Hydra

Robert Grimm
New York University
Say Hello to Hydra

A gigantic monster with 7 (or 9) heads from Greek mythology
Say Hello to Hydra
(Some 3000 Years Later...)  

- Kernel for the C.mmp  
  - 16 PDP-11s, 32 MByte RAM, central clock, relocation hardware  
- The three questions (once for each paper)  
  - What is the problem?  
  - What is new or different?  
  - What are the contributions and limitations?
Design considerations

- Support a multiprocessor environment (duh)
- Separate mechanism from policy
- Integrate design with implementation methodology
- Reject strict hierarchical layering
- Provide pervasive protection
  - Including a *single* reference monitor
- Make the system reliable
Implementation Methodology and Protection

- “Abstracted notion of a resource”
  - Instances of resources are *objects*
  - Objects have *types*
  - Applications perform operations on resources through *procedures*

- Protection
  - Controlled access to resources within execution domains
  - Controlled passing of control and resources
  - Expressed and enforced through *capabilities*
    - Cannot be forged by applications
Let’s Make This a Little More Concrete...

- **Procedure**
  - Code (sequence of instructions)
  - Data: List of capabilities
    - Caller-independent and “holes” for caller-dependent ones

- **Local name space (LNS)**
  - Record of a procedure’s execution environment
    - Combines caller-independent with -dependent capabilities

- **Process**
  - From outside: Unit of scheduling
  - From inside: Stack of LNS’s
    - Representing “cumulative state of a single […] task”
The Gory Details

Objects

- Implemented as tuples
  - Unique name: 64 bit number
  - Type: unique name of the *class* object
    - The type of a class object, in turn, is the special object “TYPE”

- Representation
  - Capabilities: Only accessible through kernel
  - Data: Not interpreted by kernel

- Reference-counted
The Gory Details
Capabilities

- Also implemented as tuples
  - Reference to an object
  - Set of access rights
    - Global: Kernel rights
    - Type-dependent: Auxiliary rights
      - Enables single reference monitor! But?

- Each access right corresponds to an operation, and (typically) procedure

- Putting objects and capabilities together
  - LNS: An object whose capabilities specify accessible objects
    - Also, a mapping from small integers to unique objects
The Gory Details
Invoking Procedures

- **CALL** to invoke a procedure
  - Goal: Create a new LNS based on procedure’s capabilities
  - Argument checking based on *templates*
    - Required type
    - Required access rights
    - New access rights: *protection domain transfers* (incl. *amplification*)
- **RETURN** to (shockingly) return from a procedure
  - Removes top LNS
    - Nothing said about checking rights on returned values…
Let’s Switch Gears
Policy/Mechanism Separation

- **Goal:** “Enable the construction of operating system facilities as normal user programs”

- **Assumptions**
  - User-level programs are buggy or even malicious
    - Prevent direct access to hardware
    - Assure fairness between competing applications
  - User-level programs run in their own protection domains
    - Make policy decisions quickly

- **Engineering trade-off:** Parameterized policies
  - Fast short-term decisions with long-term application control
Scheduling

- The basic policy/mechanism separation
  - Short-term: Scheduled by kernel
  - Long-term: Scheduled by *policy module* (PM)

- The operational view
  - PM sets policy and *starts* a process
    - Policy stored in process context block (PCB)
      - Priority, processor mask, time quantum
      - Maximum current pageset (for *paging*!)
  - Kernel brings process in core and schedules it
  - Kernel stops process and notifies PM
    - Through a policy object, which serves as a mailbox
When Is a Process Stopped?

- Its time quantum is exhausted
  - Time slice duration, number of slices
- It blocks on a semaphore
- It returns from its base LNS
- It exceeds its maximum CPS size (see paging)
How to Schedule Fairly?

- Basic idea: Provide “rate guarantee[s]”
  - But not (yet) implemented (!)

- Goals
  - Each PM receives guaranteed percentage
  - If CPU is underutilized, the excess is shared among other PMs
  - If PM does not get its guarantee, it is given more later
  - Priority only distinguishes processes controlled by same PM
  - Processes at same priority level scheduled round robin
Paging

- **Hardware**
  - 16 bit addresses (64 KByte)
  - Eight 8 KByte page frames
  - No demand paging

- **Three-level hierarchy**
  - LNS
    - Accessible resources
  - Current page set (CPS)—changed through CPSLOAD
    - In-core resources
  - Relocation page set (RPS)—changed through RRLOAD
    - In-core and addressable resources
The Finer Points of Paging

- Page only needs to be in-core when it is in RPS
  - Initiate I/O on CPSLOAD but do not block
- Only CPS for top-level LNS needs to be in-core
- Procedures are bootstrapped through explicit CPS and RPS specifications
- Scheduling and paging interact
  - Only in-core processes can be scheduled!
Paging Policy

- Policy/mechanism separation
  - Kernel performs paging and page replacement
  - PM only determines which process to run and its max CPS

- Process paging
  - On start, top CPS brought into core
  - On stop, top CPS becomes eligible for eviction
  - On call/return, CPS automatically changed
Paging Policy (cont.)

- Paging guarantees
  - Sum of all max CPS sizes ≤ available page frames
- Page replacement
  - Performed by kernel, giving priority to top-level pages of runnable processes
Paging Policy (cont.)

- **Issues**
  - Not enough information visible to PMs
    - Which pages are in-core
    - Which pages are shared
  - Not enough information visible to kernel
    - Which pages are going to be used real soon now
    - Which pages are more important than others
  - Too strong a guarantee
    - Pages may be shared ➔ Underutilization of existing memory
Protection

- Access protection enables clear separation of policy and mechanism
  - No need for parameterized policies or active PMs
- But what about protecting a process from its PM?
  - Each process can ask for descriptive info on its PM
  - Kernel notifies process if it has been
    - Started before its semaphore has unblocked
    - Scheduled on the wrong processor
    - Started after exceeding its max CPS (without a change in that number)
Some (Nasty) Questions

- Did Hydra ever work?
  - “[W]e are more concerned with philosophy than with implementation”
- How is protection enforced?
- How is arbitrary rights amplification prevented?
  - Remember: Access to capabilities is mediated by kernel
    - Special right to add amplification templates
    - “Protection in the Hydra Operating System” [SOSP75]
- What is missing from the “abstracted notion of a resource”? 
More (Nasty) Questions

- What are the short-comings of capabilities?
  - Compared with, say, Unix or Multics
- Are parameterized policies good enough?
Discussion