

Complex Lab – Operating Systems

Introduction

Martin Küttler

Organisation

- ▶ Website (e.g. source repository):
<https://os.inf.tu-dresden.de/Studium/Praktikum/>
- ▶ Mailinglist: kpr2017@os.inf.tu-dresden.de
Subscribe at <https://os.inf.tu-dresden.de/mailman/listinfo/kpr2017>
- ▶ Repository: <https://os.inf.tu-dresden.de/repo/git/kpr.git>
- ▶ Send Code to: martin.kuettler@os.inf.tu-dresden.de

Organisation – Dates

24.10.17	Introduction
26.11.17	1st Assignment due
28.11.17	Sessions & Memory
09.01.18	Graphical Console
23.01.18	Keyboard Driver
31.03.18	Final Assignment due

Requirements

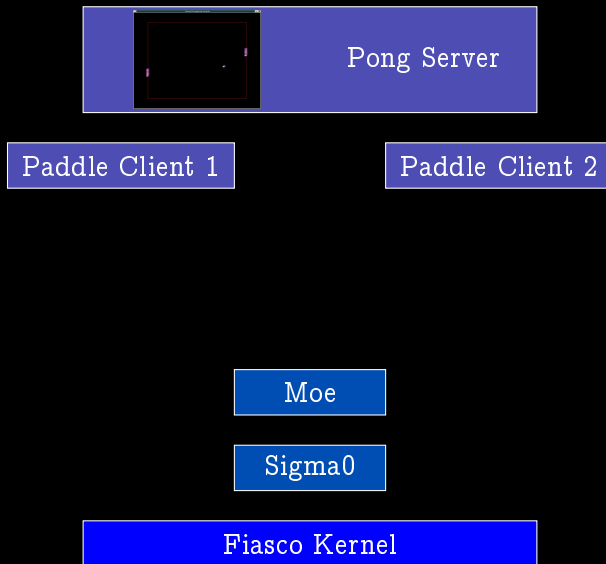
- ▶ Basic OS knowledge: threads, processes, virtual memory, drivers, ...
- ▶ Experience with Linux: editor, shell, development (make, gcc)
- ▶ Experience with C/C++

Goal of this course

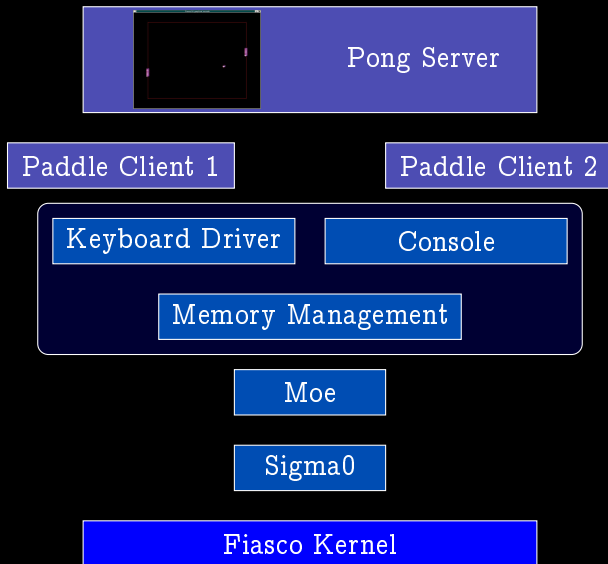
Multi-user Pong game



Goal of this course



Goal of this course



Required Software

- ▶ A Linux system (preferably Debian/Ubuntu)
- ▶ qemu
- ▶ build-essentials: gcc (<7), g++ (<7), make
- ▶ gcc-multilib, g++-multilib

The Repository

`doc/source/` documentation

`obj/l4/$ARCH/` build directory

`src/kernel/` Fiasco kernel sources

`src/l4/pkg/` sources for all userspace packages

`src/l4/conf/` boot configuration

To build everything in the beginning:

```
$ make setup
```

```
$ make [-j X]
```

Add `CC=gcc-6 CXX=g++-6` to `make setup` if your default compiler is gcc 7. Also write the following to `src/l4/Makeconf.local`:

```
CC=gcc-6
```

```
CXX=g++-6
```

```
OBJ_BASE=<full-path-to-build-dir>
```

Build Infrastructure

simple Makefile:

```
PKGDIR  ?=    ../..
L4DIR   ?=    $(PKGDIR)/../..

SRC_C   =     main.c
SRC_CC  =     file.cc

TARGET  =     my_program
REQUIRED_LIBS =

include $(L4DIR)/mk/prog.mk
```

Generate template Makefiles and file structure:

```
kpr/src/l4/pkg$ mkdir my_pkg
kpr/src/l4/pkg$ cd my_pkg
kpr/src/l4/pkg/my_pkg$ ../../mk/tmpl/inst
```

Headers

Package headers are in:

```
kpr/src/l4/pkg/my_pkg/include/my_header.h
```

They are installed to

```
kpr/obj/l4/x86/include/l4/my_pkg/my_header.h
```

Compiling

Compiling stuff

```
# build everything (e.g. in the beginning)
kpr$ make [-j X]
# build single pkg from build dir
kpr/obj/l4/x86/pkg/my_pkg$ make
# build single pkg from source dir
kpr/src/l4/pkg/my_pkg$ make [O=<BUILDDIR>]
```

Run:

```
kpr/obj/l4/x86$ make qemu [E=<entry>]
```

Qemu configuration in `kpr/obj/14/x86/Makeconf.local`

- ▶ `MODULE_SEACH_PATH`: Where binaries, config files, etc. are
- ▶ `QEMU_OPTIONS`: command line parameters

Boot configurations in `kpr/src/14/conf/modules.list`

entry name

kernel fiasco <parameters>

roottask moe <program/startup script>

module modules to load (one per module-line)

IPC overview

- ▶ External resources are accessed through capabilities.
- ▶ Applications have a namespace containing capabilities.
- ▶ Some magically appear at startup, others can be added in startup script.
- ▶ Communication requires capability to an IPC gate.

Lua

Example Ned config scripts can be found in

- ▶ `src/l4/conf/examples`
- ▶ `src/l4/pkg/l4re-core/ned/doc/`
- ▶ `src/l4/pkg/examples/clntsrv`

General idea for our simple usecase:

- ▶ Programs are started with `L4.default_loader.start`
- ▶ They take capabilities to channels (aka IPC gates) that are created with `L4.default_loader.new_channel`
- ▶ Programs can get a client or a server (`:svr()`) capability

IPC

In the program, these capabilities can be used to send and receive messages.

Server side:

- ▶ Register Server object (that implements handler) with the named cap.
- ▶ Run server loop
- ▶ Handle requests

Client side:

- ▶ Query IPC gate capability at name service.
- ▶ Invoke cap with arguments.

Sending data works with the UTCB (lecture). We use a high-level interface on top.

IPC interface

- ▶ Define a class that inherits from `L4::Kobject_t` that has a member function for each message type.

```
struct MyClass : public
    L4::Kobject_t<MyClass, L4::Kobject, MY_PROTO_NUM>
{
    L4_INLINE_RPC(int, foo, (int arg1, int arg2));
    typedef L4::Typeid::Rpc<foo_t> Rcps;
};
```

- ▶ Functions return error code.
- ▶ All functions are listed in type `Rcps`.

IPC interface

- ▶ The server implements a class that inheriting `L4::Epiface` that implements the methods.
- ▶ Types are converted between interface and implementation; special types for inout/out parameters, capabilities, arrays, etc. are available.
- ▶ If you need to send caps, the `L4::Kobject_t` needs a fourth template argument of type `L4::Type_info::Demand_t<NUM>`, with NUM being the maximum number of caps per call.
- ▶ See `src/l4/pkg/examples/clntsrv` for a example.
- ▶ See documentation `doc/source/html/index.html` (specifically e.g. `doc/source/html/l4_cxx_ipc_iface.html`)
- ▶ The source is the ultimate documentation (and it includes all the normal documentation).
- ▶ When you have questions, use the mailing list (kpr2017) or send me an email directly.

Assignment 1

- ▶ Download the repo, set it up and compile everything.
- ▶ Try it in Qemu, make sure that hello works.
- ▶ Build a client-server version of hello: The client should send a string to the server via ipc.
- ▶ As a second step, also consider large strings.
- ▶ Deadline on 26.11.17.

Next meeting

We meet next on 28.11.17, where we start the real project.