CSE221 Lecture 4

Aronya Baksy

October 2024

1 Protection and the Control of Information Sharing in Multics

1.1 Goals

- Controlled sharing of information: informed every step of the design process
- Check permissions for access rather than exclusion, the default state is deny
- check all accesses to all objects for current authority
- design is not secret - don't assume attacker ignorance
- principle of least privilege, dont give more access than necessary
- A simple human interface to make it easy to apply the protection mechanisms
- Functional Goals: provide centralized admin control as an *option* only, and allow the use of *protected subsystems* to implement custom access control schemes

1.2 Access Control Lists

- Physical storage in MULTICS is organized in terms of *segments*.
- Each segment has an ACL (list of users that can access the segment)
- Each process has a *principal identifier* associated with it, consisting of an individual **user name**, followed by a **project** (user group) and a **compartment**
- ACL entries are sorted left to right, with names sorted before don't cares
- A *compartment* is a type of user group that the user specifies when they authenticate themselves
- In addition to segments, other types of objects like directories, removable media descriptors and message queues are also proposed
- Permissions in terms of r/w/e (some combos are not allowed)
- idea to add a "trap" extension to call an interrupt handler to run a custom procedure before granting access
  - useful for, say, permitting access only between 9am and 5pm.
  - not feasible due to security concerns on where the trap could run and cost of executing custom procedure on every access.
- Disadvantage of ACLs: tough to add/delete users, expensive to check each access

1.3 Hierarchical Control

- All objects are categorized into a hierarchical tree of directories
- Objects that can modify a directory entry can modify ACLs of all objects nested inside that directory

1.4 User Authentication

- Unique username and password (len = 8 ASCII chars)
- unauthenticated batch jobs held in a queue until user authenticates themselves interactively
- Proxy login: allows user B to log in with their creds and request that a job be run under user A's principal identifier (only if A has provided a list of proxies to be used)
- Provides password generation program
- Passwords stored in encrypted form
- Passwords are not printed
- Auto logout is implemented after some specified interval
- After 10 wrong login attempts, network connections are broken
- Init procedure scheme allows full project admin control over processes in the group

1.5 Primary Memory Proection

- Second level **descriptor-based** protection on top of ACL implemented in h/w
- Virtual address space of a process implemented using a *descriptor segment* array
- Each reference to virt mem consists of a **segment number** (index into the descriptor segment) and a **word number** (offset)
- Three classes of descriptor extensions for protection purposes:
  - mode control
  - protected subsystem entry control
  - and control on which protected subsystems may use the descriptor at all
- Back pointers exist between segment-wise ACL on disk and virtual segment-wise descriptors to propagate any changes to all shared segments
- Descriptors enable hardware checking for calls to protected subroutines which allows to implement things like debuggers, which cannot be accidentally disabled

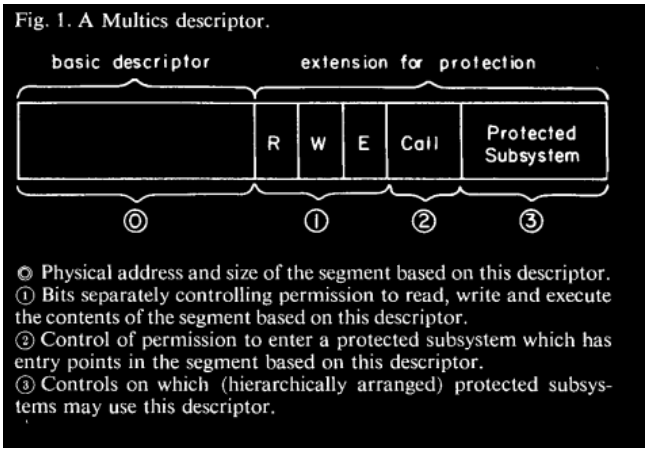

Figure 1: Descriptor in Multics

1.6 Protected Subsystems using Rings

- These are subsystems that have well-defined entry points and are available at the user level
- The rightmost descriptor field forms protected subroutines
- Each subsystem is assigned a number (0-7) and the hardware only allows access to a descriptor for subsystems whose assigned number is $\geq$ the descriptor's value for the field
- Some descriptors allow subsystems to access the entry points of other protected subsystems
- This sets up a scheme called the "rings of protection" (0 - innermost and most privileged ring, 7 - outermost and least privileged ring)

2 Singularity: Rethinking the Software Stack

- Microkernel based OS
- Three main architectural ideas:
  1. software-isolated processes - protected execution environment
  2. contract-based channels - fast, verifiable comm between processes
  3. manifest-based programs = list verifiable behaviours
- **Sing#:** an extension of C# provides verifiable, first-class support for OS communication primitives, strong support for systems programming
- The ABI provides safe access to memory, exec resources and messages

2.1 Software Isolated Processes

- SIPs do not share writable memory, and are isolated by software verification (programming language memory and type safety, not h/w protections)
- Cannot dynamically load or execute code, sealed at execution time (extensions happen via separate SIPs)
- Singularity uses static verification and runtime checks to verify that SIP code cannot access memory regions outside the SIP
- Inexpensive to spawn, avoids layers of partially redundant mechanisms and policies

2.2 Contract-Based Channels

- Channel: bi-directional, lossless, provides in-order message queue
- Contract: Written in Sing#, consist of message types and possible protocol states that signify the interaction between processes
- Compiler can verify that code never lands in an invalid protocol state, and contract verifier checks that programs using contracts are in compliance with the contract definition

2.3 Manifest-Based Programs

- Users invoke manifests to start programs
- Manifest describes the program's code resources, required system resources, its desired capabilities, and its dependencies on other programs.
- Primary purpose of manifest is to allow static and dynamic verification of program's intent and behaviour

2.4 Singularity Kernel

- Provides the three resources mentioned above in addition to scheduling, abstractions of hardware
- Most of the kernel is implemented in Sing# which is memory and type safe (some parts in unsafe Sing# or C++)
- SIPs access kernel primitives through the ABI
- ABI maintains a system-wide isolation invariant i.e. one SIP cannot modify another SIP's state through the ABI
- The kernel can place privileged code in-line into trusted functions within SIPs at install time to optimize performance

2.5 Memory Management

- The address space is logically partitioned into a kernel object space, each SIP's object space, and the **exchange heap** for communication of channel data
- Only one SIP can access a memory block in the exchange heap at a time
- This property is statically checked and enforced: SIP has at most one pointer to a block during its execution
- SIPs pass ownership of heap blocks to other SIPs by passing pointers in messages