A Hardware Architecture for Implementing Protection Rings

Gaelen Hadlett

Introduction

- Why?
- What?
- How?
- Where?

Access Control

- Protection of Computation and Information
- Multiple Users with Different Goals
    - Interaction or Exclusion
- Stored Information
    - Data and Executables

Evaluation Criteria

- Four Criteria to Judge Usefulness
  - Functional Capability
  - Economy
  - Simplicity
  - Programming Generality

Functional Capability

- Meet Variety of Protection Needs
- Natural Flow
- Maximize Capabilities

Economy

- Complexity, Inconvenience, Storage
- Keep Cost Low To Keep Options Open
- Cost Proportional To Capability
  - Specific and General Needs

Simplicity

- Tied to Economy
- K.I.S.S.
  - Complexity Implies Insecurity
- Easily Understood
  - Confident in Usage

Programming Generality

- Combinable Procedures
- Unaffected Internal Structure

Virtual Memory Overview

- Independent Segments
  - Addressed (*s*, *w*)
- Segment Descriptor Words
  - Addresses Single Segment
- Processor Provides Translations
  - Occurs For Every Access
  - Facility for Separate Memory

Multics Overview

- Three Basic Assumptions
  - Process with New Virtual Memory for Each User
  - Storage Organized as Collection of Segments
  - Access Limited to Users by Segments Access Control List

Access Control Mechanisms

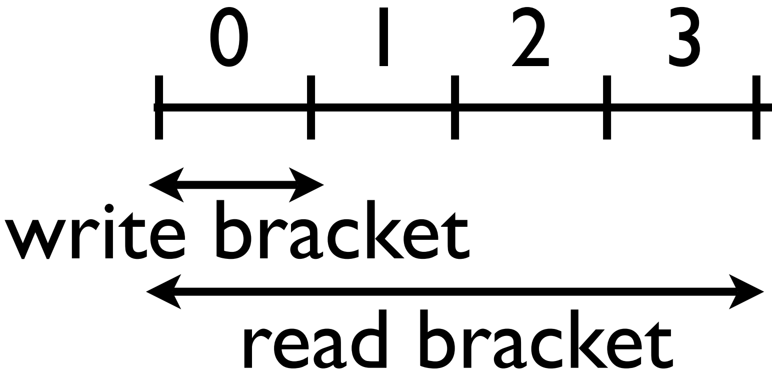
- Write, Read, Execute Flags
  - Stored in *sdw* of Segment
- Access Determined in Translation
- No Provisions for Level of Access

Protection Rings

- Fixed Number of Domains
- *r* rings from *0* through *r-1*
- Decreasing Capabilities
- Capabilities of *m* subset of *n*, where *m* > *n*
- Highest Level of Access at Ring *0*

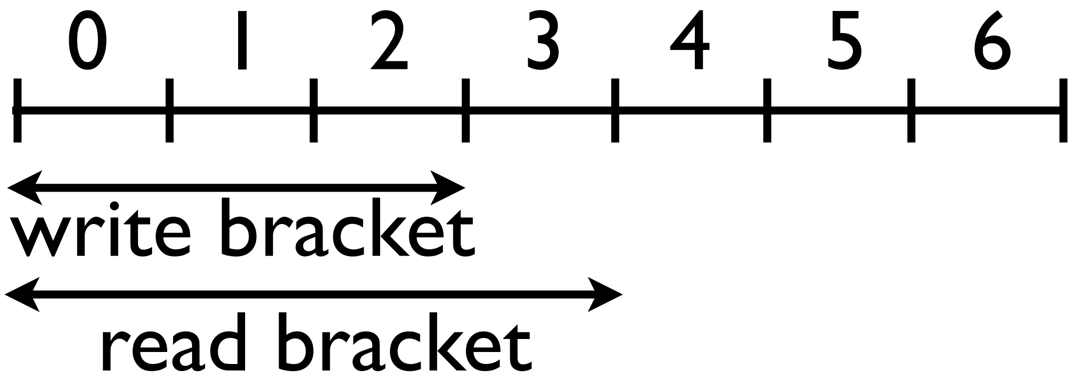
Protection Rings

- Read, Write, Execute Flags
    - Encode *off* or level of extent



Read = *on*
Write = *on*
Execute = *off*

Partial Implementation

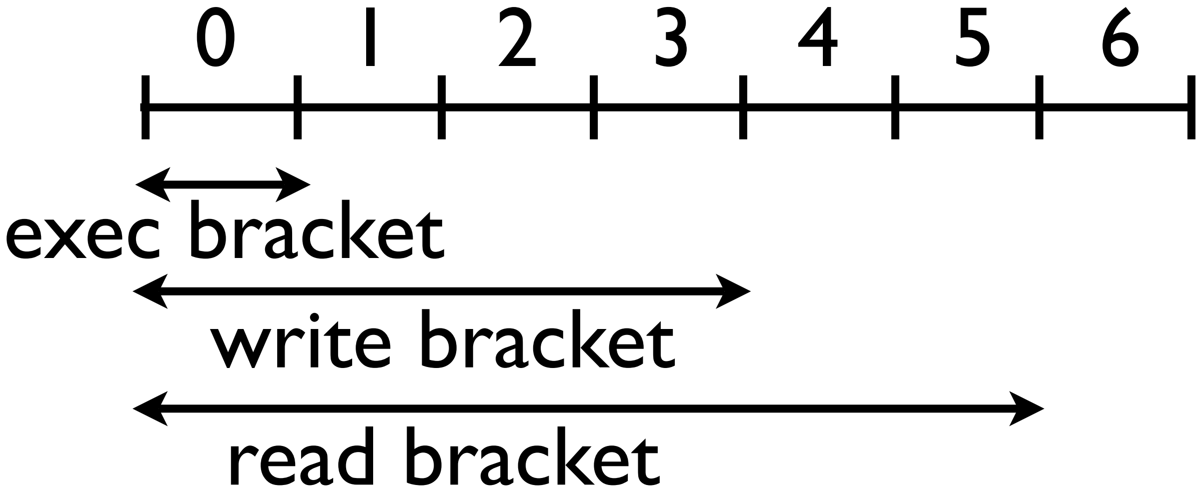


0 1 2 3 4 5 6

write bracket

read bracket

Read = *on*
Write = *on*
Execute = *off*



0 1 2 3 4 5 6

exec bracket

write bracket

read bracket

Read = *on*
Write = *on*
Execute = *on*

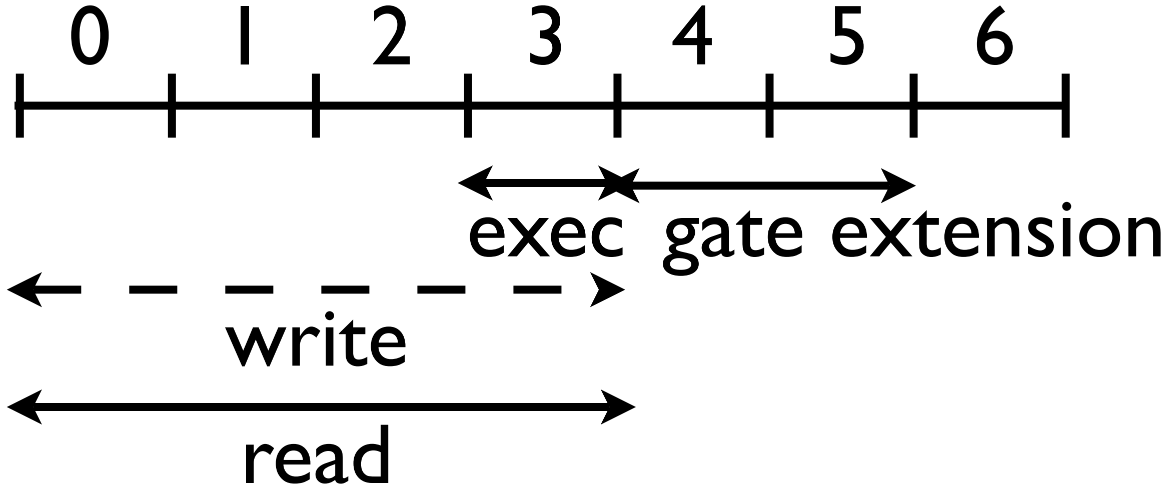
Protection Rings

- Protected from higher rings
- Changing domains carefully controlled
  - Changes restricted through *gates*
- Execution must be transfered to *gate*
- *Gate extensions* provide *gate* transfer
- Also stored in *sdw*

Protection Rings

- Upward Transfers
  - No special gates required
  - Following instruction must be executable in new ring
- Execution Bracket
  - Not always lowest level
  - Standard libraries have high level

Gate Extentions



Read = *on*
Write = *off*
Execute = *on*
Gate list = *0, 1, 2*

Access Calls

- Downward Calling
  - More Access
- Steady Calling
  - Same Access
- Upward Calling
  - Less Access

Downward Calling

- Assume cooperation from lower rings
- Call to *gate* of lower ring
- Called function has full access
  - Implied by nested structure

Downward Calling

- Three Problems
  - Called area must find new stack area
  - Way to validate references
  - Positive about callers ring level

Steady Calling

- Call and Return Without Ring Change
- No Protection Problem
  - Same Set of Access Capabilities
- Same Mechanism as Downward Calling

Upward Calling

- Upward Call, Downward Return
- Ring *n* Calls Ring *m* Where *m* > *n*
- Two Major Problems
    - Caller References Protected Arguments
        - Called Has Less Access
    - Gate Required For Return Call

Conclusion

- Easily Create Protected Subsystems
- Layered Implementation of Supervisor
- Self Protection

References

Schroeder, M. D. and Saltzer, J. H. , A Hardware Architecture for Implementing Protection Rings, Communications of the ACM 15(3), March 1972, pp. 157-170.

Questions?