Towards capabilities in HelenOS

The elephant in the room

http://www.helenos.org

Jakub Jermář
jakub@jermar.eu

FOSDEM '18
Brussels  3 & 4 February 2018

www.fosdem.org »

HelenOS
HelenOS in a nutshell

- Portable microkernel-based multiserver OS
- Designed and implemented from scratch
- Not a clone of any existing OS / API
- Virtually no third-party code
- Fine-grained userspace components
Since last FOSDEM...

- HelenOS 0.7.0 (April)
- HelenOS 0.7.1 (November)
- HelenOS Camp 2017
- Fork us on GitHub!
- CZ.NIC feeds one HelenOS developer
Coming soon

- C++14 support
- USB 3.0 support
Google Summer of Code

- Microkernel devroom
- Organizations Announced on February 12
- Student Application Period starts on March 12
- [http://gsoc.microkernel.info/](http://gsoc.microkernel.info/)
Terminology

Capability:
Task-local name for a reference to a kernel object; userspace uses integer handles to refer to capabilities

Kernel objects:
Reference-counted wrappers for a select group of objects allocated in and by the kernel that can be made accessible to userspace in a controlled way via capability handles
Motivation

- Fix broken mechanisms
- Reduce number of mechanisms
- Fix broken interfaces
- Get rid of global names
- Modernize the system
Example: passing files

• How to pass an open file which exists in the VFS server from the parent task to the child?
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?

**IPC_M_CHANGE_AUTHORIZE**

- Two clients of a server can negotiate a change of their state kept in the server
- All VFS files map to single kernel object
- Actively involves the server + kernel notification
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?

1. IPC_M_CONNECT_TO_ME(port = 6)
2. EOK
Example: passing files

- How to pass an open file which exists in the VFS server from the parent task to the child?

**IPC_M_CONNECT_(TO_ME/ME_TO)**

- Mechanism to create (callback) IPC connections
- Does not currently accept a *port number*
- VFS not involved
- One kernel object per one VFS file
Example: IRQ handlers

- Device drivers can register an IRQ handler. How to identify the handler so that it can be unregistered?
Example: IRQ handlers

- Device drivers can register an IRQ handler. How to identify the handler so that it can be unregistered?

Before HelenOS 0.7.1

```c
    g_devno = SYS_DEVICE_ASSIGN_DEVNO();
    SYS_IPC_IRQ_SUBSCRIBE(irq, g_devno, ...);
    SYS_IPC_IRQ_UNSUBSCRIBE(irq, g_devno);
```
Example: IRQ handlers

• Device drivers can register an IRQ handler. How to identify the handler so that it can be unregistered?

Before HelenOS 0.7.1

```c
int g_devno = SYSDEVICE_ASSIGN_DEVNO();
SYS_IPC_IRQ_SUBSCRIBE(irq, g_devno, ...);
SYS_IPC_IRQ_UNSUBSCRIBE(irq, g_devno);
```

- A microkernel should not assign devno's
- No enforcement to use the devno for registration
- Everyone can unregister any IRQ handler
Example: IRQ handlers

- Device drivers can register an IRQ handler. How to identify the handler so that it can be unregistered?
Example: IRQ handlers

- Device drivers can register an IRQ handler. How to identify the handler so that it can be unregistered?

Since HelenOS 0.7.1

```
handle = SYS_IPC_IRQ_SUBSCRIBE(irq, ...)
SYS_IPC_IRQ_UNSUBSCRIBE(handle)
```

- Capability handles are task-local
- Need to posses the capability in order to unregister
Elephant in the room

- Capabilities are great
- How to introduce them to HelenOS?
Elephant in the room

- Capabilities are great
- How to introduce them to HelenOS?
- We don’t have to (start from scratch)
- HelenOS already has them (in a limited way)
Basic HelenOS RPC

```
Task
```

```
Answerbox
```

```
Phones
```

```
ipc_call_async_fast(2, method, arg1, arg2, arg3, ...)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

```
ipc_wait_for_call_timeout
```
Call forwarding

**Task**

```
ipc_wait_for_call_timeout
```

**Answerbox**

```
ipc_call_async_fast(2, method, arg1, arg2, arg3, ...)  
ipc_wait_for_call_timeout
```

**Phones**

```
ipc_answer_fast(callid, retval, arg1, arg2, arg3, arg4)
```

**Call forwarding**

Task

Answerbox

Phones

Answerbox

Phones

Answerbox

Phones

Answerbox
Creating a new connection

Task

Answerbox

Phones

Task

Answerbox

Phones

Answerbox

Phones

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task

Answerbox

Task
Creating a new connection

Task
- `ipc_call_async_fast(2, IPC_M_CONNECT_TO_ME, arg1, arg2, arg3, ...)`
- `ipc_wait_for_call_timeout`

Answerbox
- `ipc_answer_fast(callid, EOK, arg1, arg2, arg3, arg4)`

Phones
- `ipc_forward_fast(callid, IPC_M_CONNECT_TO_ME, arg1, arg2, arg3, arg4)`
- `ipc_wait_for_call_timeout`
Elephant in the room

- HelenOS IPC: a coarse-grained capability system
- Roughly analogous to Mach IPC
### Elephant in the room

<table>
<thead>
<tr>
<th></th>
<th>Mach</th>
<th>HelenOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit of IPC communication</td>
<td>Message</td>
<td>Call</td>
</tr>
<tr>
<td>IPC communication endpoint</td>
<td>Port</td>
<td>Answerbox</td>
</tr>
<tr>
<td>IPC connection</td>
<td>Send Right</td>
<td>Phone</td>
</tr>
<tr>
<td>Receive from IPC endpoint</td>
<td>Receive Right</td>
<td></td>
</tr>
<tr>
<td>Right to answer</td>
<td>Sender includes send-once right to a reply port</td>
<td>Implicit via received call</td>
</tr>
<tr>
<td>Give connection to third-party</td>
<td>Reply with send right</td>
<td>Forward IPC_M_CONNECT_ME_TO call</td>
</tr>
</tbody>
</table>
Elephant in the room

Capability ↔ Phone
Pass capability ↔ Forward connection
Limits of IPC as capabilities

- Phone the only type of capability
- Can only pass one type of capability
- Only one answerbox per task
- Maximum 64 phones per task
- Kernel addresses as user callid’s
Work already done

- New capability framework
- IPC switched to using it
- Kernel objects: phones, calls and IRQs
- Arbitrary number of capabilities
- User refers to calls using capability handles

```c
kobject_t
.type = KOBJECT_TYPE_CALL
.refcnt = 3
```

```c
cap_t
.state = CAP_STATE_PUBLISHED
.task = <task A>
.handle = 5
```

```c
cap_t
.state = CAP_STATE_PUBLISHED
.task = <task A>
.handle = 13
```
What needs to be done

- Answerbox as capabilities / kernel objects
  - Arbitrary number of answerboxes
  - Pair with user-level async ports
  - Where does the answer go to?
- Tasks and threads as kernel objects
  - Get rid of the remaining global IDs
  - Get rid of existence checks
Future

- Resource management
  - We removed some arbitrary resource limits
    - 64 phones per task, 4 active calls per phone
  - Per-task resource pools to compensate
  - Service-for-resource trading

- Capability rights

- Revoking of capabilities

- Pass arbitrary capability between tasks
Summary

- Originally HelenOS had capabilities for IPC endpoints by accident
- Generalized the IPC subsystem to support other kernel objects
- Capabilities used to fix broken APIs
- Still need to introduce more kernel objects, especially for answerboxes
- Things need to settle down
Q&A

http://www.helenos.org
https://github.com/HelenOS/helenos
@HelenOSOrg
@jjermar

Thank you!
References

- Photo of laptop running HelenOS with USB 3.0 support courtesy of Ondřej Hlavatý