Review of Lampson's Hints Sec. 2

POCS'2020 Recitation
Mark Sutherland

Roadmap for Today

1. Exercise in Writing a Specification [40 min]
2. Critiquing our Specs [15 min]
3. Break [5m]
4. Modules and Layering [20 min]
5. Classes and Objects [20 min]

2.1 - Have a Specification

Key points of Lampson’s idea of a specification:

- Specifies what, not how
- Decouples the clients of a system, from the implementation of a system
- The code itself is a refinement of the spec

2.1 - Have a Specification

What comprises the spec:

- Externally-visible state of the system
- Actions taken by clients, their effects on the state

Difference between the state and an interface, in the example of an FS?

- State includes files, and directories
- An interface can include read/write/open/delete

Exercise in Specification

In groups, sketch a spec for Lampson’s Global Name Service

- Start with the client interface, do not consider the administrators
- Do the state first, then the actions: enumerate, get/set

Key things to remember:

- Be as clean and minimalistic as you can
- Think about **what**, not **how**

Modules, Components, and Layers in RON

Executive summary:

- Design of distributed IP routing means that fault recovery is slow
- Smaller number of nodes can proactively monitor paths and recover
- Application-level overlay network where nodes route packets co-operatively

Modules, Components, and Layers in RON

Lampson says: “A really successful interface is like an hourglass: the spec is the narrow neck, with many clients above and many codes below; it can live for decades.”

Does RON’s design contradict what he says here?

- RON graciously preserves the beautiful interface of IP, as all its traffic still passes through those layers which are blissfully unaware

Modules, Components, and Layers in RON

What techniques does RON itself use for modularity?

- Separating forwarding from routing
- The routers themselves are constructed with many different modules

How is RON an “Open system”?

- Routing policies can be arbitrarily powerful, using BPF-like matching
- Reflect - do such procedure arguments complicate the spec?

The Cost of Layers

Lampson says: “Layers are good for decoupling, but they are not free. Unless you’re very careful, there’s a significant cost for each level of abstraction. Usually this cost is worth paying.... There are two ways to reduce it: make it cheaper to go from one layer to another, or bypass some layers (making the system a lot more complicated and hard to maintain).”

Where have we seen this before?

- Exokernel arguments about performance and raw device access
- P&G paper: virtualization through emulation is costly
- Xen: the hardware-MMU layer

Reducing the Cost

What solutions are proposed to those problems?

- Exokernel: reduce cost of abstractions by implementing them in user mode, redesign kernel primitives
- P&G virtualization criteria: direct execution
- Xen memory de-virtualization: only involve myself on the path of updates

Classes and Objects Recap

Summarize the interaction between programs and specs:

- Spec and code are attached to **data** (an object)
- The classpec is providing abstraction, the instance provides the refinement
- Overloading breaks the classpec

Classes and Objects

In which paper(s) have we seen the idea of a classpec?

- “On the Duality of OS Structures”
- Compares a message-oriented and procedure-oriented system

What is the common module that both system models needed?

- Resource manager

Classes and Objects

Abstract state required for the resource manager classpec:

- Input and output ports
- Resource name
- *Should there be more here?*

Actions:

- Enter & request resource
- Exit & release

Implementation and Refinement

What do we use for input/output in message-oriented systems?

- Explicit messages (i.e., RPCs that have a payload)

How about for granting and revoking resources?

- Implicit: queues serve as the refinement of requests for the resource
- When a reply is generated, the resource is by definition free

Implementation and Refinement - II

What do we use for input/output in procedure-oriented systems?

- Function calls, most probably exposed through a loader

How about for granting and revoking resources?

- Shared-memory synchronization
- Requesting a resource explicitly “waits”, freeing it does “wake up”