ExAmour 0.2
Architecture overview

Remy SAISSY 10/13/2008

Key concepts

- Test driven development;
- Few libraries dependancy;
- Strongly related to target architecture;
- No dynamic module loading;
- No device knowledge in kernel space;
- Multiple address spaces;
- Fine grained I/O permissions;
- Dynamic events;
- Software Service Bus architecture.

Test driven development

- Use Eclipse configurations and Cunit;
- Test code takes precedence to kernel code;
- Every bug is reproduced in a test;
- Test suites run in a Linux environment;
- More than 90% of the kernel code is covered by tests.

Few libraries dependancy

- Two libraries
  - Subset of the klibc located on the ExAmour's repository;
  - Libslds, a library which provides memory allocation less algorithms for Storage and Lookup Data Structures.

Strongly related to target architecture

- Aims at using the target architecture at most;
- No abstraction of the architecture but...
- Stable interfaces through the Software Service Bus.

No dynamic module loading

- Once started, the kernel run by itself;
- Kernel space is completely unavailable;
- Drivers and other stuffs run in userspace.

No device knowledge in kernel space

- The kernel has processes, no users, ...
- The kernel knows only how to multiplex an access to a simple ressource;
- Simple ressources are :
  - Processor cycle;
  - I/O port;
  - Memory segments;
  - Machine events (Exceptions/Interruptions);
  - CPU registers

Multiple address spaces

- No virtual memory;
- Memory segments isolated by the CPU;
- Non overlapping segment;
- Not dynamically resizable segments.
- Exclusive segments are possible;
- Scope of exclusivity can be defined
  - Process scope;
  - System scope;

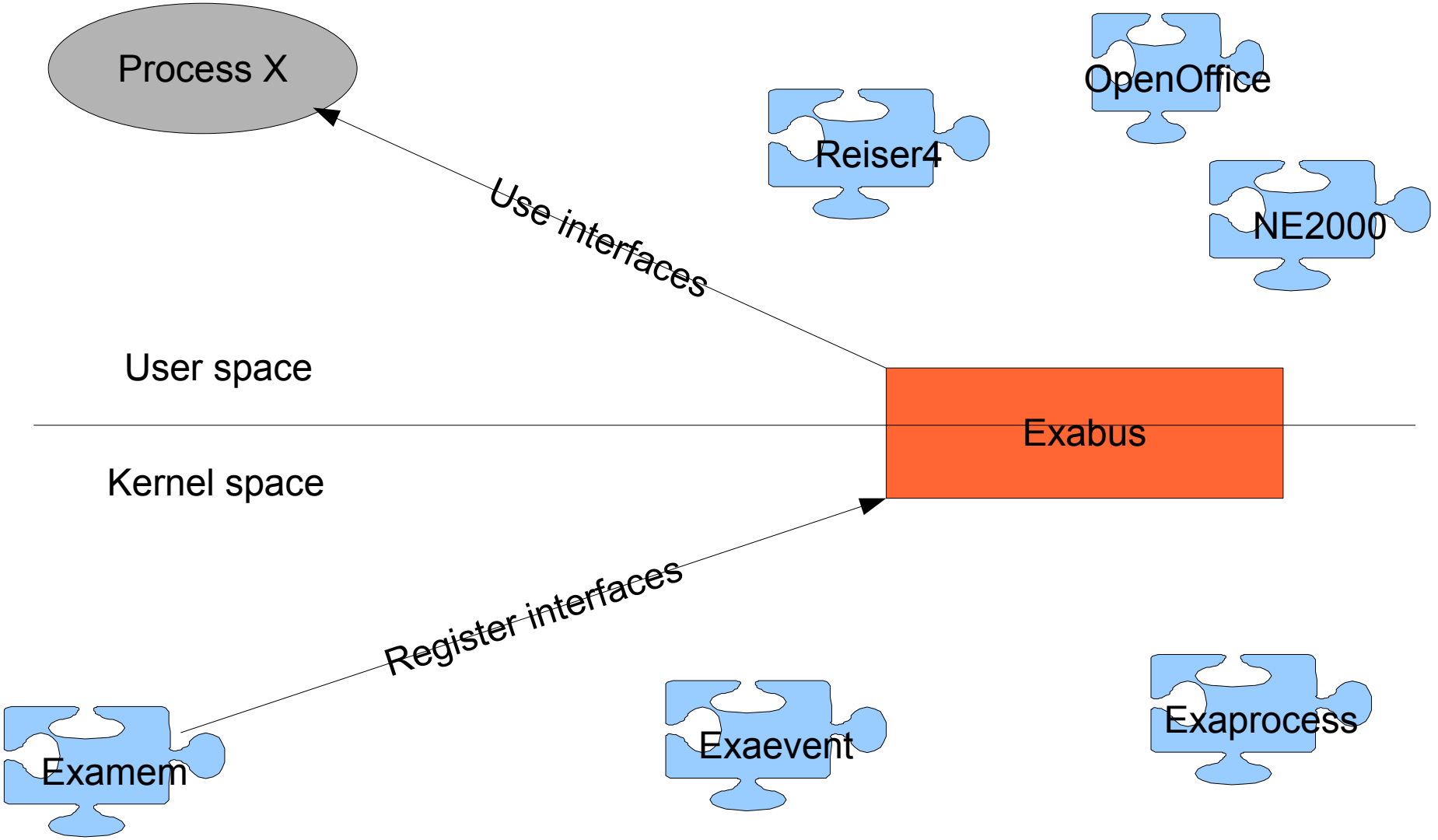
Fine grained I/O permissions

- I/O are granted on a port/process basis;
- Exclusive access is possible;

Dynamic events

- Unified mechanism for machine events and user created events;
  - exceptions/interruptions are machine created events;
  - Upcalls on_schedule, on_unschedule;
  - do_foo, do_bar are user created events;
- Events can be signals or procedures;
- Both use the Software Service Bus.

The Software Service Bus exabus 1/2



The Software Service Bus
exabus 2/2

- The kernel creates interfaces, like any process;
- An interface embeds procedures and routines;
- Low latency piece of software;
- Asynchronous communications.
- Namespace / interface / methods directory;
- NTree structure.
- Dynamic evolution along system lifetime.

Limited kernel space access

- Exabus is both user space and kernel space;
- Only one syscall that can be only run by exabus for « kernel.* » interfaces;
- Provider/consumer mechanism for exabus to feed asynchronous tasks to the kernel without kernel space crossing;
- User processes use simple software interrupts (kind of long call) to access talk to exabus.

For what kind of use ?

- To be tested and validated by experts but:
  - Embedded systems;
  - Virtual machines (jvm, clr, mozart, ...);
  - Distributed systems.
- The goal is to try out other organizations, if possible to simplify things.

Conclusion

- Project stopped because of a lack of time for one person to make all of it :(;
- Strong trust in such a project so if you want to work in it...
- remy.saissy@gmail.com
- http://gna.org/.... type examour in the search box ;).
- svn://svn.gna.org/svn/examour/