Network Working Group Edward Taft (PARC-MAXC)
Request for Comments: 617 Feb 1974
NIC #21771

A Note on Socket Number Assignment 2

INTRODUCTION

In several current and proposed protocols, as well as in a few other documents, the assumption is made (or implied) that a user process in control of one end of a Telnet connection has free access to a group of socket numbers beginning with or surrounding the Telnet socket numbers.

For example, the File Transfer Protocol (RFC 542, NIC #17759) specifies that the default data transfer sockets are S+2, S+3, U+4, and U+5, where S and U are the server and user sockets involved in the initial connection (ICP).

Similarly, the proposed Network Graphics Protocol (NIC #19933) provides for a second connection pair for graphics data, parallel to the Telnet connection, using (at both ends) sockets n+6 and n+7, where n is the Telnet receive socket.

I would like to point out to designers of protocols that the Host-Host Protocol (NIC #8246) quite explicitly places no interpretations or constraints on host assignment of socket numbers, except for the use of the low-order bit to indicate direction of data flow. We should refrain from making further assumptions (as does the "Socket Number List" document (RFC 503, NIC #15747) in stating that the low-order 8 bits are "user-specified"), lest we inadvertently exclude certain host software architectures or protocol implementations.

AN EXAMPLE

To illustrate the source of my concern, let me briefly describe the user software interface to the network in the PDP-10 NCP implementation currently in use at HARV-10, CMU-10A, and CMU-108. I will then show why the fixed socket number requirements of the two protocols I mentioned above present implementation difficulties, especially at the server end.

Opening a connection at one of these hosts causes the creation of a "device" (in approximately the same manner as, say, mounting a disk pack). As such, an open connection is subject to any one of a number of operations on devices in 10/50, including assignment of logical names, opening for data transfer, and reassignment to another job.

The NCP allows a (non-privileged) user program to specify the low-order 8 bits of the socket number of any connection which it opens, and to specify that the other 24 bits be assigned in one of three ways:

-- As a function of the job number, making a "job-relative" socket.

-- As a function of the user identification, making a "user-relative" socket.

-- As a "guaranteed unique" number, i.e. one assigned by the NCP such that no other socket number in use has the same high-order 24 bits.

A program may also specify all 32 bits of a local socket number provided the high-order 24 bits are the same as the corresponding bits in some other socket already owned by the same job.

The NCP will, of course, allow assignment of a socket generated in any of the above ways only if it is not already in use by the same or any other job.

PROBLEMS IN THE FTP SERVER IMPLEMENTATION 5

The FTP server is implemented in a manner that some readers may find reminiscent of Padlipsky's "Unified User Level Protocol" (RFC 451, NIC #14135). Rather than directly executing most FTP functions (in particular, system access and file transfer functions), it merely maps FTP commands into local commands which it "types" on a pseudo-Teletype (PTY) to a subjob, and similarly maps local responses into FTP responses.

This scheme makes maximum use of existing software and mechanisms for user authentication, accounting, and file access, and eliminates the need for the (privileged) FTP server to perform them interpretively. (This conforms to the "principle of least privilege" described in RFC 501, NIC #15818.)

In this implementation, FTP data transfers are performed by an entirely different process (with a different user identification) from the one that manages the server end of the Telnet connection. Hence, since server sockets S and S+1 belong to the server "control" job and sockets S+2 and S+3 are in the same 256-socket number range, the latter two sockets are inaccessible to the "data transfer" subjob.

Those who attended last spring’s FTP meeting may recall that I objected strenuously to the requirement that the FTP server use a fixed pair of data sockets relative to its Telnet sockets, as opposed to the old scheme in which the server has complete freedom in the choice of its data sockets. The principal reason is that it would seem to rule out the "two-process" scheme I have just described.

In fact, in our case there is a way around the problem. The FTP server control job can open the data connections itself, then "reassign" the created "device" to the data transfer subjob. A kludgey solution at best, and one I would rather have avoided! Inter-job socket reassignment is hardly an operation one is likely to find available in most operating systems.

DIFFICULTIES WITH THE GRAPHICS PROTOCOL

Providing a graphics connection parallel (at a fixed socket number distance) to the Telnet connection might potentially present the same difficulty as described above for FTP connections.

In the most frequently used model of Telnet communication, as well as in many implementations, the server Telnet is a process quite distinct from the "user" process under its control. The two processes are typically interfaced through the operating system’s terminal service in such a way that the "user" process perceives a ,terminal" as opposed to a "network connection".

In Tenex, for example, a job controlled from a network terminal has no handle whatever on the server Telnet connection; hence, it has no way of obtaining control of sockets n+6 and n+7 for a graphics connection.

In the Harvard-Carnegie 10/50 implementation, it happens (for largely accidental reasons) that a job logged in from the network does have control (i.e. is considered the owner) of the server Telnet sockets.

However, there is another problem. Many operating systems provide means by which the association between terminals and jobs may be changed.

To use familiar terminology, a terminal may be "detached" from one job and "attached" to another, in a manner which does not destroy any jobs or any network connections.

Hence, it is entirely possible that a user could start up a program that uses sockets n+6 and n+7 (where n is the server Telnet receive socket), detach his terminal from that job, attach it to another, attempt to run a program using the Graphics Protocol, and have the attempted data connection fail because sockets n+6 and n+7 are already in use (for the same value of n, since we are referring to the same network terminal).

CONCLUSION 7

There are, of course, a few network-wide socket number conventions necessary for establishing initial connection.

Reserving sockets 0-255 for standard ICP functions is an example of one such convention.

Additionally, I think that for the purpose of simplicity it is reasonable to expect any process to be able to gain control of a small block of "adjacent" sockets, such as an even-odd pair (as in ICP), if it asks for them at the same time.

However, as the foregoing examples have demonstrated, to impose further fixed socket number requirements is to risk the danger of making unwarranted assumptions about the nature of protocol implementations, the structure of user processes, etc., at individual hosts.

Once the initial Telnet connection is established, any other necessary connections should be established by negotiation over the Telnet connection (e.g. by messages of the form "Please connect to my socket xxxxxx", "OK, connecting via my socket yyyyyy"). There is absolutely no need for any protocol to specify fixed socket numbers, except for the purposes of the initial connection (ICP).