# The Virtual System Model for Large Distributed Operating Systems

B. Clifford Neuman

Department of Computer Science, FR-35
University of Washington
Seattle, Washington 98195

bcn@cs.washington.edu

Technical Report 89-01-07
April, 1989

**Abstract**

Scale is a critical element in distributed systems. The effect of scale on the user has received little attention. As the number of distinct computers that compose a distributed system grows, it becomes increasingly difficult for users to organize and find objects and services. Today's users are able to cope because they tend to use only a small number of computers, and because only a tiny portion of the objects and services in the world are easily accessible. This observation suggests one way of organizing large systems: let each user see a smaller system containing only those parts that are of interest. The virtual system model provides a framework within which users can build a view of a system in which the parts of interest are logically nearby. This report discusses the virtual system model and describes a global file system being built using the model.

## 1 Introduction

As the the size of a distributed system grows it becomes increasingly difficult for users to organize and find objects and services. Today's users are able to cope because only a tiny portion of the objects and services in the world are available to them. Users needn't be concerned with objects in which they have little interest. One way to help users cope with very large systems is to let each user see a smaller system containing only those parts that are of interest.

This report describes the *virtual system model*, a new model for organizing large distributed systems. The virtual system model supports the maintenance of customized views of the system. Each view is called a *virtual system*. The virtual system model has several advantages over existing organizations because: (1) as systems become larger and larger, a single view containing all objects in the system becomes difficult to work with and multiple views are needed; (2) the maintenance of multiple views and the ability to include parts from others allows objects to be organized in a manner that

This research was supported in part by the National Science Foundation under Grants No. DCR-8420945 and CCR-8611390.

allows them to be more easily found; (3) the virtual system model allows users to easily control the selection of servers when a particular service is desired; (4) by associating virtual systems with objects to be protected, the mechanisms by which protection is provided can be better tailored to the owner's security requirements; (5) by associating virtual systems with processes and applications to be run, the environment in which they are run can be controlled and access to objects outside that environment can be restricted (if desired); and (6) since virtual systems include the list of the processors which may be used, they provide a mechanism to bind programs to particular processors.

In looking at very large systems, three aspects must be considered. The first is the access mechanism: how an object is accessed once it has been found. The second aspect is object location: given an object's name, how the object is found. The third aspect is organization: how users see the object hierarchy.

In addressing these issues, it was decided that the following characteristics should be supported: (1) users should gain access to local and remote objects in the same way, though the underlying access mechanism might be different; (2) the name of an object should not depend on the location from which is it accessed, and an object's name should not constrain its location; (3) each user should be presented with a view of the system in which those parts that are not of interest are hidden; at the same time, (4) users must be able to access objects outside of their view once learned about through other means; (5) users should be able to customize their views of the system, not just by naming hierarchies that they import into their namespaces, but by customizing the imported hierarchies as well; and finally, (6) any solution must extend beyond the file system to allow the tailoring of users' views of other system components as well.

The virtual system model provides support for the construction of customized views by users and groups of users. The naming and access mechanisms that support these views satisfy the characteristics just mentioned. The virtual system is the collection of system components included in a customized view. Each user might have his or her own virtual system, though it might start out as a copy of some prototype. Individuals with multiple roles can construct overlapping virtual systems, each corresponding to a different role. Projects or other entities that might benefit from a separate view of the world can have associated virtual systems. Thus, a project might have a virtual system to which users interested in the project might connect. The programmers on that project, however, will probably each have their own virtual system, each a superset of that for the project.

The virtual system model applies to many services. Among them are the file system, naming, authentication, authorization, and the use of processors, services and applications. Because of the large number of files on even small collections of systems, the file system provides a natural example and a focus for a prototype implementation to test the ideas of the model. It is described in Section 3. Sections 4 through 7 discuss the other services affected by the model. In Sections 8 and 9 other aspects of the model are discussed and conclusions are drawn.

## 2 Shortcomings of existing approaches

Many existing systems provide access to files scattered across large collections of computers. These systems will be described in the context of the three issues discussed earlier: the access mechanism, object location, and organization.

Early systems, such as the ARPAnet, addressed few of these issues. Access to local objects was simple, but access to remote objects required one to first locate the object, and then to access it using a mechanism different from that used in the local case.

Systems such as NFS[13] address the access mechanism, but little else. In NFS, remote file systems can be mounted locally and the same mechanism can be used for both local and remote file access. It is still necessary, however, to locate the desired files from the universe of accessible files, and to mount the appropriate file systems. This two-step process makes it cumbersome to maintain links between individual files on different file systems, and imposes constraints on the placement of files and the organization of the file system.

Systems such as Andrew[4, 6], Locus[12, 16] and Sprite[11, 17] address the organization problem by attempting to present a single view of a system that extends across the users and sites that compose it. Such an approach follows from the goal of name transparency and source-location-independence, i.e., the name of an object should be independent of the location from which the reference is made. The object location mechanism in these three systems is based on looking in a table for a prefix of the pathname. Although the object location is not specified by the pathname, objects with pathnames sharing a common prefix (as found in the prefix table) must reside on the same file system. This makes it difficult to truly distribute similar information over a number of hosts, especially when a particular file should logically appear at multiple points in the file hierarchy.

The organizational mechanism becomes more important as the number of objects that compose the system grows. Users have a harder time trying to find things. If a system included every processor, file, service and user in the world, users of that system would be overwhelmed by the information that was available. Finding useful information could become impossible. One way to address this problem is to let each user see a smaller system containing only those parts that are of interest.

QuickSilver[3] applies this approach to the file system by supporting user-centered namespaces. For every user, a nameserver stores a list of directory prefixes along with the information needed to locate a user's files. Users can only access files that have been included in their namespaces. Several types of links allow files and directories from one user's namespace to be included in another's. By updating their prefix table and links, users are able to tailor their namespace, making it easier to find and name the objects they frequently use.

QuickSilver does not completely solve the location and organizational problems of large systems. A file's location is still not completely independent of its primary pathname. In fact, the existence of a primary pathname leads to a shortcoming with links: the source and the target of the link are not equivalent [1]; if the target is later deleted, the file is gone. Another shortcoming is that none of the

[1] Except for hard links, the effect of which (in QuickSilver) is more like making a copy of a file.

systems described so far allow the user to tailor the namespace below the level at which a directory has been included by a link or prefix table entry. Finally, QuickSilver provides little support for finding and organizing files that might be of interest, but which have not yet been included in the user's namespace.

Directory mechanisms in capability-based distributed systems such as Eden[1] and Amoeba[7] support the naming of all objects, not just files. Directories are objects that map from names of objects to the capabilities for the named objects. Capabilities correspond to links in other directory systems. Capability-based systems address many of the problems that arise in the other systems discussed in this section. They support user-centered naming in the sense that each user can start with a different set of capabilities, and can organize them as desired. Ignoring access rights, all capabilities for an object are equivalent. Removing a capability from one directory does not leave the others dangling. Finally, a capability-based system does not place constraints on an object's location. Objects whose capabilities appear in a single directory can be scattered across many sites.

Like the other systems described in this section, neither Eden nor Amoeba extend the user's ability to tailor the namespace to lower levels of the hierarchy. Capability-based systems also have problems of their own. Capabilities have two purposes: they allow one to find the object, and they provide access rights. While this is not a problem in itself, it does mean that giving away access to a directory also grants access to the objects in that directory. This can discourage users from making their directories available to others.[2] What is needed is a mechanism that allows one to share one's organization of a collection of objects without also sharing one's access rights.

Before proceeding to a more detailed discussion of a file system based on the virtual system model, the relative merits of a customized view must be discussed. Does it make life easier or harder for the user? An objection to systems like QuickSilver is that names are not unique. The same name might refer to different objects when specified by different users. Relative names might be easier to remember, but they are harder to share. The virtual system model avoids this problem by allowing one to specify names of the form virtual_system:filename. Such names refer to the same object from all virtual systems and make sharing easier. The virtual system model makes life easier for the user by reducing the amount of clutter that must be searched when looking for objects of interest. The ability for an object to appear at multiple points in multiple virtual systems also makes it less likely that an object of interest will be missed. The next section describes a file system based on the virtual system model and discusses ways that it can be used to organize information.

[2]One way around this problem requires maintaining two separate hierarchies. One contains one's own rights, and the other, the rights to be granted to others. Unfortunately, this approach requires additional maintenance. New objects must be added in both places. A second approach involves giving out a restricted directory capability that causes the directory service to restrict the rights before returning capabilities for additional objects. Since rights are type specific, this doesn't work when a directory contains objects of different types. It also restricts one's ability to grant greater rights for some objects than for others.

# 3 The Virtual File System

A file system is currently under construction to test the basic ideas of the virtual system model. A prototype is presently running at the University of Washington and on several sites in another part of the Internet. The file system has two parts: a directory mechanism and an access mechanism. The virtual system model primarily affects the directory mechanism. Multiple access mechanisms will ultimately be supported because differing parameters such as latency, bandwidth, and access patterns will place different constraints on the access mechanism at different times. Once a file has been found, an appropriate access mechanism will be chosen.

The *global file system* is a collection of *virtual file systems*, each associated with one or more virtual systems. Each virtual file system has a root and appears hierarchical, but it does not fully satisfy the constraints for a hierarchy (loops are allowed). Items can appear multiple times within a file system, as well as in multiple file systems. Objects will be found, and virtual file systems built up by users as they learn of objects outside their current environment.

The remainder of this section briefly describes the global file system that is being built and discusses ways to organize objects within it. Additional information about this file system can be found in [8].

## 3.1 Directory Service

A significant component of the global file system is the directory service. Directories map from names to object pointers. In the file system the objects are files and directories. Because the directory service can be used to name objects other than files and directories, and because the objects included in a directory do not need to be located on the same system as the directory, the directory service is actually much closer to directory services in object-based systems such as Amoeba[7] than to directories in traditional file systems.

Figure 1 shows parts of two virtual systems. Directories 1 and 2 are the roots for bcn's and lazowska's systems respectively. Only those files and directories that are useful in the discussion that follows are included in the diagram. The actual virtual systems are really much larger.

Each virtual file system has a root directory, and the files and directories that are descendants of the root directory appear to form a hierarchy. In the global file system, however, files are not organized completely hierarchically. Files may exist multiple times in the hierarchy[3] , and it is quite acceptable to have directory links that form loops. It is expected that different virtual file systems will overlap and that files and directories will appear in multiple virtual file systems. It is likely that some files will appear in almost all systems, though they may be very deep in the hierarchy, if they are not of particular interest to the user. Those files in which the user is interested will be much closer to the root. It is in this way that the user's virtual file system appears to be

[3]In Figure 1 "this paper" is found in bcn's virtual system with the names /bcn/papers/virt-system-model, /bcn/vsm/paper, and in /authors/Neuman/virt-system-model. In lazowska's virtual system, it can be found with the names /authors/Neuman/virt-system-model, /users/bcn/papers/virtual-system-model, /users/bcn/vsm/paper, and /subjects/dist-systems/virtual-system.

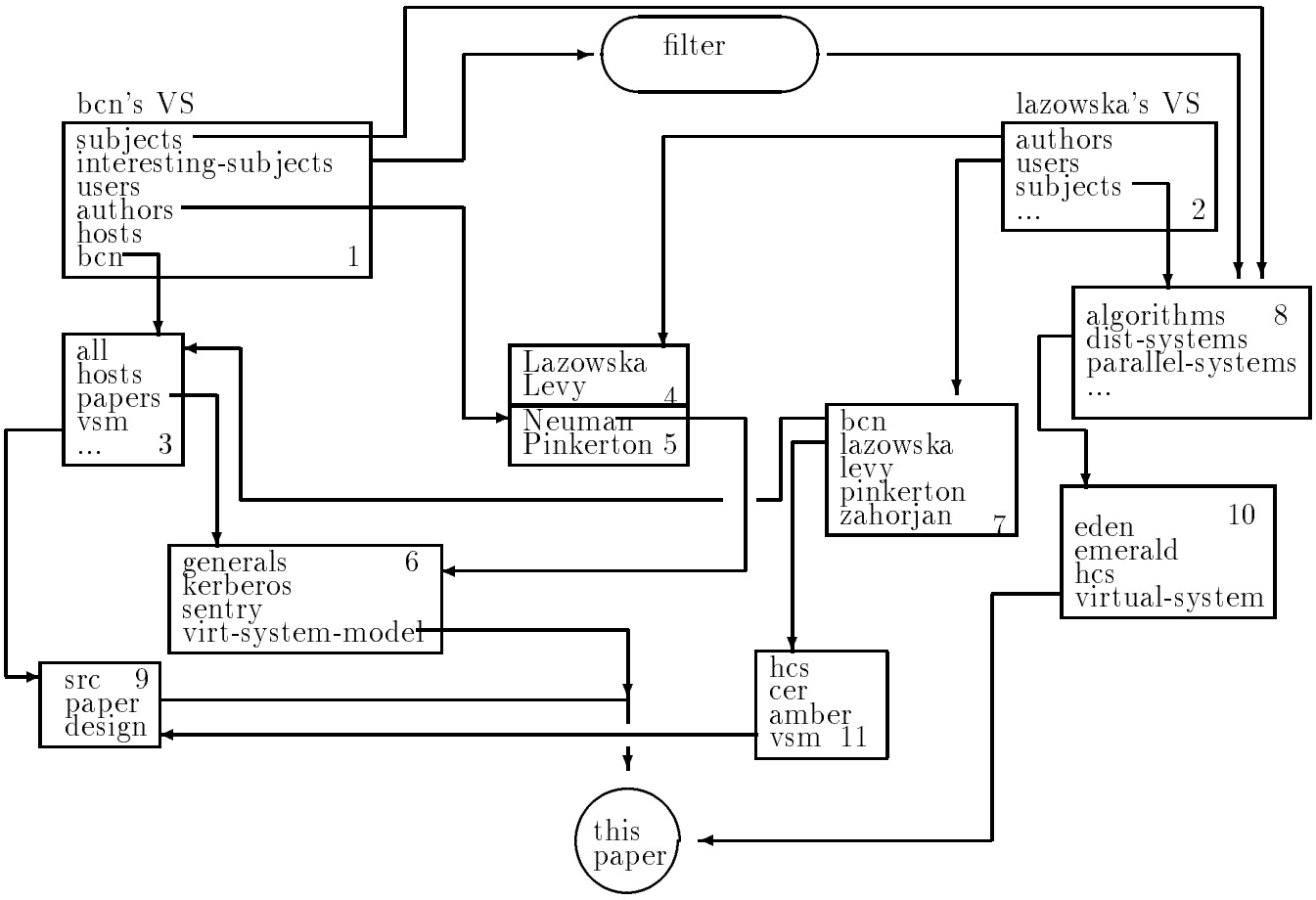


Figure 1: Parts of Two Virtual Systems

customized.

Information about individual files will be stored along with the files. Among this information is a (possibly incomplete) list of back links. These links are pointers to the directories which contain links to the file. This information can be used to find other information which is similar or related to that in a particular file.

Other information that helps users find what they are interested in may be stored along with the directory link. One such piece of information is a filter. A filter is a program and associated data that can be used to decide whether a file should be included when a directory is listed. If a link to a directory has an associated filter, then only those files accepted by the filter will be displayed. A filter can do more than just approve or disapprove directory entries. It can transform names, or perhaps even insert additional entries obtained through some other mechanisms besides simply reading the linked directory hierarchy.

In addition to traditional links, the global file system supports union links [4]. The files in a directory that is linked by a union link appear as if they are part of the directory making the link instead of as a subdirectory. The order in which union links are made is significant. In the case of name conflicts, it is the first link with a given name that is taken. The union link can be used in conjunction with filters to provide a very powerful mechanism for reorganizing file systems for a particular user or purpose.

Links to files are equivalent. There is not a primary or "real" name for a file. If a file is created, and someone else makes a link to it, then the file is deleted, the file still exists with the name of the link. The equivalence of links presents a number of problems in the area of file owership, garbage collection, and what happens when a file with multiple links is modified. These issues are addressed in [8].

Information about files and directories is accessible through directory servers that run on each host that is part of the global file system. All directory queries are directed to these servers. The local format of directories and other information is hidden from the user by the servers. Servers can store directories and other information in the native format for the system on which they reside, or they can use their own format. Information required for the global file system which is not part of the native format can be stored in supplemental (shadow) files.

Files which were not created as part of the global file system are easy to incorporate. Directory links may be made to files created through a system's native file system as long as a directory server runs on the host containing the files. Answers to requests for information about such files will be the value from the native file system where appropriate, a derivative value, where a mapping exists, a default value, or an indication that the information is not available.

### 3.2 File Storage and Access

Files are stored in the native format for the system on which they physically reside. Because native file systems do not maintain all the information needed by the global file system, the directory server on the custodial host must maintain the additional information itself. Some of this information is used for replication and access control. Other information includes a reference to the owning virtual system and a partial list of back links to the directories that have links to the file.

Files are accessed using existing remote file access protocols. In the initial implementation, Sun's Network File System is used. The directory mechanism maps from names in the global file system to the custodial host and name of the file on that host. That information can be used by existing file access mechanisms to access the desired file.

[4] Union links are based on the idea of a union mount proposed by Rob Pike at the first Workshop on Workstation Operating Systems in Cambridge, Massachusetts, November, 1987. A Union mount allows one to mount multiple file systems on a single mountpoint, the result being the union of the files in the multiple file systems.

## 3.3 Access Control

Control of access to the actual files stored in the file system is handled by the underlying file access mechanism. Many remote file access mechanisms available today do not provide adequate security. In such cases, added security is desirable, but beyond the scope of this discussion. Such security mechanisms may make use of the supplemental information (such as an access control list) stored along with the file.

Access control must also be applied to information in the directory servers. Access control information may optionally be associated with individual directory links. Queries can include an optional authenticator that can be checked by the directory server to decide how to respond to the query.

## 3.4 How the Global File System Can be Used

This part presents examples of how a global file system based on the virtual system model might be used to organize and find information in a large distributed system. This is not an exclusive list.

### 3.4.1 Personal Organization

Users will build their own hierarchies of files by creating directories, subdirectories, and files of their own, and adding links to files, directories and subdirectories created by others. Files that are frequently accessed by a user will probably have short names from the root of their virtual system. Since directories of others (and hence, whole hierarchies) can be added to a user's virtual system, the system will probably contain files that a user has never accessed and might not even know about. These files, however, will be much deeper in the hierarchy.

Directory 1 in Figure 1 is the root for bcn's view of the world. His home directory, however, is the subdirectory "bcn". In this organization, subdirectories of "/bcn" contain files and directories of a personal interest, whereas the remainder of the root directory forms his view of the rest of the world. Other users might choose to organize their system differently. For example, one might include those files and directories frequently of personal interest in the root with one's view of the rest of the world relagated to a subdirectory.

When users link other directories to their own hierarchy, it will be possible to add information to the link to specify how much of the merged hierarchy is to be included. It will also be possible to customize parts of the attached hierarchy. An example of this is seen in the subdirectory "interesting-subjects" of directory 1 (Figure 1). This filter uses information on who else has links to files to decide which of the files in subdirectories of directory 8 should be listed.

Each directory and file that a user maintains will be owned by that user. Parts of a user's hierarchy, however, may be owned by other users. Access control information will be maintained along with each file or directory, and with each directory link, and will determine who is allowed to read the

file or search the directory. It is expected that users will make parts of their hierarchies accessible to others, but how much will be decided by the individual.

### 3.4.2 Project Organization

Just as users will each have their own virtual system, it is expected that projects will have separate virtual systems too. It is likely that everything that is part of a project's virtual system will also be part of the virtual system of at least one member of the project.

Directory 9 in Figure 1 shows the project directory for the virtual system project. In addition to being included in the virtual systems of those involved with the project, it could also be the root of a virtual system of its own. A project virtual system serves several roles. It is a prototype virtual system that can be given to, or merged with that of new users when they become part of the project. It also provides a starting point from which those wanting information about a project can look. Finally, it provides a logically central starting point from which everything related to a project should be accessible.

### 3.4.3 Index by Author

It is expected that users would maintain a single directory containing papers of theirs which they consider published, and which they want others to be able to access. A service wanting to provide an index of published papers by author would simply set up links from its "author" directory to these directories in each authors hierarchy. By doing this, the indexing service only needs to make updates when a new author needs to be added. Today, library card catalogs provide indices by author. In the future, libraries could be one of the providers of indices by author.

In Figure 1, directories 4 and 5 form an index by author. Both bcn and lazowska maintain their own indices, but by agreement, they decided to distribute the work of maintaining it. Each directory includes the other using a union link. Figures 2a and 2b list the contents of directories 4 and 5 without expanding union links. Anyone with a link to either of these directories will see the contents listed in Figure 2c.

### 3.4.4 Indices by Topic

Just as users might maintain a directory of their own papers, they might also maintain directories containing links to interesting files or directories on a particular topic. Look at directory 10 in Figure 1 for an example of such a directory. It contains links to papers on distributed systems. Other user might maintain their own list of papers on distributed systems, but with a different emphasis. If a number of people in a field decide that they each trust the views of the others, then they might collaborate to provide a directory whose contents are the union of each of their directories. Users with an interest in a particular topic can use any of the collections that are readable, or the union of several of them.

```
   Neuman,Clifford      JUNE.CS.WASHING /u1/bcn/papers
   Pinkerton,Brian      JUNE.CS.WASHING /u1/bp/Papers
U                       KRAKATOA.CS.WAS /u1/lazowska/authors
```

Figure 2a: Listing of bcn:/authors without expanding union links

```
   Lazowska,Edward      KRAKATOA.CS.WAS /u1/lazowska/papers
   Levy,Henry           WHISTLER.CS.WAS /u1/levy/papers
U                       JUNE.CS.WASHING /u1/bcn/vfs/bcn/authors
```

Figure 2b: Listing of lazowska:/authors without expanding union links

```
   Lazowska,Edward      KRAKATOA.CS.WAS /u1/lazowska/papers
   Levy,Henry           WHISTLER.CS.WAS /u1/levy/papers
   Neuman,Clifford      JUNE.CS.WASHING /u1/bcn/papers
   Pinkerton,Brian      JUNE.CS.WASHING /u1/bp/Papers
```

Figure 2c: Listing of bcn:/authors, union links expanded

As with the author index, it is likely that master indices will be maintained by libraries or other organization. These indices will allow one to find the directories containing information on a particular topic. Directory 8 is an example of such a master index. There are likely to be multiple central indices, each of which might compete for use based on their own strengths such as how complete their index is, how accurate it is, or how much junk you get back when using the index of a competitor.

### 3.4.5 Browsing

One way that information can be found using a file system organized with the virtual system model is through browsing. If you are interested in a particular topic, you can connect to the virtual system of someone who you know is also interested in that topic, or perhaps to the virtual system for a related project. You could then look through those virtual systems for documents or files of interest. Of course, you would only see those files that the owner of the virtual system has authorized you to see. The inability to see some information, though, hopefully tends to weed out information you would not be interested in anyway.

In order for browsing to work best, virtual systems owned by projects should include a directory containing links to related work. Users would maintain directories containing pointers to files that they consider interesting or relevant. Files in which others would (or should) have little interest should be kept from their view by applying appropriate protections.

Browsing is considerably more likely to be effective under the virtual system model than on more traditional file systems. The virtual system model encourages users to make their own links to the files in which they have an interest. As such, interesting files are likely to appear in the hierarchies of multiple people, thus increasing the likelihood that they will be found by browsers.

### 3.4.6 Finding Things

In society today, if something is available that is of interest, it is usually found through directories such as the phone book or yellow pages, through reading newspapers and other periodicals, or by word of mouth. In the computer science community, these sources of information are supplemented by technical papers, electronic mail and mailing lists. These methods of discovery are natural, and it is likely that they will continue to find significant use even once other mechanisms are in place.

The virtual system model allows much of this information which might be useful in finding objects, but which to date could only be obtained by external means (such as asking the appropriate person) to be included as part of the file system. This information provides a matrix through which users can navigate to find the desired information. Services might even spring up to help users navigate through this matrix. An example of such a service is described in [14] and makes use of resource discovery agents. These agents accept queries from users and use the information provided by the user to find objects in which the user is interested. The multitude of links in a system based on the virtual system model can provide the information needed to direct such searches.

## 3.5 Comparison to Existing File Systems

In Section 2, certain shortcomings of existing approaches were discussed. The file systems considered were NFS, Andrew, Sprite, Locus, and QuickSilver. Here these approaches are explicitly contrasted to the global file system built using the virtual system model.

In all five systems, the mechanism used to find files places constraints on their location. This is the case even though the storage site is not part of the pathname. The granularity of distribution is at the level of individual file systems instead of individual files. All files in the same part of the file hierarchy will be located at the same storage site. In the global file system, distribution to storage sites is done on a file by file basis, and this constraint is avoided.

All five systems provide at least a limited level of customization. This customization is primarily the result of using symbolic links relative to one's home directory. With the exception of QuickSilver, links in these systems are not equivalent to the primary pathname for the files. Moving or deleting the file or directory that is the target of the link leaves a link that no longer works. Several types of links are supported by QuickSilver, though none are equivalent to the primary pathname for the file in all cases [5]. Links in QuickSilver do, however, solve some of the reference problems just described. In the global file system, there is no primary pathname, all links to files are equivalent, and there is no problem with dangling references.

[5] Hard links result in a copy of the file being made if the link crosses a machine boundary.

None of the five systems provide the tools necessary to make links as useful as possible. Customization by links is limited to the name of the link, and more detailed customization of the "renamed" hierarchy is not possible. In the global file system, filters and union links allow one to further customize the linked hierarchies.

NFS allows file systems to be mounted at arbitrary points on the local system. This provides a mechanism for customization on a system by system level, since each system might mount remote partitions on different mountpoints. Andrew, Sprite, and Locus each provide a single global namespace within which all files are named. They avoid the possible confusion that arises from having different namespaces on different physical processors. Both QuickSilver, and the global file system support customization on a user by user level. This customization is independent of the physical processor being used.

QuickSilver addresses customization through the user-centered namespace. It enforces use of the customized view since the only way to access the files or directories of others is through the links and updated prefix table entries customizing the namespace. This differs from the global file system which allows the virtual system name to be explicitly specified, thus allowing access to external objects.

Compared to NFS, Andrew, Locus, Sprite, and QuickSilver, the global file system provides tools allowing a significantly greater level of customization of the user's view of the file system. The union link in conjunction with active filters allow lower levels of an included hierarchy to be customized. These tools, together with a directory organization that does not tie files or directories to particular hosts allows users to organize files in a manner that best suits their own needs, and the needs of those who need to find the files.

# 4 Naming

So far, only the file system has been described. The virtual system model extends to other system components as well. The system component most closely related to the file system is naming. The directory service described in the last section can be used to name more than just files and directories. It can be used to resolve user-assigned names for services, other users, hosts, files, and even other virtual systems. Objects may have multiple names, and across different virtual systems, the same name can refer to different objects. Within a single virtual system, however, a name can only refer to a single object [6]. Thus, there exists a many-to-one mapping from *virtual_system:name_within_that_system* to any object that is part of the global system.

The directory service provides a mechanism to resolve user-level names within a virtual system. User-level names resolve to globally unique system-level names which can be used to identify, and ultimately to locate the object. Services resolve to the host on which the service is provided and the name or number of the port to which one must connect. Filenames resolve to the custodial host, and the name of the file on that host. In both of these cases, the system-level hostname can

[6] If objects are replicated, the set of replicas can be thought of as a single object.

be further resolved using existing hierarchical nameservices such as the Internet Domain Naming System[15], or DEC's global naming system[5].

# 5 Authentication & Authorization

Authentication is used to make sure that an entity claiming to have a particular name is in fact, the entity to which that name refers. Authorization is used to make sure that a named entity is allowed to perform a particular operation. For both to work, there must be an accepted mapping from a set of names to entities requiring authentication. Although this mapping does not need to be the same in all places, authorization is significantly simplified if it is. Since virtual systems overlap, allowing the use of non-unique user-level names for authentication could create parts of the system where two different sets of names are in use. This would complicate access control for objects in those overlapping regions. For this reason, authentication and authorization will be based on the globally unique names of users. This does not preclude having an additional level of indirection through which users specify names.

It is expected that authorization will be accomplished primarily through access control lists. The concept of groups is necessary for access control lists to provide a flexible interface for access control. Groups can contain users or other groups. As with names, groups should be represented uniquely within access control lists. Users, though, might use different names to identify them. A problem arises in deciding how to display the content of ACLs to the user. With groups, it is perhaps even more important for the user to see a name with which he is familiar. Mapping back from the unique name to a local name might be costly, however.

One solution is to give the user the option of how he wants access control lists displayed. The average user would probably choose the local names. Since average users might only have a small number of users and groups defined within their virtual system, the mapping might not take as long as for a more advanced user. In any case, information is needed to help users figure out what users and groups are if the groups do not map to local names, or for which local names do not exist. Each group can have a comment associated with it. These comments can be displayed next to the local or unique name when the ACL is listed.

So far, the naming side of authentication has been discussed, and how these names are used for access control. Authentication itself has not been talked about. One of the problems that arises as a system spans multiple organizations is lack of trust. Services can't be required to trust every authentication server that exists. It might not even be the case that there is a single authentication server trusted by everyone. A particularly paranoid application might only trust authentication performed solely by its local authentication server.

Another issue concerned with authentication is the distribution of authority. Unless, for every communicating pair of entities, there exists some trusted entity sharing a secret with both parties, then multiple authentication servers might be involved in authenticating a principal.[7] It is important

[7] Even with public key systems, the public key to party A must be known to a key repository whose public key is known by B.

that the end service know which authentication servers were involved when deciding whether to trust the authentication. More on this can be found in [2].

There may be different competing collections of authentication servers. Such services might differ in who they are run by, or perhaps even in the protocol required for initial authentication when the user logs on. It should be possible, from within each virtual system, to decide which authentication services one is willing to trust. By attaching an "owning" virtual system to each object to be protected, this choice of who to trust is passed along to individual objects.

An area related to authentication and access control that has received little attention is accounting. It is related in the sense that once one's quota for a service has been used up, or when one has run out of money to pay for a service, access might be denied. One also doesn't want to get billed for a service that was used by someone else. Accounting will become more important in a system like the one being described because the system provides the connection between service providers and consumers that may not be within the same organization. Services analogous to banks will be required, and a mechanism to allow users to authorize their bank to pay a server will become part of the access protocol for certain services. See [7, 9] for related discussion.

# 6 Processes and Processors

Each virtual system will be associated with zero or more processors. These will be the processors on which applications will run. Processors within a single virtual system should be able to be of different types, and the processor chosen should be based on the requirements of the application as well as other information available at the time (such as load, etc). Applications should be able to choose processors from within the virtual system on which they are running, within which they were installed, or, alternatively, on other processors that they find dynamically.

When a process or application is attached to a virtual system, a closure is formed. A closure is code to be executed and the set of bindings seen by a process. Just as an attached virtual system can be used to restrict the processors on which an application runs, it can also be used to restrict the objects that can be accessed. This can be accomplished by setting a flag indicating that processes running within a particular virtual system should only be able to access objects that are part of that virtual system, and that the ability to switch to other virtual systems should be disabled. By building a virtual system with limited size, and by protecting the virtual system so that it can't be modified from within, processes can be run in a protected environment. In order for this to work, the system on which the process executes must enforce the restrictions imposed by the virtual system. By suitably restricting what a process can read and write, the virtual system model can provide support for multi-level secure systems.

# 7 Services and Applications

In existing systems, services and applications differ in the way they are accessed. A service typically provides basic operations and is usually accessed across a network. An application is typically a local program, but it may provide a front end for one or more services. In the virtual system model applications can be associated with a set of processors on which they may run. When a user runs an application, it might run remotely. The distinction between an application and a service will be more a matter of whether the interface is intended for humans, or for other programs.

Given the name of a service or application, clients will use the directory service to find the server (or executable). If multiple instances exist, the client will have to make a choice. The choice is made by including the selected server or set of servers in one's virtual system. The use of filters on links in the directory service provides a useful tool for choosing a server. A link can be made to a directory containing pointers to instances of a service. A filter associated with the link can be used to filter out all instances that do not satisfy the client's constraints (e.g. price).

# 8 Autonomy

The physical systems that are tied together by the virtual system model are autonomous. Each can operate independently from the others. Naturally, one would not be able to access objects or services which reside on a system that is unreachable due to network or hardware problems. Availability can be enhanced by replication, though access to replicated objects can impose its own overhead and constraints.

The user level naming mechanism has no central database, and no critical nodes. The user level naming mechanism maps names to addresses which are really system level names. In some cases, resolving the system level names might require access to critical sites. Attempts will be made to minimize such dependencies.

# 9 Conclusions

The virtual system model provides a framework within which users can build a customized view of the objects and services available in a large distributed system. The approach described in this report exhibits all of the characteristics mentioned in the introduction: to the user, local and remote objects are accessed the same way; the name of an object depends neither on the object's location, nor the location of the user accessing it; users are able to customize their views of the system; and the approach extends to all components of the system, not just the file system.

Filters and union links are two important features of the directory service on which the the virtual system model is built. These tools allow a user's view to apply to new objects in the system as they are created. Those and other features of the directory service allow objects to be organized

in a manner that will allow them to be found in a very large system.

The virtual system model extends to naming, authentication, and authorization, to the use of network services, and to the running of application programs. This ability to adopt different views in these areas is particularly important in an environment that consists of many organizations where users need control over the services that are trusted, those with which one does business, and the processors on which applications can run.

A file system based on the virtual system model is presently under construction and a prototype is running on several sites scattered across the Internet. As experience is gained, and as other pieces of the system are put in place, changes are expected in the way people think about and use distributed computer systems. The virtual system model can provide the basis for a global operating system that more closely parallels the way people interact in society.

## Acknowledgements

Ed Lazowska, Hank Levy, David Notkin, and John Zahorjan helped me to refine the ideas presented in this report. Helpful comments on earlier drafts were also received from Brian Pinkerton, Terry Gray and Kathy Faust. Ed Lazowska deserves special thanks for the numerous discussions out of which many of the ideas emerged. Some of the ideas emerged from discussions with Alfred Spector and other instructors at the Arctic '88 Course on Distributed Systems.

## References

[1] G. T. Almes, A. P. Black, E. D. Lazowska, and J. D. Noe. The Eden system: A technical review. *IEEE Transactions on Software Engineering*, 11(1):43–59, January 1985.

[2] Andrew D. Birrell, Butler W. Lampson, Roger M. Needham, and Michael D. Schroeder. A global authentication service without global trust. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 223–230, April 1986.

[3] Luis-Felipe Cabrera and Jim Wyllie. QuickSilver distributed file services: An architecture for horizontal growth. In *Proceedings of the 2nd IEEE Conference on Computer Workstations*, pages 23–27, March 1988. Also IBM Research Report RJ 5578, April 1987.

[4] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[5] Butler W. Lampson. Designing a global name service. In *Proceeding of the ACM Symposium on Principles of Distributed Computing*, August 1985.

[6] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal, and F. Donelson Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[7] S. J. Mullender and A. S. Tanenbaum. The design of a capability-based distributed operating system. *The Computer Journal*, 29(4):289–299, 1986.

[8] B. Clifford Neuman. The Prospero File System: A global file system based on the Virtual System Model. Department of Computer Science, University of Washington. In Preparation.

[9] B. Clifford Neuman. Sentry: A discretionary access contol server. Bachelor's Thesis, Massachusetts Institute of Technology, June 1985.

[10] B. Clifford Neuman. Issues of scale in large distributed operating systems. Generals Report, Department of Computer Science, University of Washington, May 1988.

[11] John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson, and Brent B. Welch. The Sprite network operating system. *Computer*, 21(2):23–35, February 1988.

[12] G. Popek and B. Walker, editors. *The Locus Distributed System Architecture.* M.I.T. Press, Cambridge, Massachusetts, 1985.

[13] R. Sandberg et al. Design and implementation of the Sun network file system. In *Proceedings of the Summer 1985 Usenix Conference*, pages 119–130, June 1985.

[14] M. F. Schwartz. The networked resource discovery project. In *Proceedings of the IFIP XI World Congress*, pages 827–832, August 1989. San Francisco.

[15] Douglas B. Terry, Mark Painter, David W. Riggle, and Songnian Zhou. The Berkeley internet domain server. In *Proceedings of the 1984 Usenix Summer Conference*, pages 23–31, June 1984.

[16] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The Locus distributed operating system. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, pages 49–70, October 1983.

[17] Brent B. Welch and John K. Ousterhout. Prefix tables: A simple mechanism for locating files in a distributed system. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 184–189, May 1986.