HYDRA: The Kernel of a Multiprocessor Operating System  
W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack  
Communications of the ACM 17(6), June 1974, pp. 337-344.

1. Hydra is an example of a kernel or nucleus of an OS. As such, what were its goals?

Background: Name - many-headed dragon - Greek  
"many-headed" OS for multiprocessors  
**Prove** PDP-11s  
This paper: philosophy of system - PROTECTION

Many papers on Hydra - more details on capabilities

Little eval

- multi-processor env.
- separation of mechanism + policy
- structured programming / modularization
- reject strict hierarchy (THE) (flex)
- PROTECTION - apply uniform way throughout
- Reliability
2. What is the role of HYDRA?

- support abstracted notion of resource (objects)
- push idea that "everything is an object" to extreme
- good for research
3. What does each object in HYDRA contain?

- Procedure, LUs, Process, Files, SEUs, Dirs
  - TYPE - classify, identify valid ops
  - Unique name - id instance
  - Data - Value
  - Capability list - give access to other objects
4. What other options exist instead of protection instead of capabilities? Where are other approaches popular?

<table>
<thead>
<tr>
<th></th>
<th>File A</th>
<th>File B</th>
<th>Mem A</th>
<th>Mem B</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>R</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ACLs**
- Popular in file system
- Store permissions in each inode
5. What does a capability contain? HYDRA capabilities contain generic (kernel) and type-specific (auxiliary) rights. Generic rights include rights such as copying an object, getting, putting data of an object; manipulating capabilities (read, deleting, adding). Why are type-specific rights needed?

1) reference to an object (instance name) → knowing name is not sufficient

2) access rights (bit vector) – fixed length

1011000 → can or can't do

-split into generic + type-specific

-Use to implement policy for new type (bibliography)
-Add, print, change

-Separate policy + mechanism (Hydra does not know policy)
6. One of the most important properties of capabilities is that they can't be forged. How can this be ensured? Can you think of where capabilities are used in UNIX?

- HW or OS support

1) Tag: Every word of mem is tagged (user can't set)
   1: cap
   0: other
   User processes can't write cap words

2) Segmentation:
   - Use mem protect mechanisms to partition caps from data
   - Users don't have perm to caps

UNIX? hint: opaque handle
users can't change internals

File descriptor
- handle given to user process
- can't change internal cap rights
- no use copying to someone else
7. What is in an object of Type Procedure (i.e., Name, Data, and Capability list)? What is a caller-independent capability? What is a caller-dependent capability and how is it implemented?

Name: Foo
Data: code + static data
Cap list: caller-independent caps
local vars

dependent: caps passed in

Templates
- cap type, needed rights
  FILE 0010
  foo 1010 ✓
  bar 1100 ×

No interpretation of rights by kernel
Procedure developer defines
Separate policy from mechanism
8. What happens when a procedure is called? What is problematic about this and what is the solution?

CALL proc_object, ret slot for cap,
cap, w/ mask1, cap2 w/ mask2

1) jmp to kernel (must have right to call this procedure!)
2) check parameter TYPE + rights
3) create new LIVS (push on stack)
4) construct new capabilities (mask + caller-independent)
5) jmp into code

- Jump to kernel is costly

Opt: don't do this for all procedures (just boundary ones)
9. Why is rights amplification useful? Can you think of an example of where this is used in UNIX? Why can masking away some rights be useful?

- may need additional rights to perform useful work
  - e.g. type dependent print
    - need to read
  - only exists during procedure

UNIX?
- setuid - get rights of root (or another user) for well-defined process
  (Must have amplification right-only defining procedures have it)

Mask?
- do not trust procedure you are calling
  - e.g. proc only needs to read
    1st mask away write rights
10. What is an LNS (Local Name Space)? Why is it not the same as the Procedure Object? How does a Procedure access capabilities that are not local? Is this a correctness or a convenience issue?

- Record of execution env.
- Active procedure w/ instantiated caps
- Must be dynamic for recursion

- LNS
  current procedure
  caps

- Use "Walk" right to reach cap list of all objects

- Convenience so don't have to pass everything in
11. What is in an object of a Process Type? What types of operations on a Process should the scheduler have rights to? What operations should the scheduler NOT have rights to?

Name: pid
data: state, priority
clist: stack of UNSel

Good ops: Start + Stop

Not: Read or modify data or caps
- don't know what process is doing
12. What are the strengths of Hydra capabilities? What are their problems?

- Elegant
- Separates mechanism (protection) from policy (security)
- Complete flexibility + Power
- Uniform mechanism
- Non-hierarchical

Problems:
- Complex, hard to use
  - Hydra never completed, no real users
- Overhead of kernel calls on procedure calls to check caps
- Space for caps