CS510 Languages and Low Level Programming: Portfolio submission, Topics 7 and 8

Due on May 20, 2016 at 11:59pm

Mark P. Jones Spring 2016

Konstantin Macarenco

Topic 7. Explain the use and implementation of capabilities in access control and resource management.

Capabilities is a method of access control that gives high level of granularity and implements "principle of least privilege".
Main key points of capabilities:

- Capabilities don't care who owns them.
- Capabilities can be delegated (with the same or smaller level of access rights).
- Capabilities can be Copied (with the same or smaller level of access rights).
- Define specific set of restrictions (same file can be unlocked by two different capabilities with different level of access).

File descriptors comes very close to this definition, they can be copied, moved to any process or completely removed by the system, and defines level of access, i.e. read write, or execute.
Although it sounds good in theory this approach is hard to implement in existing system - it is incompatible with current implementation of the popular OSes.

Implementation details (taken from seL4):
All capabilities are stored in protected area (within kernel), and not visible from the outside. Each process has a capability space (CSpace), and as system starts up set of capabilities installed to each process, according to some policies. Each capability has assigned metadata to store some vital information, for example Window Capability might store set of permissions, Untyped Capability has to store pointer to the next available address, etc.
Capabilities management is done by using capability derivation tree, which for performance purposes implemented as doubly link list (efficient sequential traverse and quick insert/delete operations). This way capabilities can be easily shared with other processes without system need to track each one of them.
When a process requests some action, kernel checks if the process has appropriate capability installed in it's CSpace, if not found action is rejected. In other words they can't be forged or guessed like Global Thread IDs in L4.

Topic 8. Develop programs using the capability abstraction provided by the seL4 microkernel, or capabilities lab.

To meet this objective I Completed capability lab, except for the task 8. During the lab I learned basic principles of working with capabilities, and confident in understanding them, although real seL4 experience probably would be more valuable.
In part 7 of the lab (Improving existing operation) I picked the second approach, even though a bit more troublesome to implement, but should be faster. *printString*() system call, provides address to the region of memory to be printed, however this is not safe and should be checked for:

1. The desired address is not in any way interfere kernel address space. For example it can be a pointer to some area in the kernel space, or a pointer to a not null terminated region of memory that overlaps with kernel's address space.
2. The Array to be printed must lay within existing memory, it should not point or overlap with any untyped regions.

During system call kernel finds memory page, where the string belongs and last page that can be reached by from that point, check for null in this region.