



Complex Lab — Operating Systems 2014 Winter Term

Introduction

Basic Operating Systems Know-How

- Virtual Memory (Paging/Page Tables)
- Memory Management
- File Systems
- Device Drivers
- Processes and Threads
- Operating System Structures (What is a microkernel?)

Experience with UNIX/Linux

- Editor (for Programming)
- Using a command shell
- Development Tools (GNU Make, Binutils, GCC)

Experience with C/C++ Programming Language

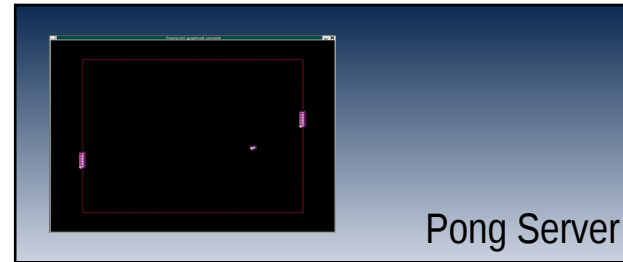
- Writing C Programs
- Using Pointers (Pointer Arithmetics)
- What is the Stack?

- Consultation roughly bi-weekly (Tue. 2.50PM, INF E08)
- Website:
<http://os.inf.tu-dresden.de/Studium/Praktikum/>
- Mailinglist (Announcements, Discussions):
<http://os.inf.tu-dresden.de/mailman/listinfo/kpr2014>
- Presentation to your tutor in the end
- System requirements
 - Linux (Debian/Ubuntu at best)
 - QEMU
 - ~600 MB disk space
- **Deadline** for Final Assignment: 31.03.2015
 - Results containing your source codes and documentation have to be sent to the tutor

Date	Topic
28.10.13	Introduction
23.11.13	1st Assignment
25.11.13	Sessions & Memory
9.12.13	Graphical Console
6.01.14	Keyboard Driver
18.01.14	Progress Report
20.01.14	Consultation
31.3.14	Final Assignment

Multi-User Pong game





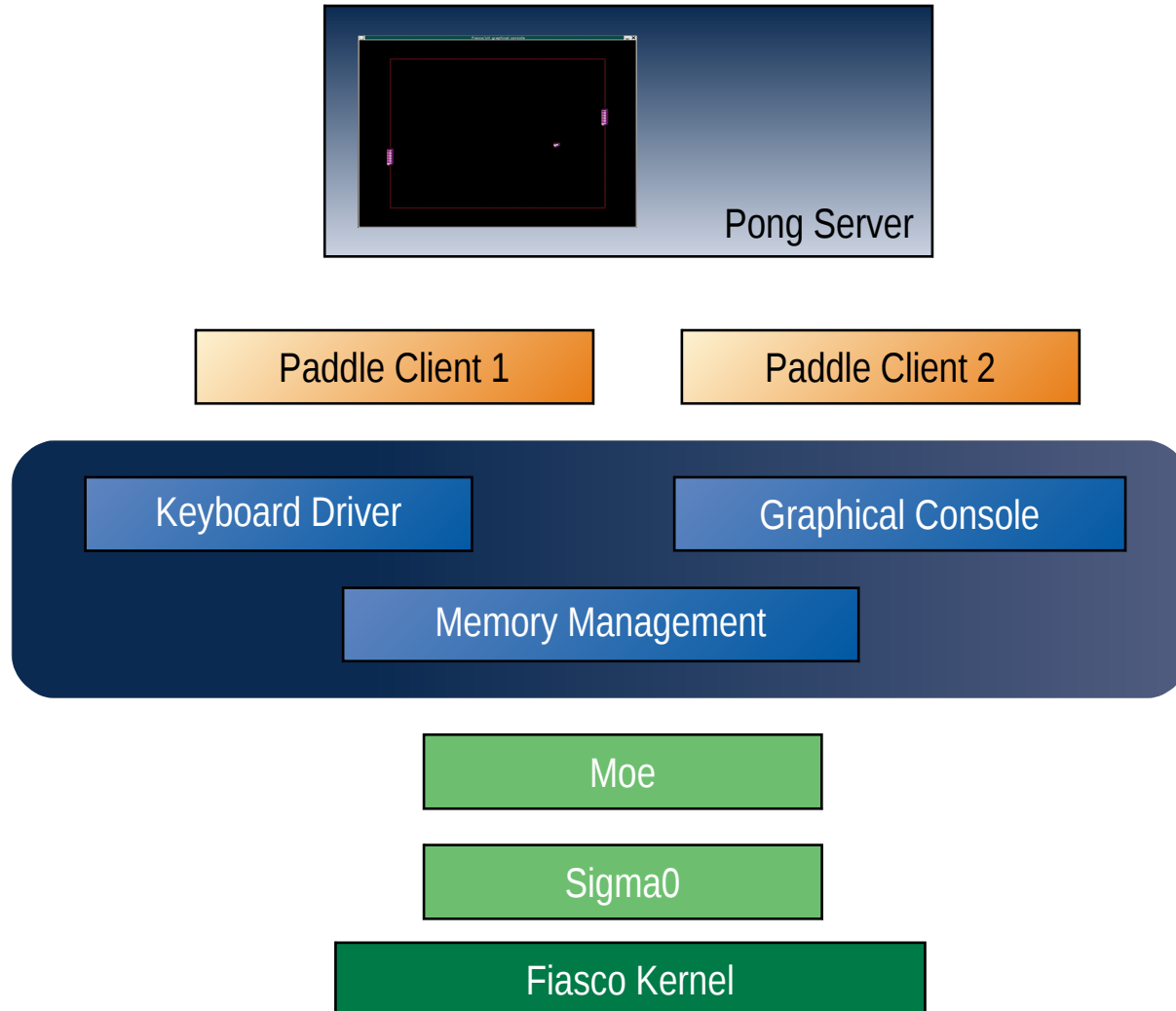
Paddle Client 1

Paddle Client 2

Moe

Sigma0

Fiasco Kernel



- Memory management
 - malloc() / free()
- Console
 - Graphical in/output, multiplexing
- Keyboard driver
 - Handle keyboard in/output
- File system
 - CD-ROM ISO file system
 - Store binaries, highscores etc.

- Subversion repository containing all source code:
 - <http://os.inf.tu-dresden.de/Studium/Praktikum/src/trunk/kpr/>
- Get acquainted to SVN – we may push updates throughout the semester.
 - git-svn may not work
- Documentation:
 - [doc/html/index.html](http://os.inf.tu-dresden.de/Studium/Praktikum/doc/html/index.html)

Components of the Framework

- Fiasco microkernel
- The Sigma0 memory manager
- Basic C/C++ Environment
 - LibC, IPC Streams, IPC Server Framework, ...
- Moe root server
- Ned – the init script interpreter
- Pong-Server and Pong-Example Client
- GNU make based build system
 - For the Fiasco kernel
 - For the user-level applications (BID)

- Building stuff:
 - \$> make setup
 - \$> make
- obj/l4/x86 build directory
- doc/ HTML documentation
- src/kernel/ Fiasco microkernel sources
- src/l4/ L4 sources
 - src/l4/mk BID – the build infrastructure
 - src/l4/pkg source packages
- bin/

BID is a collection of Makefiles and Makefile templates

- Support easy building of
 - Applications
 - Libraries
- Simple Makefiles for applications and libraries

Simple package Makefiles:

```
# directories we need to know
PKGDIR ?=      ../..
L4DIR  ?=      $(PKGDIR)/../..

# source files
SRC_C   =      main.c           # C sources
SRC_CC  =      extension.cc      # C++ sources

# build target
TARGET  =      my_prog

# additional libraries
REQUIRES_LIBS =

# include BID prog rule (compile an application)
include $(L4DIR)/mk/prog.mk
```

BID templates for new packages:

```
kpr/src/l4/pkg> mkdir my_pkg
```

```
kpr/src/l4/pkg> cd my_pkg
```

```
kpr/src/l4/pkg/my_pkg> ../../mk/tmpl/inst
```

Installs package directory structure and template Makefiles

Package include files:

`kpr/src/14/pkg/my_pkg/include/my_header.h`

will be installed to:

`kpr/obj/14/x86/include/14/my_pkg/my_header.h`

can be used by other packages using:

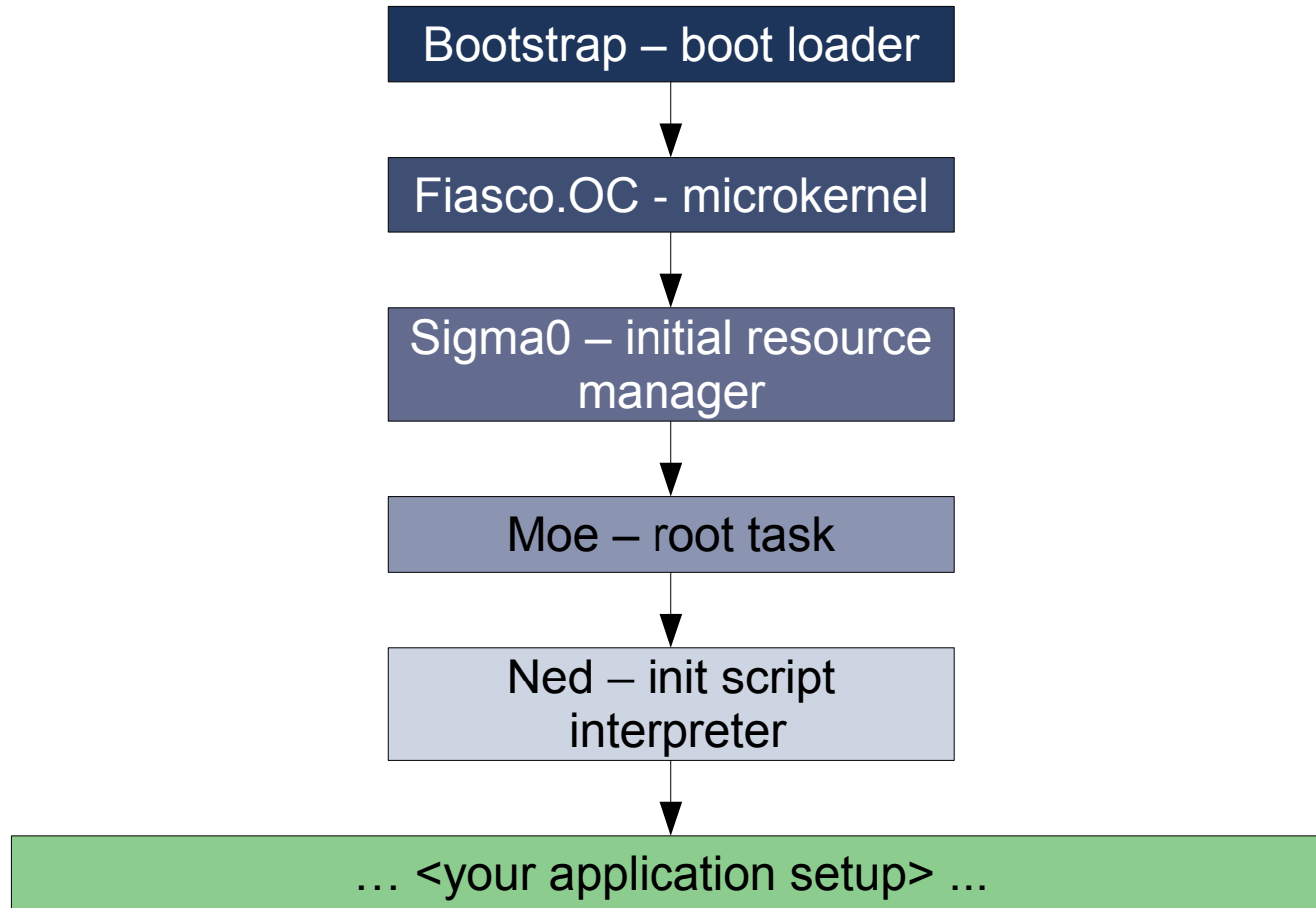
`#include <14/my_pkg/my_header.h>`

- All compiled files are kept outside the source directories
 - Default: kpr/obj/l4/x86
- You can compile stuff in 3 ways:
 1. make in kpr/ root → builds everything
 2. make in kpr/obj/l4/x86/pkg/my_pkg
→ builds single pkg
 3. make O=<builddir> in kpr/src/l4/pkg/my_pkg
→ builds single pkg
- Special build target: make qemu [E=<entry>]
 - Launches Fiasco in Qemu
 - <entry> is the menu.lst entry to boot
 - Omitting E=<> brings up a menu

- kpr/obj/l4/x86/Makeconf.local
 - QEMU settings
 - MODULE_SEARCH_PATH – where to look for binaries, config files, etc.
 - QEMU_OPTIONS – command line parameters for Qemu
- kpr/src/l4/conf/modules.list
 - Applications to boot for a given setup.
 - Similar to a GRUB menu.lst

- Specify a set of modules to boot
 - src/l4/conf/modules.list
 - Used to build a suitable config for your target
 - GRUB menu.lst entry
 - Qemu multiboot command line
 - Fiasco/UX command line
 - ...
- Basically: list all files to be loaded
 - **entry** → title
 - **kernel** → fiasco + command line
 - **roottask** → moe + parameters
 - **module** → everything else (no parameters here!)

Short overview: bootup sequence



- Moe gets 1 command line parameter
 - binary or startup script
- All boot modules available through ROM file system
- Startup script:
 - Scripting language: Lua (don't worry...)
 - Global config options
 - Name space definitions
 - Application's local view on its resources
 - Establish communication channels between apps
 - Applications
 - Command line parameters
 - L4Re options

- Package 'ned' contains documentation
 - pkg/ned/doc/hello.lua → shortest possible script
 - pkg/ned/doc/tutorial.lua → tutorial on how to setup things like channels etc.
 - pkg/ned/doc/Aw.lua → some complex things

- Capability system
- Applications have a name space containing capabilities
- Some appear by magic (at startup)
 - Parent, memory allocator, name service, ... → `L4Re::Env::env()`
- Others configured at startup/runtime
- IPC gates
- Server:
 - register IPC gate at name service
 - Run IPC server loop
 - Handle request (dispatch function)
- Client:
 - Query IPC gate at name service → capability
 - Invoke capability with arguments

- UTCB contains message data
- C:
 - Obtain UTCB pointer using `l4_utcb()`
 - Fill in data into `utcb->mr[0]`
to `utcb->mr[L4_UTCB_GENERIC_DATA_SIZE]`
 - Use `l4_ipc_call()` to invoke server
- C++:
 - Create `Ipstream` and pass the UTCB as parameter
 - Write data using stream operators
- Special: Buffers and Flexpages – see the docs

Interfaces are based on IPC messages

- C++ stream syntax to generate/read messages
- Server Framework to handle server objects
- Client Stub code to provide a client side API

Examples in kpr/l4/pkg/

./hello	simple Hello World Application, no IPC
./clntsrv	client/server IPC example

Demo: Starting client-server examples

- Server
- Registry
- Server_object

Server Interface:

dispatch Function of server object

- Interesting parameter: L4::Ipc_iostream& ios
 - Contains all parameters
- 1st parameter: message tag (protocol id + message descriptor)
- 2nd parameter: opcode
 - determines the method to call
- Write return values to ios again

- Allocate capability slot
- Query for server capability name
 - This is the name you put into the client's name space!
 - Result: cap slot filled with valid capability (check!)
- Create `L4::Ipc_iostream` and fill in parameters:
`L4::Ipc_iostream ios(l4_utcb());`
`ios << param1 << param2;`
- Call server through stream interface:
`ios.call(server.cap(), Protocol::Foo);`

- Make familiar with source archive
- Try out Fiasco on Qemu
- Build client-server version of "hello":

```
class Hello {  
    void show(char const * str);  
};
```
- Server has to implement `Hello_server::dispatch(...)` function (print the string `<str>`)
- Client has to call the server (`Hello::show(...)` function)
- Try other interfaces
- **Deadline for this Assignment is November 23th**

- November 25th
- Topics:
 - Discussion about the solutions for the first practical assignment
 - Kick-off for next assignment: Implementing malloc()/free() and sessions

Operating System Basics:

- Modern Operating Systems – Andrew S. Tanenbaum (OS Concepts, OS Structures, Virtual Memory)

L4 Specific Documentation:

- BID, Fiasco, Environment (kpr/doc/...)
- Mailing Lists (kpr2014)
- Doxygen comments in source code (HEADERS!)
- There's always a tutor to ask as the last resort.