1 Implementing ACLs

For this problem, assume each object has a single ACL, and ACL entries are of the form (Subject, Permission, GRANT or DENY). Subject may be a user or, if the system supports groups, a group. Permission specifies an action (e.g. read or write). Each entry specifies whether the permission is GRANTED or DENIED.

Recall that in some access control list systems, access is granted if any entry in the ACL GRANTS permission, some deny access if any entry in the ACL DENYS permission, and some GRANT or DENY based on the first rule that applies to the current request.

This question asks you to contemplate different data structures for implementing ACLs in these different settings. Recall that we need to perform two basic operations on ACLs:

- Given a Subject and a Permission, determine whether the ACL GRANTS or DENIES that Permission to that Subject.
- Add and delete entries. If ACLs are order-dependent, then a position must be specified when adding the new entry.

If a system only has users (but not groups), what kind of data structure (e.g. tree, hash table, linked list, something else) would you use to represent order-independent ACLs? Order dependent?

If the system supports users and groups, what kind of data structure would you use to represent order-independent ACLs? Order dependent?

Do you conclude that order-dependent or order-independent ACLs are more efficient, or are they the same?
2 Hardware Capabilities

Imagine a CPU that uses capabilities to govern access to memory. A capability \((p, l)\) grant access to the memory starting at \(p\) and extending for \(l\) bytes.

The hardware designers attempt to build this system by taking a standard load/store CPU and making the following changes:

- They add new capability registers \(c_1, \ldots, c_8\). Each capability register holds a capability \((p, l)\).
- They modify load so that you must use a capability register to specify the memory address, i.e.
  \[
  \text{load } \%\text{dest}, \%c_i, \%r_j
  \]
  This loads the word at the offset specified in general-purpose register \(r_j\) in the region specified by capability \(c_i\) into the register specified by \(\text{dest}\). Note that \(\%\text{dest}\) may be a general-purpose or capability register. If \(r_j\) exceeds the length of the region specified in \(c_i\), then the instruction generates a segfault.
- store is modified analogously.

Analyzing and completing the basic system:

- Give the interface for store, similar to how load is specified above.
- The system, as currently described, is flawed. Show how to obtain a capability to any address on the system. (Hint: forge a capability).
- Describe how to fix the issue you exploited above. You may modify the CPU, RAM, etc. Simpler modifications are better.

Now consider trying to support object-oriented programming on such a system. We want objects to have public methods, and we want to ensure that other code can only jump to the first instruction of a public method. Furthermore, we want to enable objects to have private data, which can only be accessed while executing code from that object (or code that it calls and explicitly passes a capability).

- How can you enforce public interfaces in hardware? (Hint: make the program counter a capability register, extend capabilities to specify read, write, and/or execute permissions)
- How can you set it up so that class methods can access private data from objects of that class, but no other part of the program can? (Hint: treat any capability in the program counter as implicitly having “read” permission)
Finally, let’s build some run-time tools

- Describe an instruction for creating a sub-capability from a capability.
- Show how to use this instruction to implement malloc.
- What about free?
- What about garbage collection?