To: Distribution

From: Richard J.C. Kissel

Date: 08/28/79

Subject: General User Interface to Multics File Transfer Facilities

Send comments to:
Kissel.Multics on MIT or System M

or call:
617-492-9319 or HVN 261-9319

## INTRODUCTION

This MTB proposes a general user interface to be used by any file transfer facilities implemented on Multics. It is modeled after the queued user interfaces on Multics such as those for absentee, retrieval requests, daemon requests, etc. However, interactive use must also be an integral part of a file transfer interface and so, it too, is provided for.

The remainder of this MTB describes the general features which must be in a file transfer interface and specific proposals for implementing these features on Multics. MPM style documentation for the proposed interface is given in Appendix A. Information necessary for a system administrator to set up the proposed facility at a site is given in Appendix B.

## GENERAL FEATURES

A general interface for file transfers must be able to support such diverse underlying protocols as: the Level 6 FTF (File Transfer Facility), the LFT (Logical File Transfer) protocol defined in HDSA, the Arpa file transfer protocols, and user defined protocols (perhaps operating on a bisync or X.25 communications line).

The goal of this MTB, then is to define the features necessary for an abstract file transfer facility, and then define the Multics user interface to this abstract facility. In implementing a specific file transfer protocol the programmer must map the features of the specific facility to the features of the abstract facility.

## Interactive or Queued Use

The user interface must provide for requesting file transfers to happen interactively (in the user's process) or by queuing the request (for later action by a daemon process). The rest of the user interface for these two modes of operation should be identical to avoid user confusion and programmer maintainability problems.

## Authenticated or Unauthenticated Use

Users should be able to provide the information necessary for a foreign host to authenticate their file transfer. Also, Multics should allow file transfers to or from a foreign host for authenticated or unauthenticated users (the latter must be restricted in some fashion).

## File Names

The user must specify two file names to do a file transfer: the file name on the local host, and the file name on the foreign host. The foreign file name is only of interest to the foreign host and should not be restricted by the user interface. The local file name will be a Multics pathname. An interesting extension to consider would be to allow the local file name to specify a device, or perhaps, an attach description. This would allow, for instance, the transfer of a tape file directly, without first copying it into a segment.

## Host Identification

The user must be able to specify the foreign host that is to participate in the file transfer. They may also want to specify a particular physical connection, if there is more than one between the local and foreign host, and the particular file transfer protocol to use if more than one is available.

This concludes the description of the features which must be in the abstract file transfer interface. What follows are specific proposals and issues involved in implementing this abstract file transfer interface on Multics.

## IMPLEMENTATION PROPOSALS

Transferring files in the user's process is conceptually simple. The program for doing the requested transfer just runs in the user's process and does the transfer while the user waits.

Queuing requests for later processing by a daemon gives rise to a large number of issues which must be solved. The first step in solving these issues is to model the queuing after queued facilities already on Multics, i.e. absentee. This essentially solves the user interface problems: how requests are identified after they are queued, how the user can assign priorities to his requests, and in fact, the general syntax of the user commands.

The remaining difficult problems are how the queues are arranged and what the daemons that process them look like. From the user's point of view the file transfer takes place to or from a foreign host, and perhaps, over a particular physical connection and using a particular protocol. This would seem to argue for a separate set of priority queues for each host, connection, and protocol combination. However, this seems unwieldy, leading to a possibly very large number of queues.

Another approach might be to have a set of priority queues for each physical connection. Since a transfer must take place over a physical connection it must be possible to map the user's view into physical connections. This works fairly well for "hardwired" connections, although queues must be added and removed as connections come and go. It can also be made to work for "dialout" connections, although the daemon which processes the requests associated with a "dialout" connection must be able to handle transfers for arbitrary foreign hosts using some arbitrary set of protocols.

Another possibility is to have a set of queues for each different protocol. This works fairly well except that it is hard to map the user's view of foreign hosts onto protocol queues. For instance, a user might want to list all of his transfer requests for a particular foreign host, in general, this would involve looking at all of the queues, which could be a time consuming operation.

Another approach is to have a set of queues per foreign host. This might lead to a very large number of queues because of the essentially unlimited number of possible foreign hosts (i.e. at least all of the current Arpanet hosts).

I currently favor a single set of priority queues for all file transfer requests. This is easy to implement and it makes the system administrators job easy. Further, the arrangement of queues should not be a user visible interface so that changes can be made as experience with the simplest implementation is gained.

At least two issues associated with this implementation must be resolved:

1) How are multiple processes allowed to access a single message segment?

2) How can the access of a user process which is running a protocol and needs access to the queues be restricted so that he can not destroy the queues?

The first issue could be resolved by adding a conditional update function (somewhat like stacq) to the message segment primitives. Suggestions for the resolution of the second issue are welcome.

We can now look at the daemons that process these queues, and the general framework in which they will run.

Each daemon will be responsible for implementing a specific file transfer protocol. There may be more than one daemon running at a site implementing some file transfer protocol if it is a heavily used protocol. When a daemon begins operation it will be given a set of logical connections that it is responsible for, and a set of foreign hosts for which it is to process requests.

Daemons will simply scan the queues looking for a file transfer request for a host for which it is responsible using the appropriate protocol. A request may be entered for a host with no required protocol, in which case any daemon which is responsible for that host may service the request.

A host table will be necessary to map the site specifiable host names into actual addresses on some network. The host table will also need to specify the default protocol or "don't care" for each host. The host table is also the mechanism by which hosts are mapped to queues if it becomes necessary to implement more than one queue.

The last points I will discuss are motivated by the problems of transferring files to or from a foreign host by an unauthenticated user on the foreign host. Transferring files from the foreign host is analogous to the problem of reading in card decks. In both cases the best solution seems to be a system pool where files are put for later pickup by authenticated users. In order to prevent the proliferation of system pools with the attendant problems of creation, management, and quota assignment, I propose there be a single system pool to be used by card input, file transfer, and any other facilities that need this type of storage. The existing pool_manager_ entries would be used to manage this system pool. There seem to be two equally good structures for this system pool:

1) >System_Pool>AIM_LEVEL>FUNCTION>USER_ID>ENTRY_NAME

2) >System_Pool>AIM_LEVEL>USER_ID>FUNCTION>ENTRY_NAME

AIM_LEVEL represents a quota directory for each AIM level as required by the rules of AIM. FUNCTION represents a directory for each different function, for example, card_input or file_transfer. USER_ID is the registered user_id of the user to whom the segments specified by ENTRY_NAME belong. The user is given s access to his directory and r access to his segments. The major difference in these structures is that in case 1 the system maintains the FUNCTION level of directories while in case 2 the user (probably using system code) manages these directories.

The transfer of files from the local host to a foreign host by an unauthenticated user on the foreign host can be handled by giving that user the same access to the files as the daemon process which is implementing the protocol.

Finally, the transfer of files by authenticated users, whether on a foreign host or the local host, is controlled by the normal access control methods of the local host.

# APPENDIX A

What follows is MPM style documentation for the proposed user commands. The commands are:

```
enter_file_transfer_request, eftr
cancel_file_transfer_request, cftr
list_file_transfer_request, lftr
move_file_transfer_request, mftr
```

The syntax of each command follows, [] enclose optional arguments, alternatives are separated by ¦.

Name: enter_file_transfer_request, eftr

This command requests a file transfer to take place either by queuing the request for later service by a Daemon process or interactively in the user's process.

Usage

```
     eftr <source> <destination> [<control_args>]
```

where:

<source> ::= [-name ¦ -nm] <from_path> [-at <host_id>]
is the source of the file transfer. <from_path> must be preceded by -name or -nm if it begins with a "-". It must be enclosed in quotes if it contains spaces or special characters. It must be followed by "-at <host_id>" if the file does not reside on the local host. The <host_id> is the site specifiable name of a foreign host.

<destination> ::= [-name ¦ -nm] <to_path> [-at <host_id>]
is the destination of the file transfer. It has the same syntax and restrictions as <source>.

<control_args> can be any of the following:

-queue N, -q N
specifies the priority queue in which the request is to be placed. This may not be specified if -interactive is specified. The default is queue 3.

-interactive, -i
specifies that this request is to take place interactively in the user's process. This may not be specified if -queue is specified.

-protocol STR, -prot STR
STR specifies the protocol to be used for the file transfer. STR must be quoted if it contains spaces or special characters. The default is specified by the system administrator in the host table.

-user STR
STR specifies the user on whose behalf the file transfer is to be done. This may be used by the foreign host for authentication of the file transfer. The default is the user id of the user who submitted the request.

-password STR, -pw STR
STR is a password that may be used by the foreign host to authenticate the file transfer. There is no default.

-force, -fc
specifies that the destination file is to be written even if it already exists. The default is not to overwrite existing files.

Name: cancel_file_transfer_request, cftr

This command cancels a file transfer request that has been queued by the eftr command.

Usage

```
     cftr <request_identifier> [<control_args>]
```

where:

<request_identifier> is one of the following:

<from_path> [-host <host_id>]
is the full or relative pathname of the source of the file transfer. If the source is not on the local host the "-host <host_id>" argument must be specified.

{-entry | -et} <from_entry> [-host <host_id>]
identifies the request to be cancelled by <from_entry>, the entryname portion of the source file pathname. The star convention is allowed.

-id <request_id> identifies the request to be cancelled by its request identifier. See the MPM Reference Guide for a description of Request identifiers.

<control_args> can be any of the following:

-queue N, -q N
specifies that queue N contains the request to be cancelled, where N is a decimal integer specifying the number of the queue. If this control argument is omitted, only the default queue is searched. This control argument is incompatible with the -all contol argument.

-all, -a
searches all )riority queues for the request starting with the highest priority queue and ending with the lowest priority queue. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses messages telling that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

-user <user_id>

specifies the name of the submitter of the request to be cancelled, if not the group identifier of the process. The <user_id> may be specified as Person_id.Project_id, Person_id, or .Project_id. Extended access to the queue are required. This control argument is primarily for operators and administrators. Both r and d extended access to the queue are required.

Name: list_file_transfer_request, lftr

This command lists file transfer requests that have been queued by the eftr command.

Usage

```
     lftr [<request_identifier>] [<control_args>]
```

where:

<request_identifier> is the same as described for the cftr command.

<control_args> can be any of the following:

-absolute_pathname, -absp
prints the full pathname of each selected request, rather than just the entryname.

-admin [<user_id>], -am [<user_id>]
selects the requests of all users, or of the user specifed by <user_id>. If the -admin control argument is not given, only the user's own requests are selected.

-all, -a
searches all queues and prints the totals for each non-empty queue wherher or not any requests are selected from it. If the -all control argument is not given, the default queue is searched. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses the printing of the state of the request. This control argument is incompatible with the -long and -total control arguments.

-long, -lg
prints all of the information about each selected request including the long request identifier and the full pathname. If this control argument is not given, only the short request identifier, entryname, and state are printed. The -long, -brief, and -total control arguments are incompatible.

-long_id, -lgid
prints the long form of the request identifier. If this or the -long control argument is not given, the short form of the request identifier is printed.

-position, -psn
prints the position within its queue of each selected request. When used with the -total control argument, it prints a list of all the positions of the selected requests.

-total, -tt
prints only the total number of selected requests and the total number of requests in the queue plus a list of positions, if the -position control argument is given. If the queue is empty, it is not listed. This control argument is incompatible with the -long and -brief control arguments.

-user <user_id>
selects only requests entered by the specified user.

Name: move_file_transfer_request, mftr

This command moves file transfer requests to a different priority queue, host, or makes them interactive.

Usage

```
     mftr <request_identifier> [<control_args>]
```

where:

<request_identifier> is the same as described for the cftr command.

<control_args> may be any of the following:

-queue N, -q N
specifies that queue N contains the request to be moved, where N is an integer specifying the number for the queue. If this control argument is omitted, only the default queue is searched. This control argument is incompatible with the -all control argument.

-all, -a
searches all queues for the requests to be moved. This control argument is incompatible with the -queue control argument. The target queue is not searched by the -all control arguemnt.

-to_host <host_id>
specifies that the request should be moved to the host specified by <host_id>. If this control argument is not given, the original destination host is used.

-to_queue N, -to_q N
is a required control argument specifying which queue to move the request to.

-brief, -bf
suppresses messages telling the user that a particular request identifier was not found or that requests were moved when using star names or the -all control argument.

-user <user_id>
specifies the name of the submitter of the requests

to be moved. The default is to move only requests entered by the user executing the command. The <user_id> can be Person_id.Project_id, Person_id, or .Project_id. This control argument is primarily for the operator and administrators. Both r and d extended access to the queue are required. This control argument causes the command to use privileged message segment primitives which preserve the original identity of the submitter. If the process has access isolation mechanism (AIM) ring one privilege, the AIM attributes of the original submitter are preserved. Otherwise, the AIM attributes of the current process are used.

# APPENDIX B

Documentation for the system administrator on how to set up the file transfer Daemons, how to set up the request queues, and how to set up the system pool directories is to be supplied.