EXTENDED ABSTRACT

THE CASE FOR CAPABILITY BASED COMPUTERS

R. S. Fabry
Computer Systems Research Project, University of California, Berkeley

The idea of a capability which acts like a ticket authorizing the use of some resource was developed by Dennis and Van Horn as a generalization of addressing and protection schemes such as the codewords of the Rice computer, the descriptors of the Burroughs machines, and the segment and page tables in computers such as the GE-645 and IBM 360/67. Dennis and Van Horn generalized the earlier schemes by extending them to include not just memory, but all systems resources: memory, processes, input/output devices, and so on; and by stressing the explicit manipulation of access control by nonsystem programs. The idea is that a capability is a special kind of address for an object, that these addresses can be created only by the supervisor, and that in order to use any object, one must address it via one of these addresses. The name comes from the fact that having one of these special kinds of addresses for a resource provides one with the capability to use the resource.

The use of capabilities as a protection mechanism has been the subject of considerable interest and is now fairly well understood. Access control schemes using capabilities and capability-like notions are, as a whole, the most flexible and general schemes available. It will in fact be assumed that the reader is familiar with the advantages of capabilities for protection putposes; a somewhat different advantage of capabilities will be developed here.

It will be argued that there is a substantial advantage in using capabilities as a basic component of the address of every object which is not a part of the processor doing the addressing. In order to accomplish this, user programs must be able to store capabilities freely into various permanent user data structures (subject, of course, to some scheme for preserving the integrity of the representation of capabilities). Not all schemes which use capabilities actually allow capabilities to be used as permanent addresses in this way. For example, the original Dennis and Van Horn scheme did not, because it insisted that capabilities be stored only in C-lists associated with computations; one could not use capabilities to construct permanent user data structures.

The advantage of a capability used as an address is that its interpretation is context independent. It provides an absolute address for an object. This fact is more important than it may at first appear.

Before the use of address relocation in the form of base and limit registers, paging, and segmentation, a number of independent jobs would have been allocated fixed areas of physical memory in which to run. Addresses within the jobs would be relocated at load time and a job would not be moved once it had started running. The lack of the ability to relocate jobs in memory severely limited the freedom of the memory allocator and resulted in underutilized computers. To avoid this underutilization, the various forms of address relocation have been introduced. They allow the jobs in memory to be moved around independently of each other. But in increasing the freedom of the memory allocator, a new problem has been introduced. Consider the case of two jobs which need to interact with each other. In the primitive system without relocation, jobs share a single address space and could be allowed to interact freely, sharing data structures and addresses as easily as if they were a single job. As soon as address relocation is introduced into the system, however, addresses no longer have a fixed interpretation; their meaning becomes context dependent; each job has its own address space, or perhaps even several. This fact has generally been interpreted as an advantage: Base and limit registers, paging, and segmentation, by virtue of their address relocation, allow users to be easily and totally isolated from each other, thus providing a form of protection of one job from another. On the other hand, the addressing of shared objects has become more difficult, and this side effect is generally ignored. This effect is particularly ironic for those systems which stress their usefulness for cooperating users who want to work together, sharing programs and data.

I am familiar with two attempts to design a capability-based computer in which every explicit memory access uses an address in the form of a segment capability and work number pair and which allow capabilities to be directly manipulated by user programs in the traditional ways that addresses are used. The first system to be designed was the Chicago Magic Number Computer developed by the Institute for Computer Research at the University of Chicago. This system was never completed. The second system to be designed was the System 250 built by the Plessey Company in England. The Plessey system has been built and running for some time now and is available comercially. Based on experience with these two implementations, a number of implementation considerations have been clarified.

Certain recent advances eliminate the need for the modification of the representation of capabilities by the operating system and suggest how to solve the own variable problem in a general way. These advances eliminate the major implementation problems of previously designed systems.