1/21/72 M.A. Padlipsky

File
Multics
Implementation
Reports

MULTICS ARPA NETWORK IMPLEMENTATION STATUS

As has been observed in all other areas of Multics, the initial implementation of the Multics ARPA Network software has proven to be largely a "first draft", with attendant problems of both known and unknown nature. Currently, both the cost of the Network on Multics in terms of charges and resource consumption, and the subjective "feel" of Multics Network usage are far too heavy. Consequently, the Multics Network/Graphics Group has become engaged in a concerted effort to improve the performance of the existing implementation considerably. It should be noted at the outset that considerations of pride of craftsmanship--although certainly not absent--are far from being the primary motivation for these efforts. The two major considerations which necessitate extreme measures in the area of performance improvement actually are the fact that Multics is considered to be a major service offering of the Network to its user community, and the fact that the appearance Multics presents over the Network potentially has a strong impact on future general Multics development.

Known Problems

Ignoring certain interrelationships to facilitate presentation, the known problems of the current implementation may be separated into two areas, those involving the "Network Daemon" process and those involving Network-sense "server" and "user" processes. Metering and a priori analysis have been performed in both areas.

The Network Daemon process is currently a separate Multics process which exists primarily to furnish a guaranteed responder to Network control messages (such as requests for connection). Presently, the Daemon is explicitly logged in by Operations, and after an initializing start-up exec-com the Daemon blocks, waiting for wakeups from the device interface module which manages the IMP (the "IMP DIM"). We have distinguished several aspects of the Daemon's operation which bear heavily on its performance:

1. Cost of Wakeups. A counter is maintained in the Daemon which causes a cumulative ready message to be printed out on the Daemon's console every 100 wakeups. Under typical system loads, the average transaction seems to require some 250 ms. of Daemon time. Paging is quite high compared to cpu time, although no arithmetic has been done on it. (Page-tracing has been, however.) It should be noted that metering has also been performed on the Interprocess Communication Facility block/wakeup primitives, and that with paging time included each block/wakeup cycle appears to cost some 100 ms.--about 40% of the Daemon's transaction time
2. Number of Wakeups. With the cost of each Daemon wakeup's being high, concern naturally focused on the number of wakeups the Daemon takes. The current strategy requires a wakeup for

each "RFNM" (the Network-defined request for next message) and a wakeup on the arrival of an allocate message for a particular socket. Indirect evidence suggests that more that half of the Daemon's wakeups are to process RFNM's. As allocates depend on the idiosyncrasies of the foreign host, they are less easy to quantify. It is of interest, though, that in both the RFNM and the allocate cases the Daemon wakes up essentially only to in turn wake up another process and allow it to proceed.

3. Superfluous Paging. There are three known areas in which Daemon paging activity is greater than need be. a) Page tracing revealed that about half the page faults taken in the Daemon were on the ring 4 environment which is present only because it was initially more convenient to block in that ring rather than in ring 1, where the Network Control Program (NCP) primitives reside. An experiment was performed wherein the Daemon was run from the ring 1 "Repair" system daemon which showed a per-transaction time decrease of some 25% stemming from the elimination of the third ring from the environment. b) Less quantifiable, but still significant, it is known that the binding order of the NCP primiitves needs tuning. c) Also unquantifiable, and a much deeper issue, it is known that the NCP primitives and the IMP DIM could be made to decrease the working sets of both the Daemon and all non-Daemon Network processes by sharing data bases rather than maintaining separate ones as at present.

4. Foreign Site-Induced Extra Overhead. It should be noted that the Daemon must respond to such activities of foreign sites as periodic surveys, echo commands, and the like. This is a fact of Network life, and not much can be done about it other than working out a scheme for charging for such activities--which would be rather difficult both politically and practically. More can be done about a second kind of extra overhead, however. That is, in cases where the foreign system, by means of the allocate mechanism, limits Multics to sending very small messages out, otherwise avoidable Daemon wakeups are engendered at great rate. This problem can be dealt with either by education or, possibly, by protocol changes.

5. Unnecessary Charges. Two sources of charges exist which make Network bills more expensive than they need be. First, the Daemon runs with a console largely for debuging convenience; were connect time costs (which are based in part on the the port which is tied up) to be reduced for consoleless processes (such as absentee) charges could be decreased noticeably by making the Netowrk Daemon a consoless process. (As the Daemon is meant to be running whenever Multics is, its connect time is high.) Second, there currently exists a deficiency in the Multics charging apparatus such that a process which remains logged in across a shift change is charged at the rate for the initial shift for the entire "console session" rather than at the appropriate rate for the actual time spent on each shift. As the Daemon is normally logged in during first shift, it is charged at the highest rate for all its time unless the system

or the Daemon goes down and is brought up again during second or third shift. (Note that as overhead decreases and Network use increases it is quite possible that the Information Processing Center will assume the Daemon's costs directly; however, this factor should not diminish our concern with charges in the abstract.)

Turning to non-Daemon processes, it should be noted that there are two types under consideration: "server" processes (which are created by Multics in response to a login over the Network), and "user" processes (which allow a Multics user to login to a foreign system, under the management of the "network" command). Note also that the "third ring" problem mentioned in regard to the Daemon is also quite significant in these other Network processes. In this instance, page faults on the ring 1 environment when invoking NCP primitives are in some sense superfluous, and the fact that wall crossings from ring 4 to ring 1 have been measured at some 12 ms. (as opposed to 4 ms. or so for crossings into ring 0) takes on added significance as it represents a larger fraction of the per-transaction time. The following considerations also apply to the performance of Network processes:

1. IO Transaction Times. As a simple measure of the difference between local typewriter processing times and their Network counterparts, repeated "nothing" commands were performed both directly logged in to Multics and in a server process. The cpu times (from ready messages) averaged about 2.5 to 1 higher for the Network process on similarly loaded system; page faults (also from ready messages) were about 3 to 1. (When the same experiment was performed by a friend at SRI, it turned out that when he was transmitting a character at a time the situation was nearly twice as bad as when he was sending line at a time.) A slightly more elaborate experiment involved direct (hcs_$usage_values) metering of ios_$write calls; the Network process was worse by about 4.5 to 1 in this case. Ready messages for non-IO bound commands are also typically higher in server processes, but no concerted collation has been performed.
2. "network" Command Times. The experiments described above involved frequent Multics-to-Multics logins over the Network, using the "network" command. Typically, the ready messages when the command was terminated were quite close to the totals in the server process. As the command is, after performing the initial connection protocol (say 5-6 seconds), simply doing local/Network reads/writes, this suggests that per-transaction IO costs are indeed high. A special metering version of the command (using usage_values) showed reads and writes running around 80-90 ms. This result agreed closely with the metering of ios_$write calls mentioned above. (It must also be observed that use of the "network" command also generates greater Daemon activity.)
3. Character Processing. In both user and server processes,

conversion must be performed between the 8-bit Network ASCII character set and the 9-bit Multics set; Telnet Protocol control characters must also be processed; further, for server processes canonicalization, erase and kill processing, and software escape processing must also be performed. Direct metering of the "telnet_" subroutine (which in the current implementation manages code conversion and control character processing) showed averages of 3-4 characters per ms. Similar metering of "canonicalize_" (which is currently called each time the server process receives an input line) showed averages of about 1 character per ms.

4. Response Time. The "feel" problem shows up most clearly in server processes. When 50 or 100 single character write calls were made in a tight loop during the ios_$write experiment, there was rarely a single perceptible pause when directly logged in, whereas when logged in over the Network there were 3 to 7 pauses of several seconds each. The pauses, of course, happened because the server process had used up its quantum and had to be re-scheduled (without any "interaction switch" set). Another frequently noted waiting problem occurs when logging in on a heavily loaded Multics system: pauses of several minutes have been experienced during the printing of the message of the day.
5. Foreign-Site Induced Extra Overhead. Many foreign sites, TIP's in particular, can transmit to Multics on a character at a time basis. This generates extra overhead in terms of wakeups for the associated Multics server or user process, particularly for the server process, which must be given a wakeup by the Daemon to inspect each message (for end of line) in the current implementation. Also, for both server and user processes if the foreign site gives small allocations the process may also have to be awakened when the allocate message arrives (in addition to the wakeup of the Daemon process); on this point, see also the "write_force" discussion below.

Short- and Medium-Range Solutions

The primary criterion applied to the choice of immediate solutions for the above problems was that of high, quick payoff. That is, major strategy changes which would involve considerable redesign and reprogramming have been deferred in favor of working first on those less fundamental issues which hold promise of large performance improvement in a short time. The broader issues are discussed below. The decision to defer their implementation was motivated by a desire to avoid having Multics Network use remain in its current, unpleasant state for a matter of months when significant improvements could be effected in a matter of weeks.

The following tasks are either currently being worked on or have been recently completed:

1. RFNM's at Interrupt Time. The IMP DIM is being altered to process RFNM messages when it receives the interrupt from the IMP, rather than wake up the Network Daemon to handle then. When finished, this change should halve the number of Daemon wakeups.
2. Ring 1 Daemon. The Network Daemon now blocks in ring 1 rather than ring 4. The elimination of page faults on the ring 4 environment results in a per-transaction time decrease of about 25%.
3. Ring 0 NCP. The NCP primitives are being altered to operate in ring 0 rather than ring 1. This change will allow the user and server Network processes to avoid taking page faults on a third ring environment, and will eliminate the long (about 12 ms.) ring 4 to ring 1 wall crossing which currently occurs on each IO transaction. If the ring elimination savings are as similar to those in the Daemon as expected, transactions should cost a third less. (Other implications of this change include the ability to return the Daemon to blocking in ring 4, the elimination of wall crossings whenever the NCP primitives or the Daemon call the IMP DIM, and two rather important philosophical points: it facilitates the integration of the NCP and the IMP DIM, particularly in regard to data bases, and it makes possible the major strategy change regarding Daemon blocking discussed below.)
4. "write_force". In order to improve user and server process response time, the following tactic has been adopted: The formerly-privileged NCP "write_force" primitive has been opened up to all Network processes; this allows the Daemon to accept (up to the limit of its available buffers) all the characters the other process wishes to write irrespective of the current allocation for the socket. Thus, the other process need not be awakened when allocates arrive, and the Daemon simply sends the next portion of the message when it processes the allocate. (As this change dealt with "feel", it is not really quantifiable; however, the day after it went it we received an unsolicited compliment about the improved feel of Multics from an ITS user.)
5. Canonicalization. Work is in progress on the problem of server process input stream canonicalization. Although the design is complicated by considerations involving Telnet Protocol control character processing and the packing of 8-bit characters into 9-bit fields, we hope to come up with a scheme which decreases the per-transaction costs appreciably by avoiding full-fledged canonicalization processing in those cases where it is unnecessary.
6. Message Size. In an attempt to alleviate the superfluous wakeups problem brought about by a foreign site's sending a Multics server process messages on a character at a time basis, the login responder for the "CNet" project now prints a message asking TIP users to go to line at a time mode. An update to the Network Resource Notebook which also stresses this point will be prepared.

The following tasks are pending, awaiting either personnel availability or decisions on their desirability:

1. "Speedy Wakeups". Once the NCP is in ring 0, it would be straightforward for the Daemon to send wakeups to server and user processes with the same pxss_$wakeup_int "speedy wakeup" mechanism currently used by the IMP DIM to wake up the Daemon. There is a policy decision required here, but it would seem that the improved response to be gained is worthwhile--especially in view of the confusion as to whether the "interaction switch" exists anymore for TTY DIM use, and whether the Network could use it even if it does.
2. Shift Charges. A system-wide solution to the per-shift charging problem which the Daemon in particular encounters apparently involves arduous design and implementation considerations. A stop-gap solution could be introduce by instructing Operations to log the Daemon out and in when the shift changes. This might create more confusion than it's worth--but it's probably worth several hundred dollars a month.
3. Deconsolizing/Automating. Daemon costs might also be cut by making it a consoleless process as discussed earlier. A further argument for deconsolizing is that it would be desirable to have the Network initialized automatically (during system initialization) to eliminate the possibility of not having the Network up due to operator oversight. (This "automating" could also be applied to the shift charge problem.)
4. Accounting. Another approach which could be taken to the problem of Daemon costs is to introduce accounting callsfor the process in whose belhalf the Daemon is operating, in like manner to standard "SysDaemons". This tactic would merely shift, not solve, the overhead problem, but it may be a more equitable approach than writing the Daemon entirely off to overhead.
5. Allocates. As alluded to above, the problem of "superfluous" wakeups to service allocate messages could could possibly be attacked on the protocol level. That is, we could formally propose that the Telnet Protocol be amended to legislate a "reasonable" minimun allocation for the receive socket on the user side of a Telnet connection. 80-120 characters seems to be a reasonable range to shoot for, on the premise that one should be able to send a whole typical line at one write. There might be strongish resistance to this view from the TIP people, however, so we do not intend to go ahead with any such proposal without a strong MAC-wide consensus in its favor.

Major Strategy Issues

As noted, the previous section dealt with solutions which could be produced on a time scale of days or weeks, rather than months. We turn here to issues which either may involve a longer time scale to implement or at least involve decisions on major points of strategy.

1. Daemon Transaction Cycle

To take the item with the highest potential payoff first, the following section was prepared by Ed Meyer as a preliminary proposal:

Network Daemon Efficiency
Blocking vs. Waiting

Through metering experiments, it has been discovered that a call to ipc-$block which actually give up the processor costs about 100 ms., about 1/2-2/3 the expected cost of the entire Network Daemon event processing cycle. If some means of avoiding part of the cycle were available, the savings might be large.

There is an alternate method of giving up the processor that is available in ring 0, "waiting". Typically, it is used to give up the processor for short periods of time while waiting for a faulted page to come into core. Its cost is assumed to be no more than a few milliseconds. A scheme for employing the "wait" rather than the "block" mechanism has two advantages: 1) "wait" is presumed to be several tens of times as efficient as "block"; 2) the Daemon will operate mostly in a single ring environment, avoiding the costs of ring crossing and faults on the ring 4 environment.

Under this scheme, the Network Daemon would operate as follows: the Daemon process, which is brought up in ring 4 (after the NCP goes to ring 0), enters the NCP in ring 0 and calls pxss to wait on the Network event id. The interrupt side of the IMP DIM calls pxss to "notify" the Daemon of an event. Upon returning from the call to pxss_$wait the Deamon performs processsing in the NCP - IMP DIM complex, then calls wait once more. This is similar to the current scheme except that "wait" and "notify" calls are substituted for "block" and "wakeup" calls.

A possible additional feature is for the Daemon to return to ring 4 and block conventionally in the case of extended Deamon inactivity (perhaps on the order of a minute).

The advantages and disadvantages of this scheme relative to the current mechanism seem as follows:

Disadvantages: 1) It ties up certain system resources while waiting in ring 0. A waiting prcess remains eligible, and this costs perhaps two pages in core plus a few system table entries. (It also requires that the system tuning parameters be suitably modified.) 2) A scheme is required to force the Network Daemon out of ring 0 at critical times, such as at system shutdown.

Advantages: 1) There are obvious response and efficiency improvements for the Network. There is at least a 50% processing reduction merely due to not calling block, but probably also a significant reduction through not constantly tracing through a ring 4 environment. 2) It can be argued that there is also a corresponding improvement in overall system performance, because the Daemon is giving back to the system about 100 ms. of processor time per transaction and the core saved by not paging on the ring 4 environment, which amounts to at least 9 pages. Considering that the Daemon currently wakes up every 20 seconds, on the average, this is by no means insignificant. 3) To implement this scheme involves a minor reprogramming effort. In essence, calls to "wait" and "notify" must be substituted for calls to "block" and "wakeup".

Other considerations: It have been previously proposed that more processing be done at interrupt time, specifically that the ALL data allocation and RFNM "ready for next message" messages be intercepted at interrrupt time by the IMP DIM and processed there. This scheme and the "wait in ring 0" scheme are not exclusive alternatives. Each is independently desirable and provides benefits not available with the other.

The processing of ALL and RFNM messages at interrupt time rather than by the Daemon can eliminate many wakeups of the Daemon. However, it is infeasible to do all types of processing at interrupt time, and the Network Daemon must still be independently invoked. Therefore, it is still desirable to improve the efficiency of the Daemon.

While "processing at interrupt time" seems to be a large reprogramming effort of uncertain dimensions and impact on the system, "waiting in ring 0" is comparatively simple, with system implications more limited in scope. Moreover, it can be rapidly implemented. Therefore, it would seem that "waiting" ought to be accomplished first.

2. IMP DIM - NCP Interface

As implied by the "waiting" proposal, the division of labor between the IMP DIM and the NCP is currently rather one-sided, with the NCP (and/or the Network Daemon) handling even such functions as allocations. This separation historically stemmed from a view that the IMP DIM should be concerned only with the IMP as a device (or, with the IMP - Host Protocol), while the NCP and Daemon should be concerned with the Host - Host Protocol and the whole Multics process side of the street. At the very least, further experience suggests that it would be quite desirable to avoid waking up the Daemon on receipt of allocate messages, and allow the IMP DIM to handle them--preferably at interrupt time. In turn, this suggest that an integrated view of data bases should be taken. (Working set size considerations also support merger of data bases.) With more integration between IMP DIM and NCP it might also become possible to revise buffering strategies along the lines recently suggested by Bob Daley: take the view that there are two kinds of sockets (Telnet and "other") and adopt a Multics TTY DIM-like posture toward the Telnet sockets--i.e., knowing that they are for interactive messages, go to many small buffers and arrange their management to lend itself to greater interrupt-time work. (Note that even if this particular approach is not adopted, buffering strategies are suspect from the efficiency point of view if only because they are also an initial implementation-- and not widely understood.)

None of the above points has gone much beyond the interesting idea stage, although it is clear that some alterations of the interface along these lines will be necessary to improve efficiency. Another point at issue in this area, however, is that of certainly functional deficiencies. In particular, the current setup does not allow the aborting of queued writes in the IMP DIM when the Daemon or a server process detects a "quit". Also, inconsistencies should be resolved on such issues as the byte size associated with a socket and the handling of multiple IMP messages when reading. Fairly firm plans have been in hand for some time on the "reinterfacing" topic for some, but not all, of the functional deficiencies; implementation has, however, continually been pushed back owing to the existence of more pressing problems.

and for allowing the IMP DIM to manage allocations at least on the call side

3. Server Process IO

A topic which merits further investigation (even though it's currently quite "blue sky") is the possibility of using the standard ring 4 TTY DIM code in the server process in order to benefit from page-sharing, and appropriately cause calls to ring 0 to be directed to Network software. As this depends on the NCP's being in ring 0 and on the TTY DIM's (reportedly planned) move of canonicalization to ring 4, the topic is not of immediate impact, but could pay off strongly in the long run. An interesting implication which should also be noted is that such an employment of standard system code would be a useful step toward making the Network more a part of Multics and not merely a "wart", as Corby so aptly phrased it.

4. Daemon Elimination

As a final, long-range topic, it should be noted that Jerry Saltzer argues that in principle the Network Daemon could be eliminated entirely, with its connection-initiating and other control functions shifted to the Answering Service. Whether this would be workable in practice is moot.