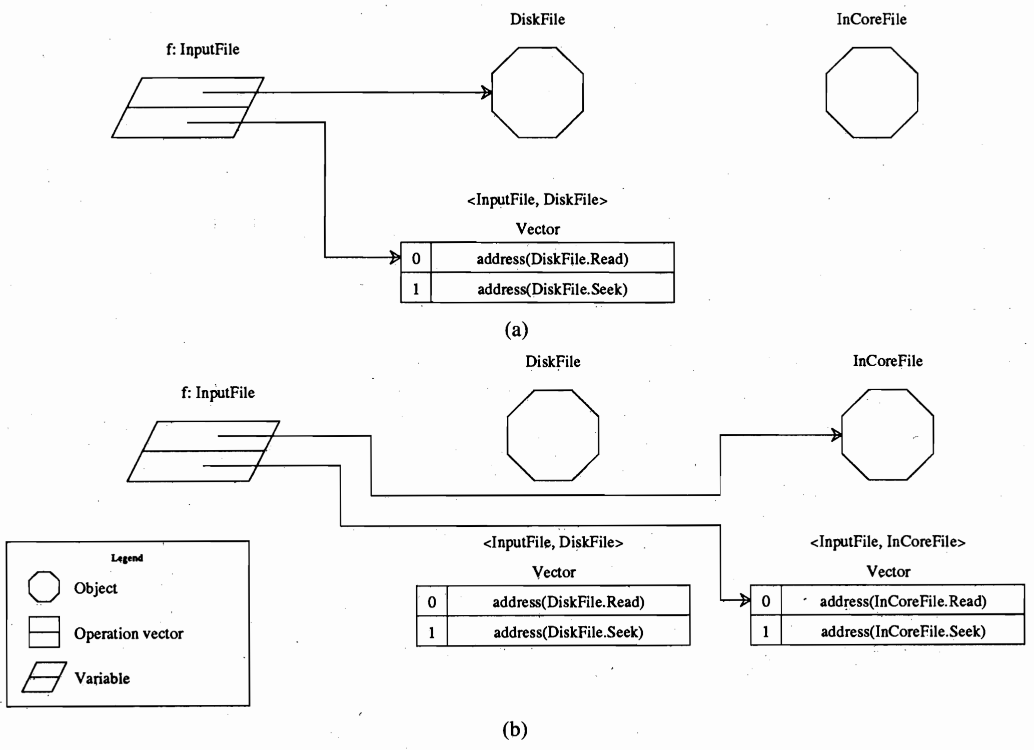

Fig. 3. (a) Before the assignment. (b) After the assignment.

the abstract type may be invoked on the object named by the identifier. The content of the vector is determined by the implementation of the object currently named by the identifier, since it consists of the addresses of the machine code sequences that implement the operations. Consider a variable f of abstract type a. The operations that may be performed on f are determined by a; the addresses of the appropriate code for these operations depend on the implementation c of the object named by f. Thus the pair $<a, c>$ uniquely determines an *operation vector* associated with f. In Fig. 3(a), a is *InputFile* and c is *DiskFile*. The vector has one element for each of the *InputFile* operations *read* and *seek*; the values of these elements are the addresses of the corresponding *DiskFile* routines.

When an assignment is made to f, the contents of its operation vector may need to be changed. For example, when an *InCoreFile* object is assigned to f, the operation vector associated with f must become appropriate to the pair <*InputFile*, *InCoreFile*>, as shown in Fig. 3(b).

This scheme replaces the method lookup required by Smalltalk by a single indexing operation. The cost is an additional word of storage for the pointer to the operation vector, and occasional recomputation of the elements of these vectors on assignment. The operation vectors themselves may be shared between all identifiers of identical abstract type that name objects with the same implementation, since it is the pair <abstract type, implementation> that determines the contents of the vector.

V. Distribution Support

As previously stated, the principal objective of Emerald is to simplify the construction of distributed programs. System concepts such as concurrency, multiple nodes, and object location are integrated into the language. This differs from, for example, EPL, where distribution is layered on an existing language through the use of a preprocessor, and from Accent [27], where distribution is provided as an operating system facility.

In Emerald, objects encapsulate the notions of process and data and are the natural unit of distribution. At any time each Emerald object is located at a specific node. Conceptually, a node is an object of a system-defined type. Node objects support node-specific operations, thereby allowing objects to invoke kernel operations. Such access to the underlying kernel is analogous to that provided by *kernel ports* in Accent.

Programmers may choose to ignore or exploit the concept of object location. In a distributed system, objects must be able to invoke other objects in a location-independent manner. This facility makes network services transparently accessible. In Emerald, locating the target of an invocation is the responsibility of the system. An object is permitted to move between successive invocations, or even during an invocation. While applications can control the placement of objects, most applications can ignore location considerations since the semantics of local and remote invocation are identical.

Nevertheless, there are two reasons for making location visible to the programmer: performance and availability. In a network, the efficiency of interobject communication is obviously a function of location. An application can colocate objects that communicate intensely and thus reduce the communication overhead. Alternatively, numerical applications can achieve significant performance gains by placing concurrent subcomputations on different nodes. An object manager may increase availability by placing replicas of its objects on different nodes.