Project MAC
Computer Systems Research Division

February 4, 1975
Request for Comments No. 67

A NETWORK-WIDE FILE SYSTEM FOR MULTICS

by David D. Clark

INTRODUCTION

There exists a network protocol whose function is to allow programs on one machine to access the file system of another machine through the network. This protocol, called RSEXEC, was developed by Bob Thomas at BBN, and is currently being used to allow the various TENEX systems on the network to interact. This memo describes one way to make the RSEXEC protocol applicable to the Multics system as well. Using the implementation described here, a Multics user would be able to access files on another system as follows: He would be able to create a link from a directory on his system to a segment on a foreign system, he would be able to set his working directory to be a directory on a foreign system, and he would be able to add directories on foreign systems to his search rules.

THE RSEXEC PROTOCOL

RSEXEC is a user level protocol designed for communications between systems. It differs from other user level protocols, such as TELNET or file transfer protocol, in that it is not intended to be used directly by people, but is rather intended for communication between processes. The information transmitted back and forth between systems is in general not strings of ASCII characters, but rather binary data structures. When a user indicates his intention to reference files on a foreign system, RSEXEC logs in on that foreign system a "helper" process, which will transmit back to the local system status about files or, as appropriate, the files themselves. The RSEXEC protocol is used for communication between these various helper processes, and the main process of the user at his local site. The RSEXEC protocol essentially defines the number of commands which the main process may request the helper process to execute on its local system. The following list identifies the most important commands, using the equivalent Multics command name where possible:

login	(to create the helper process)
who	
send_message	
list	
status	
rename	
delete	
change_wdir	
copy	(local to the foreign system)
concatenate	(two files on the foreign system)
copy	(between two systems)

COMMAND ENVIRONMENT SEEN BY A USER OF RSEXEC

The intention of the RSEXEC scheme is that the above named commands shall be used by a subsystem running on the user's local system to provide an environment which allows sharing of files between systems, and at the same time imitates closely the normal command environment seen by a user of his own system. Thus, while the protocol defines a standard interface between systems, it is expected that implementation of the RSEXEC environment on differing systems need have little resemblance to each other. The most important aspect of the RSEXEC implementation on a particular system is the fashion in which it allows the user to refer to files on a foreign system. The TENEX implementation of RSEXEC (the only currently existing implementation) represents files on a different system by amalgamating a directory on a local system with directories on one or more foreign systems to create a "big working directory in the sky", which the user then references as one directory. This approach seems inappropriate on Multics, where the user is already familiar with the idea of a world containing many directories. A better representation of foreign file systems to the user on Multics would be the model of a network-wide hierarchy in which the user may reference directories as if they were all on his local system.

The simplest and most direct way of referencing a segment in some other part of the hierarchy is to initiate it, using an explicit pathname. Let us consider what modifications would be required to the initiate routine in order to allow this across the network. First, we must have some syntax for constructing a pathname which describes a segment lying in that portion of the hierarchy which is in some other system. For example, if a pathname begins with the character string ">arpanet>site" the first components of the name would identify the system on which the segment existed, and the rest of the pathname would identify the segment using whatever syntax was appropriate for that system (*). With such a scheme, the initiate routine could determine whether the segment in question existed on this system or on some other, and could take appropriate action. If the segment exists on some foreign system, the appropriate action is to make a copy of the segment on the local system, since an attempt to initiate a segment is usually followed by an attempt to touch the segment. Therefore what the initiate routine should do if the pathname identifies a segment on a foreign system is to copy the segment to someplace in the hierarchy on the local system, and then initiate that segment.

The change to the initiate routine just described currently represents a modification to the Ring 0 portion of the system. Notice, however, that if the restructuring of the KST currently proposed by Dick Bratt were installed, this change would represent only modification to user ring modules. Because of the experimental nature of the RSEXEC protocol, it would seem inappropriate to implement the ideas described here unless Dick Bratt's schemes are first made a part of the standard Multics System.

Given this modification to initiate, little else is required to allow a user to link to a segment of a different system. In Dick Bratt's scheme, the only function which directory control performs for a directory link is to serve as the repository of the character string which defines that link; the interpretation of that character string is done in the user ring. The modification to the initiate routine has already been described which will allow it to interpret a path-name referring to a segment of another system. Thus no further modification is required to implement cross-system links.

It is equally clear that changing one's working directory so that it is located in a distant system is not especially difficult either. When one sets one's working directory, what one is really doing is saving a character string which is used under certain circumstances as an argument to the initiate routine. Since the initiate routine, as modified above, will accept pathnames referring to segments in other systems, changing the working directory to be on a foreign system requires no further changes to Multics software.

The other way in which users refer to other components of the hierarchy is to add directories to their search rules. In the

* It is necessary that both the site name and the network name be specified; one might in principle try to reach the same site through two different networks. If performing a character string comparison on the first component of the name seems too ad hoc a way to detect a network pathname, some different initial character, such as ">>", could be used, in addition.

version of the user ring linker which exploits Dick Bratt's KST manager, the search rules are stored as a series of segment numbers. Each of these numbers represents a directory in which a search is to be made for the name in question. For this reason, we cannot directly use the trick of storing a pathname which represents a directory on a different system. A very efficient and effective way to implement search rules is to create on the local system a dummy copy of the directory on the distant system which is to be used as a component of the search rules. In this dummy directory we will create for every name in the distant directory a link to that segment. It was discussed earlier that part of the act of initiating a segment on a foreign host involves making a copy of that segment on the local system. Clearly, the place to put this segment is in the dummy directory. In other words, we will start out with the dummy directory containing nothing but links, and as a segment associated with a particular link is initiated the link will be replaced with a copy of the segment itself, having the same name.

REWRITING OF MODIFIED SEGMENTS

When a segment has been copied from a foreign system to a local system as a result of an initiation, and has been subsequently modified, the question arises as to when it is appropriate to rewrite an updated copy of the segment back to the foreign system. There are two obvious ways in which to do this. One is to perform a rewriting operation on all modified segments at the time the "finish" condition is signaled in the process. This is effective unless Multics crashes or the user hangs up. Another technique which avoids the problem of crashes but is more costly is to provide, as part of the user's process, some procedure driven by a timer which makes a periodic sweep of all the dummy directories, writing back all the segments whose date-time modified is later than the last sweep. It is also clear that the user may wish to explicitly express rewriting under certain circumstances, and that certain special commands may have sufficient knowledge of their environment to command a rewrite. For example, rewriting might always be appropriate at the time a segment is terminated.

The obvious location for the dummy directories in the local hierarchy is off the process directory. Locating directories there avoids the necessity for providing permanent quota for these directories: however, it does mean that they are destroyed if the system crashes. It might be an option to locate the dummy directories somewhere else in the hierarchy, where they would have a more permanent existence. This would reduce the necessity for taking frequent sweeps through the dummy directories.

LOGGING IN THE HELPER PROCESSES

Whenever the user indicates his intention to access a segment on a particular system, RSEXEC must login a process on that system for this purpose, so the user must make available to RSEXEC a user name and password for the relevant system. Either this information can be stored in a segment which RSEXEC may reference, or the user may supply the information from his keyboard each time it is necessary to login. On service sites where the access control mechanism can be presumed to work there is no reason why the information may not safely be stored in the file system. If this mechanism were used from the Multics development machine, however, it would seem appropriate to enter one's foreign password from the keyboard.

The fact that it is necessary to login a server process using the RSEXEC protocol causes a problem in implementing the server process mechanisms on a Multics system. Only the answering service may create a process on Multics, and the answering service expects the login protocol to be a particular format, which does not happen to be the same syntax as the RSEXEC protocol. For this reason it will be necessary to modify the answering service so that it will accept the name and password in the RSEXEC syntax. The network file transfer protocol caused the same problem, and one such module has already been added to the answering service; thus it does not seem inappropriate to do it a second time. Warren Montgomery is proposing an extension to the answering service which is intended to solve exactly this sort of problem. An additional problem with the RSEXEC protocol is that it permits the login sequence to occur at any point in the dialogue with a foreign system. In other words, the user may open communication with a helper process on a foreign system, and at some arbitrary point request that the identity of the process, the process-group-id in Multics terminology, be changed. Multics does not permit this function. The answering service, once it has created a process with a particular identity, insists that the process maintain that identity for its entire lifetime. It appears that it is possible to arrange the RSEXEC environments running on other systems so that they will agree to obey the convention that if they intend to issue a login sequence they will do so only at the beginning of any transaction. If they send an RSEXEC request other than a login as their first request the Multics implementation will presume that they are willing to run the entire transaction using a system provided process whose access is probably very limited. It would be appropriate to use this process if, for example, all the distant user wished to do was to execute the "who" command on Multics and then go away.

LIMITATIONS OF THE RSEXEC PROTOCOL

There are two obvious limitations of RSEXEC as it currently exists: It is probably inefficient for large files, and it does not protect against simultaneous update of a segment by two or more RSEXEC users. RSEXEC is inefficient for large segments because it has no way for copying part of a segment only. Thus if a user touches only one word of a large segment he will discover that his computation is delayed while the entire segment is transmitted through the network. The extent to which this is a serious problem depends, of course, on the usage patterns of RSEXEC. The extension of RSEXEC to allow the transmission of parts of files is probably an interesting research project, since it must be done in a fashion which expands to a variety of operating systems.

The other problem is that RSEXEC contains no mechanism to prevent two or more users from attempting to update the same file simultaneously. It is presumed that any given file will be accessed by only one user at a time. While this is a potential source of great confusion, the probability is small under the expected usage patterns that a shared file will be in use by means of RSEXEC. The status information which RSEXEC transmits between systems includes date-time modified, which can be used to try to identify simultaneous updating situations.

EXTENSIONS TO RSEXEC

RSEXEC can be characterized as a mechanism for referencing and manipulating segments. It is not for sophisticated manipulation of directories and file systems. The set of RSEXEC commands listed above included only four file system commands: list, status, rename, and delete. Clearly, such Multics functions as manipulation of access control lists and adding multiple names to the segments are not possible through RSEXEC. RSEXEC does, however, accept the idea of extensions to the protocol for communication between particular systems. Thus, for example, the TENEX systems use a specialized protocol for communication which contain a few TENEX-specific file system commands. In addition to delete, TENEX supports an "undelete" command and a "delete it for real" command. It is easy to imagine Multics extension to RSEXEC which allows the setting of ACL's, for example.

MAINTENANCE OF REDUNDANT SEGMENT COPIES

The RSEXEC environment described earlier on TENEX allowed users to take directories on several systems and amalgamate them to form one global working directory. One advantage of this approach is that the same file can be stored in several systems, and RSEXEC can update each of these copies so that there is redundant storage of a file throughout the network. This means that if one system goes down the files are still accessible, and the user does not have to think explicitly about where each file is stored. The representation of the foreign systems as members of a global hierarchy does not lend itself so well to this particular function of RSEXEC. Providing redundant copies automatically is an interesting and usful service, so some thought should be directed toward integrating this idea into the Multics realization of RSEXEC.