* and *resource*.
  - ➥ A code parser in Caml : stub and code generation
- ✓ Problems :
  - ➥ Syscall communication is very restrictive
  - ➥ No type check
  - ➥ *libc* hard to implement
  - ➥ Unix compatibility isn't so easy ;)

# Main features – Babel – Current status



# Status

- ✓ Loading and linking modules (elf32, ar)
- ✓ Interrupt management and IRQ multi-level system
- ✓ Kernel and user level multithreading
- ✓ Memory management : physical, virtual and kernel allocators
- ✓ Kernel level IPCs

- ✓ Babel implementation + related problems ;)
- ✓ Debugging facilities
- ✓ Basic SMP (fine grained locking) and portability considerations
- ✓ Basic IDE, FAT, console and keyboard drivers
- ✓ Primitive *libc*
- ✓ Execution of real user applications cross compiled from Linux
- ✓ Lots of nice bugs ;)

# Future work

- Redesign user$\leftrightarrow$kernel dialog through Babel, and through stub and code generation.
- Finish VMM : fork, exit, mmap, COW, etc...
- Improve drivers (IDE, Fat, console, keyboard)
- Work on SMP and portability compliance
- Implement new drivers
- Write some documentation

# Thanks

- ✓ David Decotigny and Julien Munier, developers
- ✓ Grub developers
- ✓ Bochs developers
- ✓ Richard M. Stallman for his wonderful music ;)

Kos : `http://kos.enix.org`

Bochs : `http://bochs.sourceforge.net`