Complex Lab – Operating Systems

Sessions and Dynamic Memory

Martin Küttler

Last assignment

- Sending multiple messages for large texts is ok.

Last assignment

- Sending multiple messages for large texts is ok.
- If you allocate memory, remember to deallocate the memory and capabilities.

Last assignment

- Sending multiple messages for large texts is ok.
- If you allocate memory, remember to deallocate the memory and capabilities.
- You should update your Control file (libc_be_mem, stdlib, ...)

Last assignment

- Sending multiple messages for large texts is ok.
- If you allocate memory, remember to deallocate the memory and capabilities.
- You should update your Control file (libc_be_mem, stdlib, ...)
- Please report problems (errors/missing informations in slides, missing/bad documentation) to me.

Last assignment

- Sending multiple messages for large texts is ok.
- If you allocate memory, remember to deallocate the memory and capabilities.
- You should update your Control file (libc_be_mem, stdlib, ...)
- Please report problems (errors/missing informations in slides, missing/bad documentation) to me.
- Any questions?

We are here

Paddle Client 1

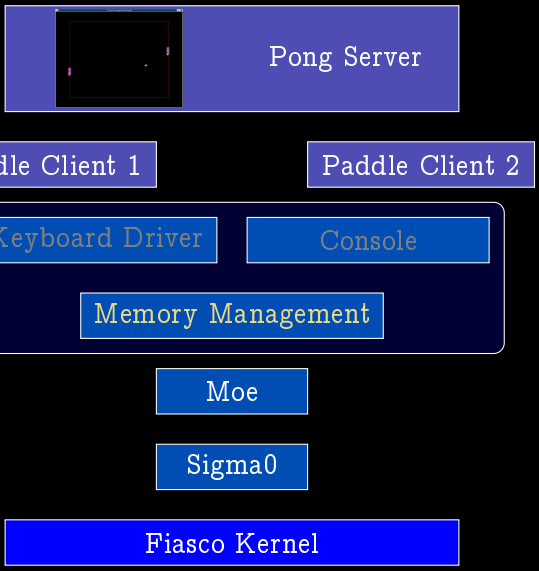
Paddle Client 2

Moe

Sigma0

Fiasco Kernel

Today's goal


Pong Server
Paddle Client 1
Paddle Client 2
Keyboard Driver
Console
Memory Management
Moe
Sigma0
Fiasco Kernel

Sessions

- Scenario:
  - Multiple clients per server
  - Server stores per-client data, needs to distinguish between clients

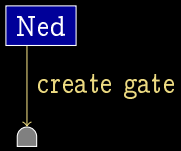
Sessions

- Scenario:
  - Multiple clients per server
  - Server stores per-client data, needs to distinguish between clients
- Poor man's solution:
  - Assign dynamic ID, which clients sends with each call
  - Problem: IDs can be faked

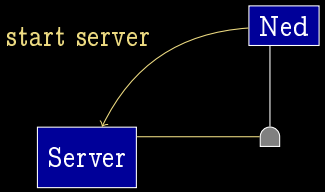
Sessions

- Scenario:
  - Multiple clients per server
  - Server stores per-client data, needs to distinguish between clients
- Poor man's solution:
  - Assign dynamic ID, which clients sends with each call
  - Problem: IDs can be faked
- Better (actual) solution: Sessions
  - One IPC gate per client
  - Clients can be distinguished by the gate label
  - Preferably clients should not even know about sessions

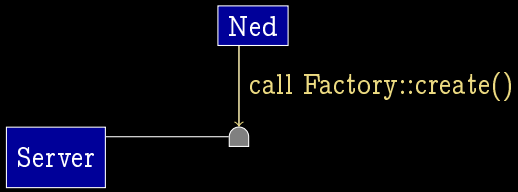
Sessions in L4Re



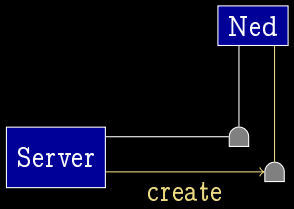
Sessions in L4Re



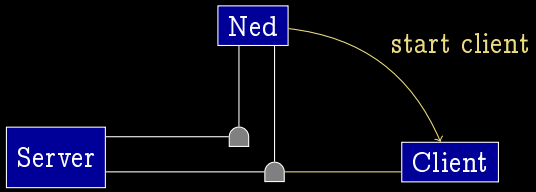
Sessions in L4Re



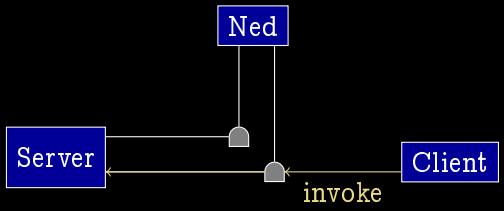
Sessions in L4Re



Sessions in L4Re



Sessions in L4Re



Lua Example: Simple

```
local L4 = require("L4");

local ld = L4.default_loader;
local log = ld:new_channel();

ld:start({ caps = { log_server = log:svr() },
           log = { "server", "blue" } },
         "rom/logging");

ld:start({ caps = { log_server = log },
           log = { "client", "green" } },
         "rom/logging_client");
```

Lua Example: Sessions

```
local L4 = require("L4");

local ld = L4.default_loader;
local log = ld:new_channel();

ld:start({ caps = { log_server = log:svr() },
           log = { "server", "blue" } },
         "rom/logging");

ld:start({ caps = { log_server = log:create(0, "args") },
           log = { "client", "green" } },
         "rom/logging_client");
```

Sessions Implementation

- Clients don't change at all (that's what we wanted, remember?)
- Servers need to handle the create call.

Sessions Implementation

- Clients don't change at all (that's what we wanted, remember?)
- Servers need to handle the create call.
- Before we look at that, ...

Sessions Implementation

- Clients don't change at all (that's what we wanted, remember?)
- Servers need to handle the create call.
- Before we look at that, ...

A short tour of the L4Re IPC server framework

A short tour of the L4Re IPC server framework

- L4::Server implements the basic server loop:

```
void loop() {
  while (1) {
    m = recv_message();
    ret = dispatch(m, utcb);
    reply(m, ret);
  }
}
```

A short tour of the L4Re IPC server framework

- L4::Server implements the basic server loop:

```
void loop() {
  while (1) {
    m = recv_message();
    ret = dispatch(m, utcb);
    reply(m, ret);
  }
}
```

- For each IPC gate there is a L4::Epiface, which
  - keeps the capability to the IPC gate,
  - handles messages from this gate (implements dispatch())

A short tour of the L4Re IPC server framework

- L4::Server implements the basic server loop:

```
void loop() {
  while (1) {
    m = recv_message();
    ret = dispatch(m, utcb);
    reply(m, ret);
  }
}
```

- For each IPC gate there is a L4::Epiface, which
  - keeps the capability to the IPC gate,
  - handles messages from this gate (implements `dispatch()`)
- How does the server know which Epiface it should call?

IPC tour: Epiface registry

- L4::Epifaces are stored in a per-server registry.
- The registry can find Epifaces by an ID (label of IPC gate)
- L4::Basic_registry: ID is pointer to object
- L4Re::Util::Object_registry provides a convenient interface:

```
L4::Cap<void> register_obj(L4::Epiface *o,
                           char const *service);
L4::Cap<void> register_obj(L4::Epiface *o);

bool unregister_obj(L4::Epiface *o);
```

IPC tour: Registry server

L4Re::Util::Registry_server is a L4::Server that maintaines a L4Re::Util::Object_registry

```
static L4Re::Util::Registry_server<> server;

class MyServer : public L4::Epiface_t<MyServer, MyInterface>
{ ... };

// When you need a new session object
server.registry()->register_obj(new MyServer());
```

Session Implementation – Factory Server

```
class SessionServer : L4::Epiface_t<SessionServer, L4::Factory>
{
public:
    int op_create(L4::Factory::Rights, L4::Ipc::Cap<void>& res,
                  l4_mword_t type, L4::Ipc::Varg_list<> args) {
        if (type != 0) return -L4_ENODEV;

        L4::Ipc::Varg tag = args.next();
        if (!tag.is_of<char const *>()) return -L4_EINVAL;

        auto helloserver = new HelloServer
                                  (tag.value<char const *>());
        server.registry()->register_obj(helloserver);
        res = L4::Ipc::make_cap_rw(helloserver->obj_cap());
        return L4_EOK;
    }
};
```

Sessions

- With that you can add support for multiple clients in the hello server.

Sessions

- With that you can add support for multiple clients in the hello server.
- Assignment 1.5:
  - Make your hello server a logging server that supports multiple clients
  - Client messages should be prefixed with an id string, that is passed to the server in the create call.

Sessions

- With that you can add support for multiple clients in the hello server.
- Assignment 1.5:
  - Make your hello server a logging server that supports multiple clients
  - Client messages should be prefixed with an id string, that is passed to the server in the create call.
- Problem: Now you need dynamic memory, but `malloc` and `free` are missing.

Memory Allocation

- Memory allocation is (currently not) implemented in a backend of L4Re's C library (in src/l4/pkg/l4re-core/libc_backends/)
- You can get new pages from Moe:
  - Allocate a dataspace capability
  - Get a dataspace from Moe:
    `L4Re::Env::env()->mem_alloc()->alloc(size, ds);`
  - Attach dataspace to local address space:
    `L4Re::Env::env()->rm()->attach(&addr, size, flags, ds);`
- To free unused pages:

```
L4Re::Env::env()->rm()->detach(addr, nullptr);
L4Re::Env::env()->mem_alloc()->free(ds);
```

Incorrect malloc()

```
void *malloc(unsigned size) {
    L4::Cap<L4Re::Dataspace> ds
        = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();

    if (!ds.is_valid()) return 0;

    long err = L4Re::Env::env()->mem_alloc()->alloc(size, ds);
    if (err) return 0;

    void *addr = 0;
    err = L4Re::Env::env()->rm()->attach(&addr, size,
        L4Re::Rm::Search_addr, ds);
    if (err) return 0;

    return addr;
}
```

Memory Management – Lists

- Idea:
  - Keep list of (address, size) pairs
  - In `malloc`, search for an appropreate entry
- Problem: You'd need dynamic memory for that list.
- Typical Solution: Inlining
  - Put size and next-pointer directly into your memory
  - Do not hand out the memory where size is stored – it's needed for free.
  - That's what most libC-implementations do.

Memory Management – bitmaps

- Manage memory as pool of fixed-sized chunks.
- Use bitmap to store available chunks.

Memory Management – problems

- You will need some initial memory. You can use L4Re's memory allocator for that.
- As soon as you have multiple threads (you will), you need proper locking.
- There are more options for the implementation. Come up with something yourself, or have a look in some book / the internet.

Assignment 2

- Implement a session-capable hello server (that's going to be our logging server)
- For that you'll need to implement `malloc`, `free` and `realloc`.
- From there on, you should be able to use C++'s STL.