Reflections on an Operating System Design

Butler W. Lampson and Howard E. Sturgis

Presented by Neal H. Walfield

Cal System

- General purpose OS
  - 200 users
- Classes of Applications
  - Editing
  - “Typical Fortran batch jobs”
  - Large batch jobs
- Legacy support

Structure

- Capabilities
- Objects
- Domains
- Layers
  - Abstract machine / New architecture / Virtual Machine
  - Unprivileged
  - No reliance on later layers
- Explicit accounting

Isolation

- Domains
  - Protection from others
  - Confined
- Controlled breaching via messaging

First Protection Layer

- Microkernel
- 8 objects
- No reliance on disk

Kernel Objects

- Kernel files - Mach Memory Object
- Event channels - Inter-process signalling (fixed size queue)
- Allocation blocks - Memory and CPU quota
- C-lists
- **Capabilities**
- Labels - Names a domain
- **Processes** - Hierarchy of domains
- **Operations** - Authority to invoke a domain

Capabilities

- Name objects
- Data: `<type`, `rights`, `value>`
- `value`: object pointer or word
- As object pointer: `<unique name`, `index>`
  - Indexes Master Object Table (MOT)
  - Name stored in MOT entry
  - O(1) revoke
  - O(1) relocation

Processes

- Virtual machine
- Contain tree of domains
- Call stack - no reply capability

Operations

- Realize user-objects
- Sealed closures
  - Authority to transfer control to another domain

Extensibility

- Invalid operations return `abnormally`
- Kernel chains to next level in `operation`
- Cost of abstraction is zero
- Not for overriding functionality

Disk Files

- Extend kernel files to support paging
- Invocation only goes to disk file when kernel file returns abnormally

Directories

- Symbolic name to *user* capability
- Access control lists
- Directory is trusted by user?

Accountability

- Reduction in sharing
- Difficult to attribute, e.g., automatic
- Lots of unnecessary paging

Object Paging

- Kernel objects not paged:
  - No reliance on disk (transparent paging)
  - Data integrity[1] (user pagers)
- Kernel resources *are* sparse

[1] User-level checkpointing through exportable kernel state: Tullmann, et al., 1996

Duplicity

- Process $\approx$ Domains
- Event Channels $\approx$ Operations

- Motivated by performance concerns
- Unnecessary

Negative Results

- 2–3 iterations for new ideas to be implemented efficiently
- Don’t ignore design flaws
- An OS is more than a kernel

Positive Results

- Layering
  - Simplification
  - Reliability
- Capabilities
  - Consistent and uniform naming
  - Consistent and uniform access control
- Devices as processes

My Observations

- Little focus on security
- Access control does not rely on delegation
- System not persistent

Questions

- Domain Labels: identify a service in any process?
- How do types work?