IPC - Setup

Option 1: **Do It Yourself**

$ > wget tudos.org/~benjamin/ipc.tar

- unpack to directory of your choice, copy pkg/ipc in place and rebuild
- update menu.lst, add required links to new binaries and moe scripts
- build and run iso image

Option 2: **Out Of The Box**

$ > wget tudos.org/~benjamin/Makefile

$ > make setup && make

IPC Building Blocks

Server

- Allocate Capability
- Create Gate
- Register name
- Server loop: handle messages

Client

- Allocate Capability
- Query name
- Send message and wait for reply

Server and Client: Caps

- L4::Cap<type>
  - Generic capability
  - e.g. void, L4::Kobject, L4::Dataspace
- L4Re::Util::cap_alloc.alloc<L4::Kobject>()
  - Allocates new capability

Server: Create gate & register service

- L4Re::Env::env()->factory()->
  create_gate(cap,thread,label)
  - Cap: capability to this new gate
  - Thread: the thread to be bound to this gate
  - Label: An inbound message gets a label when passing a gate; used to distinguish gates

- L4Re::Env::env()->names()->
  register_obj(string,cap)
  - String: the name to be registered
  - Cap: capability to be associated with this name

Client: Query service

- L4Re::Env::env()->names->query(string,cap)
  - Search in the name space for a string
  - returns capability to that object or an error
  - e.g. query(“my_server”,cap)

Server and Client: Communicate!

- Server:
  l4_ipc_wait(UTCB,label,timeout)
  - UTCB: l4_utcb(), User Thread Control Block, used as transfer buffer
  - Label: pointer to unsigned long
  - Timeout: how long to wait, e.g. L4_IPC_NEVER

- Client:
  l4_ipc_send(cap,UTCB,message_tag,timeout)
  - Cap: capability to peer (a gate)
  - Tag: details of the IPC operation, e.g. number of words/pages/... to send, here: l4_msgtag(0,0,0,0)

Data transfer

- Sender: copy data into own UTCB
- Send message with message tag (0,n,0,0) to transfer n words
- Receiver: copy data from own UTCB

- l4_utcb() returns own UTCB
- l4_utcb_mr_u(utcb)->mr[0..n]: message regs

- L4Re::Util::cap_alloc.alloc<type>()

- L4Re::Env::env()->names->query(string,cap)
- l4_ipc_send(cap,UTCB,message_tag,timeout)

- L4Re::Env::env()->factory()->
  create_gate(cap,thread,label)
- L4Re::Env::env()->names()->
  register_obj(string,cap)
- l4_ipc_wait(UTCB,label,timeout)

- l4_utcb_mr_u(l4_utcb())->mr[0..n]: message regs

Higher level: Streams

- Known from C++: operator << and >>
- Works here with IPC streams:
  1. Get cap ✔
  2. Query service ✔
  3. Create a stream backed by the UTCB
- Client:
  4. Call server
  5. Get result out of the stream
- Server:
  4. Derive from predefined class, overwrite loop
  5. Get message from stream
  6. Put answer back into the stream

Null_server

```
Class Null_server : public L4::Server_object
{
    public:
        int dispatch (l4_umword_t, L4::Ipc_iostream &ios)
        {
            unsigned  i;
            ios >> i;
            ios << i;
            return L4_EOK;
        }
}
```

Null_client

- See before: Alloc cap, query service name

```
L4::Ipc_iostream s(l4_utcb());

s << 23;

l4_msgtag_t res = s.call (server.cap());
if (l4_ipc_error (res,l4_utcb())  ... // handle error

l4_uint32_t response;

s >> response;
```

Sending and receiving strings

- Send

```
s << “Hello”;
```

- Receive

```
char* str;
unsigned long l;
s >> L4::ipc_buf_in (str,l);
```