Kid Operating System

David Decotigny, Thomas Petazzoni

http://kos.enix.org

Outline

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

Why this presentation ?

- ☞ *Present* our project
- ☞ *Share* our little experience
- ☞ Get some *feedback*

What is *KOS* ?

- ☞ *K* → historical origin
- ☞ *O*perating *S*ystem
- ☞ Distributed under the *GNU GPL*

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

- For those curious about OS programming
  - Bibliography
  - Documentation about details of the implementation
  - Source code

How do we work ? (1/2)

- Technical organisation ($\Leftarrow$ 4 main developers disseminated accross France)
  - CVS
  - Web $\rightarrow$ documentation + CVSWeb
  - Mailing lists

How do we work ? (2/2)

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

Main features (1/6) – Loader

- Loading phase
  - *Modular structure* : the loader + ELF modules
  - GNU Grub loads them
  - Modules *linked* together by the loader
  - Kernel starts

Main features (2/6) – Loader

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

Main features (3/6) – Dynamic kernel stack

- ✓ Linux uses static kernel stacks for CPL0 (8 KB): stack overflow would overwrite task management structure
- ✓ *Wish*: allow to program assuming larger stacks with reasonable memory usage
- ✓ *Solution*: use of double fault for demand paging of CPL0 kernel stacks
- ✓ *Drawback*: not suitable for hard real time

Main features (4/6) – Multi-levels interrupts

- ✓ *Wish*: maximal liberty while programming interrupt handlers
- ✓ *Approach*: 4 levels scheme
  1. *clISR*: *non interruptable* service handler
  2. *stISR*: *immediate & interruptable* service handler
  3. *DSR*: *lock-free & deferred* interruptable service handler
  4. *proxy*: blockable interruptable service *thread*

Main features (5/6) – 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 (6/6) – *Babel*

✓ *Wish*: cleanly extend classical open/close/read/write/*ioctl* VFS interface.

✓ *Approach*:

- Each driver exports an interface made of *methods* for CPL3↔CPL0 dialog
- A service (*root interface*) is in charge of *registering and managing* these interfaces
- An interface generator for the programmer

Status

- ✓ Loading and linking modules
- ✓ Dynamic kernel stack expansion
- ✓ Basic interrupt handling framework
- ✓ Kernel level multithreading
- ✓ Kernel memory management and basic VMM
- ✓ Kernel level IPCs
- ✓ First Babel implementation
- ✓ Debugging facilities

Future work

- Complete IRQ multi-level system
- SVR4-style VMM
- User level multithreading
- CPL3↔CPL0 dialog through Babel
- SMP and portability compliance