Scribe Notes 9/4
Steve Hanna
CS261

---

1 Protection in Multiuser Operating System

Imagine the following example: Alice and Bob wish to use a computer that is a shared resource between them. Only one user can be on each machine at a time.

1.1 Security Risks?

Informal Terminology

- Bob reads A's information
- Keyloggers
- Delete other users' files
- Resource Starvation

Formal Terminology

- Confidentiality
- Integrity
- Availability
- Isolation

1.2 How might we solve this problem?

1. Each user has his own computer – this is the standard model
2. Imagine a physically secure room, encased in concrete, embedded in the side of a mountain that has a steel door and the machine operator is the only person with access. Operator runs the users' code on the machine in this room, returns the results to the user who provided the job and then resets the machine to the pre-execution state – this is the standard model, elaborated; a somewhat gold standard.

   Assumptions for item 2: The operator is not malicious and is competent, the machine's state is completely reset between jobs, there exists mutual exclusion, the conduit for receiving data and returning data to the user is secure. We also assume physical security and that the computer is not networked.

2 Virtual Memory Review

2.1 Architecture Setup

Recall how virtual memory is setup architecturally: the CPU is connected to the MMU and the MMU is connected to the RAM. The CPU supplies a virtual address to the MMU and the MMU provides the physical address mapping so that the data can be retrieved in the RAM.

2.2 Some CPU State

1. Mode Bit, Are we running in supervisor mode? (comparable to the x86 ring model, ring 0 is supervisor, ring 3 user mode, ring 1 and ring 2 unused in most operating systems)
2. Interrupt Handler Address
3. Registers

2.3 Loading a program (Basic, Conceptual)

Assume the supervisor bit is set and the kernel is initiating these tasks (running in ring 0).

1. Setup the page table
2. Read the image from the disk and copy to allocated page
3. Set EIP to image entry point and unset supervisor bit

2.4 Availability

Utilize the CPU timer and set it to some time quantum. When this time expires, the interrupt handler is called and a context switch occurs to the kernel, the kernel decides the next process/thread to schedule, sets up virtual memory accordingly. x86 Intel based architectures use the *CR3* register. For a detailed view of x86 virtual memory see the Intel architecture manuals or the several decent articles linked from *Wikipedia*.

2.5 Problems?

Virtual Memory is a nice example of *Isolation* but it's not perfect; side-channel attacks still exist. **Remember**: Strict isolation is not the goal, controlled sharing is!

3 Controlled Sharing

We want to extend operating system primitives to communicate in some way, we want to have IPC (Interprocess Communication).

3.1 Simple Message Passing Model

Assumptions: each process runs on its own respective machine ("domain"). Domain is defined here as a protected subsystem that has behavior and state (state being the data).

3.2 Implementation?

Example: Unix uses sendmsg, Windows uses SendMessage, SendMessageEx.

We would like to have a policy that uses no copying, that is the transition of data from Process A to Kernel to Process B doesn't involve copying the data to the kernel from Process A and copying the data from the kernel to Process B. How about a memory mapping approach? Just share the page!

Problems with this? Access permissions, race conditions when sending or receiving data, pointer issues, page mapping information leaking (due to page boundaries). Well, we could just use copy on write (Lazy Copy). Still, we have a lot of issues and no good solutions.

Best way to do this? Just copy the data!

4 Access Control Matrices

Here we have a notion of active (subjects) and passive objects.

Example **ACM**:

Left column is *foo*, right column is *bar*, top row is *daw*, bottom row is *bob*.

$$\begin{array}{c} \\ Subjects \end{array} \begin{array}{c} \text{Objects} \\ \begin{pmatrix} RWO & R \\ Empty & RWO \end{pmatrix} \end{array}$$

R - Read

W - Write

O - Owner

4.1 Typical Implementation

Typically this is implemented as a sparse matrix and a list is kept of nonempty cells. Usually an ACM is *huge*, so an efficient implementation is needed.

In operating systems this type of policy is implemented as either an **Access Control List** (ACL) or a capabilities-based system.

ACL: (column data) the permissions are stored in the inode of the object (file). Capabilities-Based: (row data)the user account has list of accessible resources.

4.2 Some Details

So far we've neglected the fact that access control lists are dynamic. Users are added and removed, files created and deleted, rights are added and removed.

When a subject owns an object is had an artificial right that enables the subject to give arbitrary access to objects.

- owners can share permissions/give permissions
- anyone can drop their privilege
- anyone can create new objects

The analysis of whether an arbitrary object can be accessed through a series of permissions manipulation is in general, **undecidable**.