7

The StarOS System

7.1 Overview of StarOS

Carnegie-Mellon's Hydra/C.mmp project examined the use of multiprocessors in the solution of artificial intelligence problems. C.mmp supported up to 16 processors and memories connected through a crossbar switch. By 1975, however, it was clear that multiprocessors involving hundreds of microprocessors would be possible. The C.mmp crossbar scheme, which increases geometrically in complexity with the number of processing elements, was infeasible for such systems. Therefore, the CM* project [Jones 80a], started in 1975 at Carnegie-Mellon, took a different approach to interconnection—one that grows linearly in complexity with the number of processing elements. By 1979, the CM* configuration contained 50 operational processors.

CM* consists of a large collection of *computer modules*, in which each computer module is a DEC LSI-11 processor with its bus, local memory, and peripherals. A computer module *cluster*, shown in Figure 7-1, is formed by a set of computer modules communicating through a *map bus*. Memory requests generated in each computer module are routed by a switch, either to local memory or to the map bus. The CM* system consists of a set of clusters connected by an *intercluster bus*. A computer module can issue addresses for local, intracluster, or intercluster memories.

The connection between clusters is managed by a unit called the *Kmap*. The Kmap is a horizontally microprogrammed processor that, in addition to supporting intercluster refer-

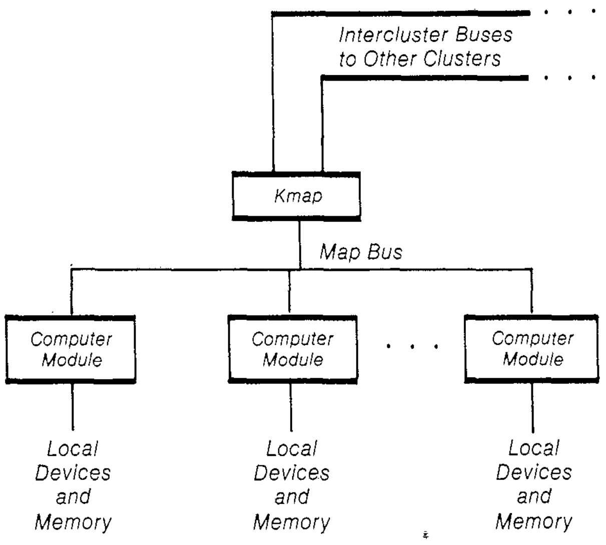


Figure 7-1: A CM* Cluster

ences, is used to execute operating system functions. Performance-critical parts of the operating system, such as capability operations, are therefore implemented in Kmap microcode.

Two operating systems were constructed to support distributed software for CM*: STAROS and Medusa [Ousterhout 80a, Ousterhout 80b]. STAROS, the subject of this chapter, is an object-based operating system that supports the execution of task forces [Jones 78b]. A *task force* is a collection of cooperating processes executing concurrently to perform a single job. Task forces are distinguished from most cooperating process schemes by their dynamic nature. The structure of a task force corresponds to the available resources rather than to the functional requirements and can change with dynamic resource changes.

In general, each of the processes within a task force is small if measured by its resource requirements. A task force process executes within a small domain and interacts with other task force processes for many of its needs. STAROS objects reflect the constrained needs of this environment, and the structure is much simpler than that of Hydra. The following sections take a brief look at object structure and addressing in the STAROS operating system.

7.2 StarOS Object Support

All information in the StarOS system is contained within objects. Each object has a type, and the type defines the operations that can be performed on the object. As with Hydra, objects are addressed by capabilities that name the object and specify the permitted rights to the object.

A StarOS object contains two parts, a *data portion* and a *capability portion* (or C-list). The portions are stored in a single contiguous memory segment. Objects cannot grow dynamically and therefore retain the size with which they were created. The data portion is located at the low-address end of the segment, and the capability portion is located at the high-address end. A process possessing a suitably privileged object capability can directly manipulate the data portion of the object with processor data instructions.

A StarOS process can directly address 64K bytes of memory (local or remote) at any time. This limit is dictated by the 16-bit PDP-11 addressing architecture. StarOS partitions this address space into 16 4K-byte windows. Each StarOS object has a maximum size of 4K bytes in its data portion and 256 slots in its capability portion. A suitably privileged process can request that an object's data portion be mapped into one of its windows, allowing direct instruction access.

The StarOS kernel defines a small set of object types, as listed in Table 7-1. These are known as *representation types*, and

BASIC OBJECT	Segment with data portion and C-list.
C-LIST	Basic object with capability portion only.
PROCESS OBJECT	Schedulable entity that contains the root C-list for addressing.
STACK OBJECT	An object supporting PUSH and POP stack operations.
DEQUE OBJECT	A two-ended stack, supporting PUSH and POP at head and tail.
DIRECTORY OBJECT	An object containing descriptors of physical object information.
DATA MAILBOX	An object for sending and receiving data messages.
CAPABILITY MAILBOX	An object for sending and receiving capability messages.
DEVICE OBJECT	The representation of a physical I/O device.

Table 7-1: StarOS Representation Types

instances of these types are known as *representation objects*. Operations on representation objects are supported by calls to STAROS. All other objects are implemented by user-defined type managers that construct other abstractions out of the basic representation objects. These user-defined types are known as *abstract types* and their instances are called *abstract objects*. Thus, an abstract object has an abstract type, which indicates the operations that can be performed on the object, and a representation type, which indicates the kernel type from which that object is constructed.

7.3 STAROS Capabilities

All references to STAROS objects, representation or abstract, are made through capabilities. A STAROS capability is 32-bits long and contains a 3-bit type field, a 13-bit rights field, and a 16-bit data word field, as illustrated in Figure 7-2. The interpretation of the data word depends on the capability type. STAROS supports several capability types, and the capability type field specifies one of the types listed in Table 7-2. The data capability is used to transmit small amounts (16 bits) of information efficiently without requiring the creation of a basic object and its overhead. The representation and abstract capabilities contain unique 16-bit names in their data words. A type manager token capability contains a unique 16-bit type identifier in its data word, allowing the possessor to operate on abstract objects of that type.

The capability rights field consists of several type-dependent and type-independent fields, as illustrated in Figure 7-2.

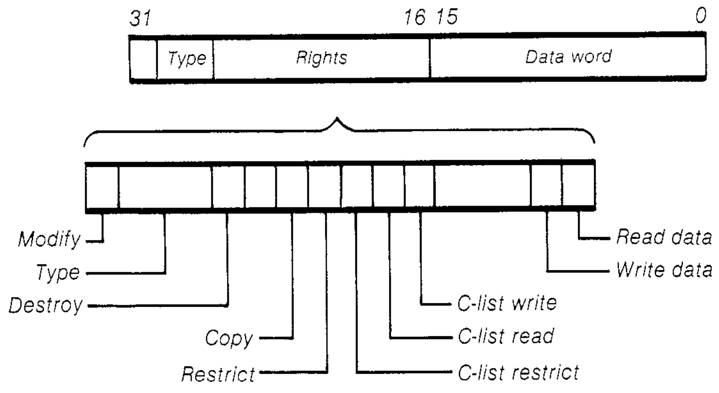


Figure 7-2: STAROS Capability and Capability Rights Word

REPRESENTATION CAPABILITY	Names one of the kernel-defined representation objects and contains kernel-interpreted rights to the object.
ABSTRACT CAPABILITY	Names an abstract object and contains type-specific rights.
TOKEN CAPABILITY	Identifies the owner as the possessor of a special privilege (for example, as the garbage collection process or as the type manager for a specific type).
NULL CAPABILITY	Marks an empty slot in an object's capability part.
DATA CAPABILITY	Contains a 16-bit data value in its data word.

Table 7-2: STAROS Capability Types

Bits 0-7 of the rights word contain rights to the object addressed by the capability. For an abstract capability, this 8-bit field is defined and interpreted by the type manager. The rights shown in Figure 7-2 are for a representation capability for a basic object. Basic object rights permit reading and writing of the data part, loading and storing of capabilities in the C-list, and restriction of capability rights in the C-list of the object to which the capability points.

The *copy* and *restrict* rights apply to the capability itself and indicate whether or not the capability can be copied or if rights in it can be restricted. A capability without restrict rights can never be deleted, so new copies of capabilities are always given restrict rights. Finally, the *modify* and *destroy* rights are generic object rights, and specify whether the addressed object can be destroyed or modified in any way. Modify rights operate as in Hydra—modification of an object requires modify rights in each capability along the path to the target object.

7.4 Object Addressing

Each representation object or abstract object is addressed through a capability that contains its 16-bit unique name. At any time there can be many capabilities for an object, but there is only one 16-byte *descriptor* for each object. The *descriptor*, which corresponds to a Hydra active fixed part, is located on the cluster on which the object is stored. The format of an object descriptor is shown in Figure 7-3.

The garbage collection process uses the color field to indi-

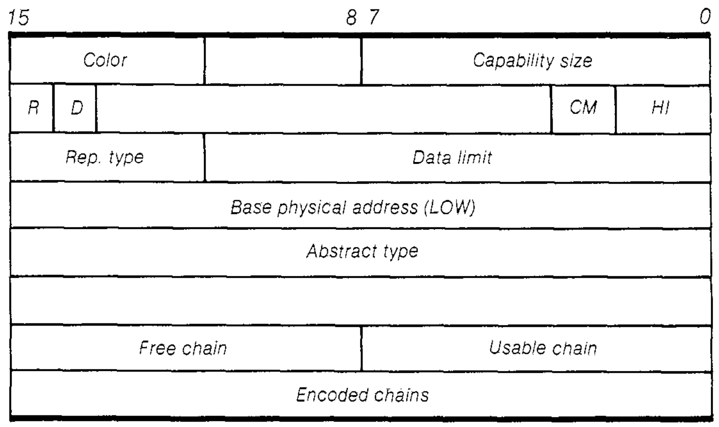


Figure 7-3: STAROS Object Descriptor Format

cate the garbage collection status of the object (for example, whether a capability for the object has been passed outside the local cluster). The capability size and data limit fields specify the size of the capability portion (in slots) and data portion (in bytes) of the object. Since the object is stored contiguously, these fields determine the total size of the object and the position of the dividing line between data and capability portions.

The object's primary memory location is formed by concatenating the base physical address field with the 2-bit HI field. This 18-bit address is local to the cluster processor specified by the computer module number (CM). An object must be stored on the same cluster as its descriptor, although capabilities for an object can be passed outside the cluster. Two type fields contain the abstract type of the object and the representation type used to implement it. Finally, the chain fields are used to form linked lists of descriptors, and R and D are reference and dirty bits, respectively.

Descriptors are stored in *directories*. Each CM* cluster can have up to 32 directories, each containing up to 256 descriptors. A single *root directory* in each cluster contains descriptors for itself and the 31 subdirectories. STAROS 16-bit object names, contained in both abstract and representation capabilities, directly locate an object descriptor in one of these directories. A unique name specifies a 3-bit cluster number, a 5-bit directory number, and an 8-bit directory index, as shown in Figure 7-4.

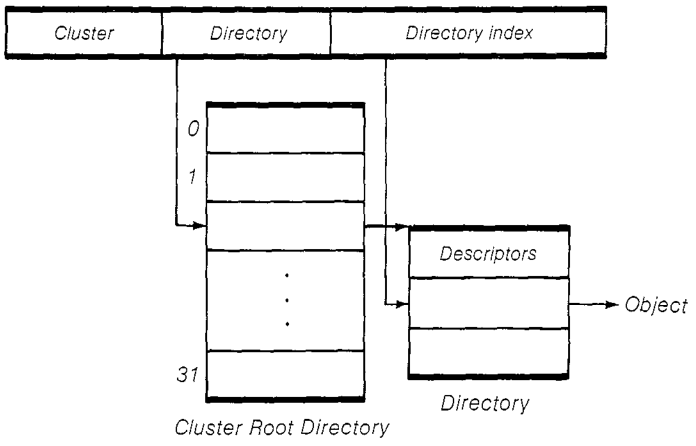


Figure 7-4: StarOS Directory Structure

7.5 StarOS Abstract Type Management

As previously stated, a type manager creates each new abstract object from one representation object (usually a basic object). The type manager returns an abstract capability for a new abstract object to a caller, but only the type manager can operate directly on the representation object implementing the abstraction. The possessor of an abstract capability can only use it as a parameter in a call to the type manager to request an object operation.

The key to a type manager's special ability is its *type token*, one of the capabilities previously described. Every type manager possesses a type token whose data word contains a unique identifier for its type. The type token is never given out except to procedures that are part of the type manager. The type manager uses the type token in the following way:

- When a process wishes to create a new abstract object, it calls the appropriate type manager. The type manager, through a call to StarOS, creates a new representation object, for which it receives a fully-privileged representation capability. The type manager then uses this capability to initialize the object as needed.
- After the object has been initialized, the type manager executes an ASSOCIATE TYPE instruction, specifying the object's representation capability and the manager's type token as parameters. This instruction stores the abstract type field from the token into the object's descriptor. The ASSOCIATE

TYPE instruction thus creates an *abstract object* from a *representation object.*

- Next, the type manager executes a DEAMPLIFY instruction to transform its fully-privileged representation capability into an abstract capability. The DEAMPLIFY instruction simply changes the type field in the capability from "representation" to "abstract."
- The type manager then returns the abstract capability to the caller. This abstract capability identifies the holder as having authority to request operations on that object. It cannot be used to access the encapsulated representation object directly.
- To perform an operation on the object, the holder of the abstract capability calls a type manager procedure, passing the abstract capability as a parameter. The type manager then executes an AMPLIFY instruction, specifying as operands the abstract capability and the type manager's private type token. If the type token's type matches the object's abstract type, the AMPLIFY instruction turns the abstract capability back into a fully-privileged representation capability, allowing the type manager to access the representation object.

7.6 Discussion

It is interesting to note the ways in which StarOS differs from the Hydra object model. StarOS limits direct access of an object's representation to the type manager. Two basic types of capabilities are provided: representation capabilities used to access kernel types, and abstract capabilities passed to users of type manager implemented objects. By turning a representation capability into an abstract capability, the type manager *seals* the capability with its special type token. Although the abstract capability has the object ID sealed within it, it cannot be used to access the object's representation. The type token is the key used later to *unseal* the capability, returning a representation capability that can manipulate the object. In this way, the type manager always receives full privilege to access any of the objects whose representation it controls.

Type tokens are a simplification of the Hydra amplification template. Hydra permitted more precise control of object access; an amplification template could be used to amplify only those rights needed by the type management procedure. In contrast, the StarOS type token mechanism always gives the type manager complete access to one of its objects.

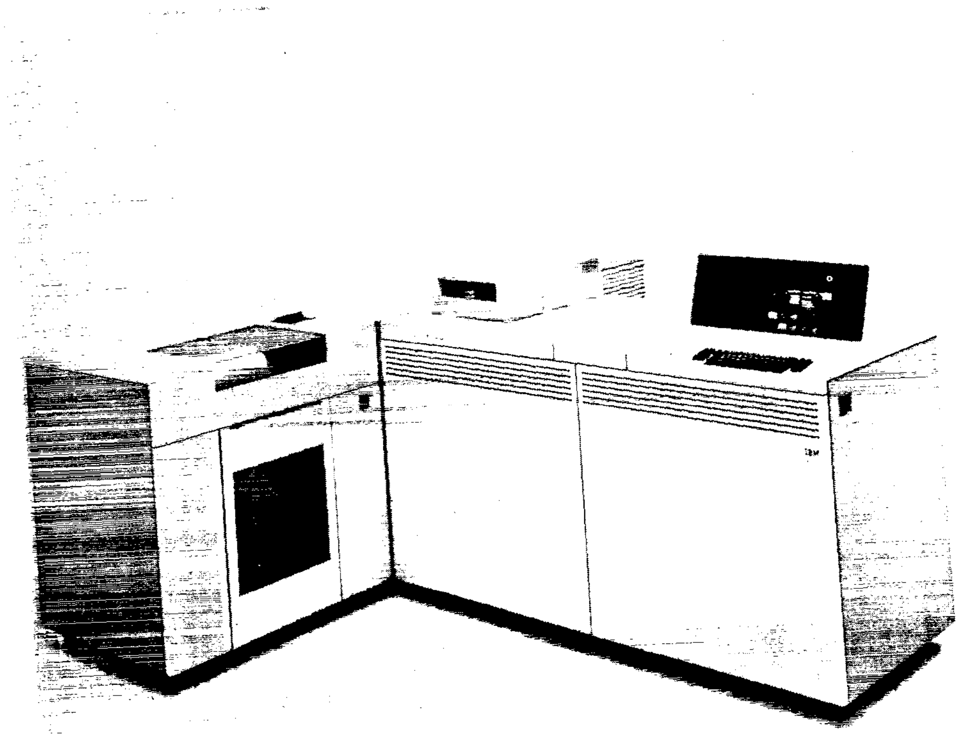
The type token is thus a special type of capability used to seal or unseal another capability. Tokens are also used to identify specially privileged processes. Because tokens are capabilities, they are stored in C-lists and therefore cannot be fabricated by users. The data capability provides an efficient means for transmitting or sharing one word of information without creating a single-word object. Data capabilities also allow small amounts of data to be sent to a capability mailbox.

Another interesting feature of StarOS is its return to a small object address space. An object's unique ID, 16 bits in length, can be used to directly locate the descriptor for an object, thus simplifying the manipulation of capabilities and objects. The structure of the ID implies that the system can support a maximum of 8K objects per cluster on each of 8 clusters. The ID leads directly to a particular cluster. Of course, this scheme makes it difficult to move an object from one cluster to another because the address is not location independent. Indeed, objects are never relocated in this way.

Finally, the implementation of operating system functions in Kmap microcode had significant performance impact. For example, a standard capability operation on StarOS takes 100 microseconds, while a similar operation on Hydra takes 1 millisecond. The ability to access an object's data portion directly is more significant. Once an object is mapped through an addressing window (at a cost of about 70 microseconds), data words can be accessed directly in several microseconds. The Hydra overhead for copying data from and to the object data-part is a millisecond.

7.7 For Further Reading

A more detailed description of StarOS is provided in [Gehringer 81], and a description of CM* switching structure and addressing can be found in [Swan 78]. The StarOS task force concept is presented in [Jones 78b]. Performance measurements for StarOS (in comparison with Medusa, a second operating system developed on CM*) can be found in [Jones 80a], which also discusses CM* and some of its applications.



The IBM System/38 computer. (Courtesy International Business Machines.)