A Universal File Server

A. D. BIRRELL AND R. M. NEEDHAM

Abstract—A file server is a utility provided in a computer connected via a local communications network to a number of other computers. File servers exist to preserve material for the benefit of client machines or systems. It is desirable for a file server to be able to support multiple file directory and access management systems, so that the designer of a client system retains the freedom to design the system that best suits him. For example, he may wish to use the file server to support a predefined directory structure or as a swapping disk. The paper explores the design issues associated with such a file server and proposes some solutions.

Index Terms—Access control, directory, distributed computing, file server, filing system, garbage collector.

I. Introduction

THE TERM *file server* has come into some currency as a name for a utility, provided on a high performance local communications network, the utility having as its main purpose the storage and retrieval of bulk data. Such utilities have been implemented with various degrees of complexity and sophistication [5], [6]; possible uses range from conventional file transfer to applications such as core-swapping which are only sensible using a high bandwidth network.

At the simpler end of the spectrum, the server would appear to a client as just a backing store device, the client being totally responsible for any structure he wished to impose on the data. A file server in this style would be responsible for the algorithm used in allocating space on the storage medium (typically, disks), and for transfers between the network and the medium, but for little else. Requests to such a server might take the following forms.

1) Allocate n units of storage, and tell me the starting address.

2) Give me the contents of the n units starting at offset p from address x.

3) Write the following n units into the storage starting at offset p from address x.

4) The n units starting at address x are no longer required and may be reallocated.

Such a server would be extremely simple, and its only function is the control of the physical storage medium.

At the opposite end of the spectrum, file servers have been implemented with a very high-level interface. Such servers appear to their clients much as one would expect a full-scale filing system on a time-shared computer to appear. Such a server would certainly have the concept of a *file* as an abstract object, and would almost certainly support facilities for access controls on files, for directory structures containing groups of files, for the lookup of textual names of files, and for interlocks on simultaneous access to files. Such a server is likely to have the concept of a *user*, probably protected by a password as in conventional time-sharing systems.

Between these two extremes there is a wide range of choice, depending on design decisions as to which facilities of a filing system one chooses to centralize in the server, and which facilities one leaves the client to implement. The simpler extreme has as advantages its simplicity (in terms of design and of implementation) and the freedom it leaves its clients—they can each decide what form of directory structure they prefer, what access controls they desire, and what form the symbolic name of a file should take. The more complicated extreme has the advantage that a client needs to implement very little in his own computer to have available all the facilities of a sophisticated filing system. It also makes very convenient the sharing of data, of complete files, and of directories between different client computers.

In many environments, all of these potential advantages are simultaneously attractive. We would wish a simple, low-level, file server in order to share an expensive resource, namely a disk, whilst leaving us free to design the filing system most appropriate to a particular client, but we would wish also to have available a high-level system shared between clients. Further, we would prefer to have these various systems use the same storage equipment in order to make better use of it, to allow simultaneous use of it, and to allow sharing of data. It is the purpose of this paper to describe a resolution of some of the problems in the design of such a file server, and to show alternatives that remain available.

II. Division of Responsibilities

In order to provide the flexibility which we desire our file server to have, we will distinguish between high-level functions more properly associated with a filing system, and functions belonging to a *backing store server*. Given such a distinction, it will then be possible to specify as the basic interface to our file server the functions of a backing store server; filing systems, available to all clients, could then be implemented as superstructures built on the functions of the backing store server. It is our belief that many functional aspects of the management of data on backing store are common to a wide range of filing systems and other uses of mass memory, and it

Manuscript received June 14, 1978; revised November 8, 1978. This work was supported in part by the Science Research Council.

A. D. Birrell was with the Computer Laboratory, University of Cambridge, Cambridge, England. He is now with the Xerox Palo Alto Research Center, Palo Alto, CA 94304.

R. M. Needham is with the Computer Laboratory, University of Cambridge, Cambridge, England.

0098-5589/80/0900-0450$00.75

is these functions that we wish to include in the backing store server. These functions should be specified at as high a level as is compatible with generality of application. The remaining functions are specific to particular uses of the backing store server, and must be implemented by particular filing systems or other clients of the backing store server.

The major function which we consider to exist only in the filing system is the lookup of text names (or whatever abstract names a particular filing system cares to implement), and their translation into *internal names* which will be recognized by the backing store server. To perform this translation a filing system will presumably maintain one or more *directories*. A filing system will probably also impose some form of access controls distinguishing between its various users. The backing store server is concerned primarily with the maintenance of a set of *objects* on its storage medium. Such objects would each have an internal name, and all transactions accepted by the backing store server would be in terms of these internal names. The backing store server is the sole agency concerned with allocating and relinquishing space on the storage medium. In order that the backing store server can do this without any knowledge of the structure of the client filing systems, a separate and universal system of indexes is provided. This will be described in the next section. Given an internal name, the backing store server will be able to determine, probably by lookup in a centralized table, where to find the object on the storage medium. We will see below that this table will be useful for keeping other information regarding a particular object. With such a division of responsibilities, it would be feasible (indeed, straightforward) to use the backing store server simultaneously for a common shared filing system, for clients running private or shared filing systems, and for clients using the server just for backing storage.

Each client could decide at any time whether he wished to use the facilities of the common filing system, or wished to use just the facilities of the backing store server. A major design constraint is that each filing system should be able to coexist peacefully with all other users of the backing store server.

III. Management of Backing Store Objects

The backing store server is responsible for maintaining an object as long as that object is required by any client, and for recovering space when an object is no longer required. We propose here a management scheme which appears to have sufficient generality for all purposes, and which is acceptably efficient. The backing store server considers each object to be either a *segment* or an *index*; this distinction of type is a fixed property of each object. A segment can be read or written at will by any client using it; an index is accessed and altered only by the backing store server. An index contains, primarily, a set of internal names or *Inames* of other objects—these preserved Inames are numbered within a particular index as $1, 2, 3, 4, \cdots$. The backing store server will guarantee to maintain an object only so long as its Iname appears in at least one index. It is the responsibility of a particular client or filing system to ensure, by means of suitable requests to the file server, that the Inames of all objects in which it is interested appear in an index. Since an index is itself an object, its Iname can be preserved in an index (and should, if the index is not to disappear). There is one *master index* whose existence the backing store server will guarantee. The set of objects maintained by the backing store server can thus be seen as a general naming network; no restrictions are placed on this network, which could become cyclic.

In order to determine efficiently whether the space used by an object may be freed, the backing store server can use a mixed strategy of reference counts and asynchronous garbage collection, as adopted in the CAP filing system [1], [2]. Associated with every object is a reference count; this is incremented whenever the Iname of an object is preserved in an index and decremented whenever its Iname is removed from an index. It is in order to ensure that these side-effects of index operations occur that operations on indexes are constrained to be done inside the backing store server. For an object whose type is *segment*, the space may be freed if and only if the reference count falls to zero. Because of the possible cyclicity of the network, a garbage collector may be needed to determine which indexes may be freed—this is an acceptably efficient technique, since the garbage collector need only consider indexes and does not look at segments. Note that, as on the CAP, the existence of reference counts for indexes gives us an additional check on the correctness of our garbage collector.

The facilities provided by the file server for filing systems and other clients would include the following forms.

1) Create object of given type and size, and preserve its Iname at offset n in index x.
2) Change the size of object x by n units.
3) Preserve the Iname of object x at offset n in index y.
4) Remove entry n of index y.
5) Give the contents of n units starting at offset p in object x.
6) Write the following n units into the storage starting at offset p in object x.

Note that facility 1) ensures that the Iname of a newly created object is preserved in an index before it is given to a client, since otherwise the backing store server could validly free the space before the client had made use of facility 3).

Given such facilities, a client could implement his favorite filing system, provided he equips himself with the Iname of an index preserved in or via the master index. For example, if he wanted a simple two-level system he might well arrange a structure such as the following.

In his initial index he preserves one entry of type *segment*, and a number of type *index*. The indexes are going to contain the Inames of his users' segments, and correspond to *user file directories* or UFD's, and the segment corresponds to his *master file directory* or MFD. In his MFD he will maintain text names and access controls for the UFD's. In each UFD he will maintain a segment containing text names and access controls for users' files, and will preserve entries for each file. The resulting structure might be pictured as in Fig. 1.

It is the responsibility of a filing system to:

1) equip itself with the Iname of its initial index;
2) preserve the Iname of each user object in some index;

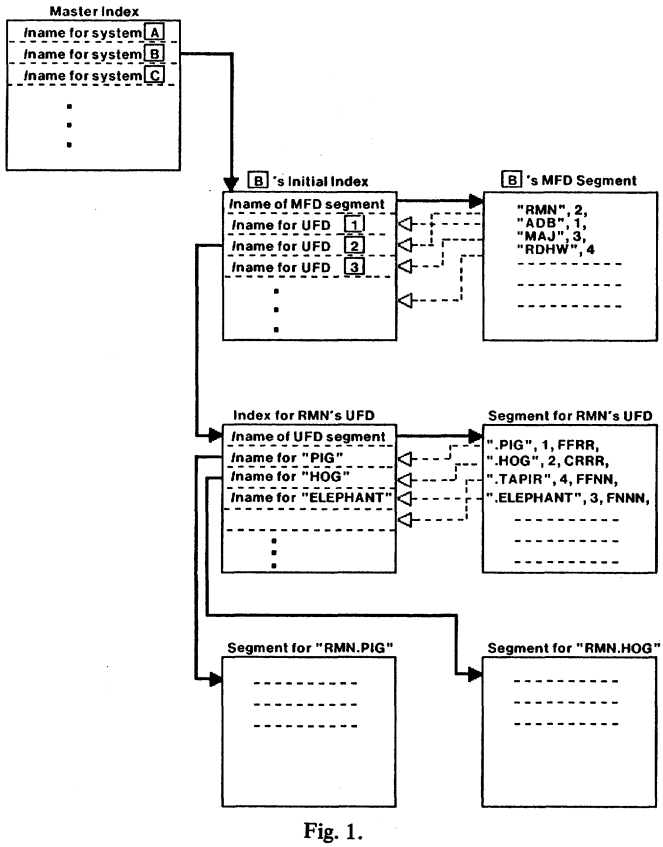

Fig. 1.

3) remove such entries when the user *deletes* the file.

Further, the filing system may provide access control mechanisms, translations to and from textual (or other) names, and make available to its users as much or as little of the underlying index network as it sees fit. Note that there is no reason why an object (even an index) should not be shared between filing systems, provided that they cooperate on its use. This is considered further below.

IV. Protection

If the backing store server is to be used by noncooperating clients and multiple filing systems, it will be necessary to have some degree of protection provided. The extent to which this is needed will depend on the clients. The other machines on the communications network may be:

1) distrusted totally, in that we require 100 percent assurance of noninterference; this may include assurance of data security;

2) distrusted, but we only require a *believable* assurance of noninterference;

3) trusted in relation to some subset of the objects;

4) trusted totally.

Case 4) might arise, for example, if the clients in question were computers with fixed burnt-in programs, whereas 1) or 2) would be appropriate for computers running experimental systems. We will assume here that the communications network has a suitable amount of security for client-server communication of data—the meaning of *suitable* may vary with the application, from total encryption to open broadcasting. We believe the most common requirement to be 3), provided the probability of interference (accidental or deliberate) can be made sufficiently small.

There are fundamentally two techniques available for protection in the backing store server/access control lists, or capabilities. For an access control list system, we would associate with each object a list of those clients entitled to use the object. This information would be maintained solely by the backing store server, independently of any access controls maintained by a filing system; its purpose is separation of filing systems, not of the individual users of a particular filing system. Note that, since the number of distinct clients will be fairly small, the access control list could be represented in a compact manner (such as a bit map). The alternative approach, which we prefer, is to base the protection on capabilities. The analog in a distributed system of the hardware-protected capabilities of a centralized system is the use as our Inames of unique identifiers chosen from a sparse name space. Such identifiers can readily be constructed in such a manner as to have as low a probability of accidental or deliberate forgery as we desire. For example, we can construct one whose first field is one greater than that used for the previous unique identifier, and whose second field is selected by a random number generator having uniform distribution. Thus, the first field is intended to give the identifier its uniqueness, the second gives it its sparseness. For example, if the first field is 48 bits wide then it will not repeat within any conceivable lifetime of the system. If the second field is 48 bits wide, then the probability of a given 96-bit pattern being a valid internal name would be at most 1 in 2^{48}. By choosing a wider second field, this probability could be made as small as desired (and could, for example, be made less than the probability of undetected hardware error).

These alternative schemes have advantages and disadvantages analogous to those of the corresponding schemes in a centralized system; additionally, there are advantages and disadvantages peculiar to their use in a distributed system. One difficulty with the access control list scheme is determining the identity of a client. In some environments, the address of a computer will be sufficient to identify the client, but more commonly we will be concerned with which program is running in the computer, rather than the physical computer; additionally, in some cases we may have more than one client in a given computer. Such considerations would lead almost inevitably to some form of password protection, which is only a short step from the use of sparse unique identifiers. Even with an access control list system, we would probably wish to make use of unique identifiers (not necessarily sparse) as internal names in order to prevent a client accidentally reusing a name which the backing store server considers to have been freed and reissued (due, for example, to some programming error by the client).

One difficulty that might arise with use of the sparse unique identifier scheme is that it would be inconvenient to give different clients different access rights to a particular object (for example, read-only to one client but read-write to another). This could be achieved with a scheme of giving the clients different Inames for the same object, for example, by including access bits protected by an encryption process. However,

the complexities and confusions that would arise would be similar to using a different address for reading a word of memory from that used when writing to the same word. As with capabilities, sparse unique identifiers pose a problem of revocation, although as with capabilities this can in principle be solved by adding an indirection [4]. One distinct advantage of the sparse unique identifiers is that they provide a protected name which a client can happily give to, and accept from, his users.

The choice between these alternatives appears to depend on how elaborate one desires to make the protection at the level of the backing store server. If a simple on-off protection is thought sufficient, then sparse unique identifiers seem preferable; for more elaborate schemes, access control lists start to gain attractiveness. On balance, our preference at present is for sparse unique identifiers. We emphasize again that this protection is in addition to any access controls imposed by filing systems constructed using the backing store server.

V. General Considerations

If the technique of providing a low-level backing store server for use simultaneously by several clients and filing systems is to be successful, then there are two important considerations: the backing store server must provide sufficient facilities to allow the efficient implementation of the desired filing systems, and a filing system should preferably not have to pay, in increased complexity or reduced performance, for facilities of the backing store server which it does not use. It seems unlikely that we could design a single backing store server to suit all possible filing systems; the most we can expect is that it should be possible, in designing a particular backing store server, to make it acceptable to all of its clients. There are, however, some design considerations for the backing store server which have general applicability.

We proposed above, that the backing store server should provide a naming network as general as any we have seen in conventional filing systems. This will only be acceptable if the generality does not impose any expense when only a less general subset is used. It would seem that this criterion can be satisfied; the only additional expense incurred is if the asynchronous garbage collector runs, and it need only run if at some stage the reference count for an index is decremented to a nonzero value. This will never happen if, for example, only a tree-structured network is ever produced.

Another example of a desired facility might be to allow the use of demountable disk volumes. Again, this can be provided within the generality of our naming network if we adopt a scheme similar to that used by UNIX [3]. We need only apply the constraint that the Iname of an object on a demountable volume can be preserved only in an index which resides on the same disk volume as the object, with the exception that a *load volume* request will specify the Iname of an index in the new volume to be preserved in an index on an already mounted volume.

A third problem which might arise would be whether adequate arrangements can be made for filing system integrity and for backup, incremental dumping and similar facilities. It would seem that there is little difficulty in including these as facilities of the backing store server, common to all clients. Whether they could be implemented by an individual client if they were absent from the backing store server seems less likely, although we have not investigated this in detail. Similar remarks would apply to the provision of facilities for indivisible updates to an object.

Accounting is well-known to present problems in file systems with access via general naming networks. It is possible to devise algorithms which do roughly the right thing, but it seems that in a file server with the general purposes of ours they would be unnecessarily cumbersome. This is because we envisage the file server being used by a small number of primary clients, namely the filing system providers, which will be able to account among their own customers in a manner which will usually be less general and less complex. Furthermore, we expect that the filing systems will be mainly disjoint, in the sense that they will share only a small proportion of their files. Accordingly, we would propose that each file be charged to the filing system that initially created it. Within that filing system it can be charged for as the filing system designer chooses. If charging to the creating system is not appropriate in some cases of shared files, then it will be left to negotiation between the filing system managers. It would appear that the most convenient way to handle these matters is to have the file server require a capability for a charge site whenever a file is created. We would expect there to be one charging capability per major client.

We can see no way of finally resolving the question of whether we can design a backing store server which avoids precluding facilities desired by some of its clients, without causing undue expense to its other clients, other than to embark on detailed design, implementation and use of a particular backing store server. It is our belief that no insurmountable difficulties will be found in this area.

References

[1] R. M. Needham and A. D. Birrell, "The CAP filing system," presented at the 6th Symp. on Oper. Syst. Principles, 1977.
[2] A. D. Birrell and R. M. Needham, "An asynchronous garbage collector for the CAP filing system," *Oper. Syst. Rev.*, Apr. 1978.
[3] D. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *Commun. Ass. Comput. Mach.*, July 1974.
[4] D. D. Redell, "Naming and protection in extensible operating systems," M.I.T., Cambridge, Tech. Rep. MAC-TR-140, 1974.
[5] H. Dewar, V. Eachus, K. Humphry, and P. McLellan, "The filestore," Dep. Comput. Sci., Univ. Edinburgh, Oct. 1977.
[6] J. Israel, J. Mitchell, and H. Sturgis, "Separating data from function in a distributed file system," in *Proc. 2nd Int. Symp. Oper. Syst.*, IRIA, Rocquencourt, France, Oct. 1978.

A. D. Birrell, photograph and biography not available at the time of publication.

R. M. Needham, photograph and biography not available at the time of publication.