Network Working Group K. Pogran (MIT-Multics)
Request for Comments: 614 N. Neigus (BBN-NET)
NIC #21530 Jan 1974
References: RFC #607, RFC #542

Response to RFC 607, "Comments on the File Transfer Protocol"

Mark Krilanovich and George Gregg have pointed out a number of "sticky" issues in the current File Transfer Protocol. Although we don't agree with all of their proposed protocol modifications, we do feel that each of the points they have raised should be given some thought by everyone concerned about FTP. Each numbered paragraph in the discussion below is a comment on the identically-numbered paragraph in RFC 607.

1. Instructions to the Server to be "passive" are defined to apply only to the next single file transfer operation, after which the Server reverts to active mode. RFC 607 does, however, point out a drawback in the present specification, in that there is no way for a user to "change his mind": once he has told a server to be "passive", he has to initiate some file transfer operation. The suggested solution is a welcome one. We suggest that the text of the "successful reply" to the ACTV command indicate whether the server had previously been in "active" or "passive" mode, viz:

200 MODE CHANGED TO ACTIVE

or

200 MODE IS ALREADY ACTIVE

It is important to note that once some servers "listen" on a connection in response to a PASV command, they no longer can examine the Telnet control connection for the possible arrival of an ACTV command. User-FTP programs should precede the ACTV command with a SYNC sequence to ensure that the server will see the ACTV command.

2. While the length of an FTP command -- either three or four characters -- might often be irrelevant to a system which interacts over Telnet connections on a line-at-a-time basis, we can see how a variable command length might be harder for a character-at-a-time system to handle, especially for a server implemented in assembly language. Quite a bit is gained, and nothing seems to be lost, by requiring that FTP commands be four characters, so we agree with the suggestion in RFC 607.

3. While the FTP document may be somewhat ambiguous in its specification of the order of the handshaking which takes place following a file transfer command, such an order does exist as far as is possible for the two asynchronous processes described in "The FTP Model" (section II. B of RFC 542) -- the Telnet Control process (Protocol Interpreter) and the Data Transfer process. The user is required to "listen" on the data connection before sending the transfer command. Upon receipt of the command the server should first check that the status of the file specified by the argument to the file transfer command is okay, and, if so, attempt to open the data connection. If there are file system problems, no attempt should be made to open the connection. In this way,

the primary response to the command gives an accurate picture of the transfer status -- i. e., connection opened and transfer started (250), or connection not opened because of file problems (450, 451, 455, 457) or connection problems (454). The secondary reply follows the conclusion of the transfer, and describes its success or failure.

If a particular FTP implementation cannot monitor the data connection and the Telnet control connection simultaneously, then it must establish a timeout waiting for the data connection RFC. In addition, a minimum interval should be specified for which all servers must wait before deciding that the data connection cannot be opened. We suggest that this interval be no shorter than fifteen seconds.

4. The protocol states that servers should return "success", replies to commands such as ACCT and ALLO which were irrelevant to them. We recommend that the protocol say "must" rather than "should".

5. Specification of maximum lengths for lines, pathnames, etc. is a fine idea, as is the suggestion of a Server poll. Typical values for the present Multics implementation (provided by Ken Pogran) are as follows:

Telnet lines: 256
User names: 32
Passwords: 8
Account Numbers: (na)
Pathnames: 168 (yes, 168)

6. We strongly disagree with Mark on this point. The algorithm a user-FTP should use goes something like this:

a. Examine the first four characters of the reply.
b. If the fourth character is a space, the reply is not a multi-line reply.
c. If the fourth character is a hyphen, the reply is a multi-line reply, and the text portion of this line and succeeding lines should be reported to the user if this is desired.
d. On each succeeding line, if the first four characters are not the three digits of the original reply code followed by a space, the line is entirely a text line and should either be reported to the user or discarded.
e. If the first four characters on the line are the three digits of the reply code followed by space, this line is the last line of the reply.

This algorithm seems simple enough, if nesting of replies is not required (see comments on paragraph 7, below). This sort of continuation-line convention provides a number of benefits to the person coding a server. Consider the problem of providing a directory listing, in response to a STAT command whose argument is the pathname of a directory. If the FTP Telnet control connection is treated as a pseudo-typewriter (as most ordinary Telnet connections are), the writer of an FTP Server may be able to "borrow" the code from the system command which provides directory status (listing) information, as follows (in a pseudo-PL/l syntax):

```
call write_out_line ("151- Directory listing follows") ;
call list_directory_contents (directory_pathname);
call write_out_line ("151 Directory listing complete");
```

The same can be done for the file status reply (code 150). Otherwise, a program must be written which performs the function of the directory-listing command, but uses a special output format. If the implementor of an FTP Server wants to be "nice" and list file attributes, as well as file names, in the directory listing (as many directory-listing commands do), this could be a fairly big job. If already-written software can be borrowed and incorporated into the FTP Server, the implementor of the FTP Server can put more of his effort into doing a better job of building the "guts" of the FTP Server.

7. It is not obvious why multi-line replies are allowed to be nested to an arbitrary depth. Only truly spontaneous replies may nest inside other replies -- and it is easy to convince yourself that they will only nest to depth one. It was envisioned that some messages from "the system" might not allow the "exterior" multi-line message to finish; the scenario might go something like this:.

```
151- Directory listing follows:
a1pha.p11
alpha
rfc.runoff
mailbox
010- From Operator:
010 Emergency shutdown in 5 mins. due to hardware probs.
beta.fortran
foo.lisp
151 Directory listing complete.
```

It has been pointed out to us that:

a. Messages from "the system" in general cannot be guaranteed to come at the beginning of a line.

b. It may be difficult to modify "the system" to preface such messages with an appropriate FTP reply code.

Therefore, we propose that, since user-FTP implementations must handle multi-line replies, system messages "splattered" into the middle of replies need not be escorted by FTP reply codes. The user-FTP thus need not detect and handle "nested" FTP replies.

8. RFC 607 proposes that any data between the last end-of-record marker of a file and the end-of-file marker be discarded. We agree. The sender of the data has clearly violated the protocol, and the receiver cannot divine the sender’s original intent.

9-11. The suggestion that reply codes beginning with the digit "2" be taken as successful, and all others be taken as failures, severely restricts use of the available "reply code space". We agree that the present scheme is disorganized and requires far too much "intelligence" on the part of a user-ftp automaton. With the present scheme, unless the automaton’s reply-interpretation is table-driven, it is extremely likely to make a mistake. We feel that the whole reply code strategy should be redesigned; some of the ideas proposed in RFC 607 could fit in with such a redesign, but we do not think that Mark’s suggestion is the way to go.

12. 000 and 020 are used by the Server to indicate that it has heard and answered the ICP to socket 3, but cannot accept file transfer commands yet. 020 was intended to indicate how much of a time delay could be expected, while 000 was ambiguous on this point. We suggest that the two be merged to mean "I am here; please wait a specified or unspecified amount of time"; then, 300 would clearly mean "I am ready; you may now send me commands".

13. There is no typographical error here. A TENEX representative suggested that, rather than give a "fail" reply to a particular request because the user is not logged in, a server might ask for his account number (or ask him to log in) and then proceed with the previous request, which has been held in abeyance. While this may be convenient for a server, it is not necessary, and certainly ambiguous to hold a command in abeyance while obtaining an account number. Since any server may spring this on a user, all user-FTP implementations must be able to cope with this twist, which adds a good deal of required complication to the "minimal" user-FTP implementation. We propose that the 331 reply be eliminated, and that the server forget the requested operation and return a 4XX reply if an account is needed.

Jon Postel has remarked that "mail text should follow the same limit as commands and long ’lines’ of mail text have been trouble for some FTP Servers." We agree. In fact, mail transmitted over the FTP Telnet control connection has other problems, too: Since FTP is (nominally, at least) supposed to be usable from TIPs, Multics implemented its standard character erase and line kill conventions on the control connection for the convenience of TIP users (it was actually easier to have those conventions in effect than to turn them off!). Of course, no erase/kill processing was done on the data connection. The intent of the MAIL request was to allow users at terminals to access an FTP Server directly and transmit mail; it was presumed that mail-sending automata which gathered the mail to be sent into a file would use the MLFL command and transmit the mail over the data connection. Presumably, long lines would not be a problem, and, of course, no erase/kill conventions would be in effect. Well, at least one major system (TENEX) has a mail-sending automaton which transmits mail over the Telnet control connection using the MAIL command - even though it has previously gathered the mail into a file! Line-length considerations could be a severe problem here, and the fact that the Multics line-kill character is the at-sign (@) caused grief in reading mail from TENEX users who included their "return address" in TENEX’s SNDMSG syntax, as USERNAME@HOST. We propose that mail-sending automata be required to use MLFL, rather than MAIL.