Network Working Group   D. C. Walden
Request for Comments: 62   BBN Inc.
Supercedes NWG/RFC #61   3 August 1970

A System for Interprocess Communication in a Resource Sharing Computer Network

1. Introduction

If you are working to develop methods of communications within a computer network, you can engage in one of two activities. You can work with others, actually constructing a computer network, being influenced, perhaps influencing your colleagues. Or you can construct an intellectual position of how things should be done in an ideal network, one better than the one you are helping to construct, and then present this position for the designers of future networks to study. The author has spent the past two years engaged in the first activity. This paper results from recent engagement in the second activity.

"A resource sharing computer network is defined to be a set of autonomous, independent computer systems, interconnected so as to permit each computer system to utilize all of the resources of the other computer systems much as it would normally call a subroutine." This definition of a network and the desirability of such a network is expounded upon by Roberts and Wessler in [9].

The actual act of resource sharing can be performed in two ways: in an ad hoc manner between all pairs of computer systems in the network; or according to a systematic network-wide standard. This paper develops one possible network-wide system for resource sharing.

I believe it is natural to think of resources as being associated with processes<1> and available only through communication with these processes. Therefore, I view the fundamental problem of resource sharing to be the problem of interprocess communication. I also share with Carr, Crocker, and Cerf [2] the view that interprocess communication over a network is a subcase of general interprocess communication in a multi-programmed environment.

These views have led me to perform a two-part study. First, a set of operations enabling interprocess communication within a single time-sharing system is constructed. This set of operations eschews many of the interprocess communications techniques currently in use within time-sharing systems -- such as communication through shared memory -- and relies instead on techniques that can be easily generalized to permit communication between remote processes. The second part of the study presents such a generalization. The application of this generalized system to the ARPA Computer Network [9] is also discussed.

The ideas enlarged upon in this paper came from many sources. Particularly influential were -- 1) an early sketch of a Host protocol for the ARPA Network by S. Crocker of UCLA and W. Crowther of Bolt Beranek and Newman Inc. (BBN); 2) Ackerman and Plummer's paper on the MIT PDP-1 time-sharing system [1]; and 3) discussions with W. Crowther and R. Kahn of BBN about Host protocol, flow control, and message routing for the ARPA Network. Hopefully, there are also some original ideas in this note. I alone am responsible for the collection of all of these ideas into the system described herein, and I am therefore responsible for any inconsistencies or bugs in the system.

It must be emphasized that this paper does not represent an official BBN position on Host protocol for the ARPA Computer Network.

2. A System for Interprocess Communication within a Time-Sharing System

This section describes a set of operations enabling interprocess communication within a time-sharing system. Following the notation of [10], I call this interprocess communication facility an IPC. As an aid to the presentation of this IPC, a model for a time-sharing system is described; this model is then used to illustrate the use of the interprocess communication operations.

The model time-sharing has two pieces: the monitor and the processes. The monitor performs such functions as switching control from one process to another process when a process has used "enough" time, fielding hardware interrupts, managing core and the swapping medium, controlling the passing of control from one process to another (i.e., protection mechanisms), creating processes,caring for sleeping processes, and providing to the processes a set of machine extending operations (often called Supervisor or Monitor Calls). The processes perform the normal user functions (user processes) as well as the functions usually thought of as being supervisor functions in a time-sharing system (systems processes) but not performed by the monitor in the current model. A typical system process is the disc handler or the file system. System processes is the disc handler or the file system. System processes are probably allowed to execute in supervisor mode, and they actually execute I/O instructions and perform other privileged operations that user processes are not allowed to perform. In all other ways, user and system processes are identical. For reasons of efficiency, it may be useful to think of system processes as being locked in core.

Although they will be of concern later in this study, protection considerations are not my concern here: instead I will assume that all of the processes are "good" processes which never made any mistakes. If the reader needs a protection structure to keep in mind while he reads this note, the capability system developed in [1][3][7][8] should be satisfying.

Of the operations a process can call on the monitor to perform, six are of particular interest for providing a capability for interprocess communication.

RECEIVE. This operation allows a specified process to send a message to the process executing the RECEIVE. The operation has four parameters: the port (defined below) awaiting the message -- the RECEIVE port; the port a message will be accepted from -- the SEND port; a specification of the buffer available to receive the message; and a location to transfer to when the transmission is complete -- the restart location.

SEND. This operation sends a message from the process executing the SEND to a specified process. It has four parameters: a port to send the message to -- the RECEIVE port; the port the message is being sent from -- the SEND port; a specification of the buffer containing the message to be sent; and the restart location.

RECEIVE ANY. This operations allows any process to send a message to the process executing the RECEIVE ANY. The operation has four parameters: the port awaiting the message -- the RECEIVE port; a specification of the buffer available to receive the message; a restart location; and a location where the port which sent the message may be noted.

SEND FROM ANY. This operation allows a process to send a message to a process able to receive a message from any process. It has the same four parameters as SEND. (The necessity for this operation will be explained much later).

SLEEP. This operation allows the currently running process to put itself to sleep pending the completion of an event. The operation has one optional parameter, an event to be waited for. An example event is the arrival of a hardware interrupt. The monitor never unilaterally puts a process to sleep as a result of the process executing one of the above four operations; however, if a process is asleep when one of the above four operations is satisfied, the process is awakened.

UNIQUE. This operation obtains a unique number from the monitor.

A port is a particular data path to a process (a RECEIVE port) or from a process (a SEND port), and all ports have an associated unique port number which is used to identify the port. Ports are used in transmitting messages from one process to another in the following manner. Consider two processes, A and B, that wish to communicate. Process A executes a RECEIVE to port N from port M. Process B executes a SEND to port N from port M. The monitor matches up the port numbers and transfers the message from process B to process A. As soon as the buffer has been fully transmitted out of process B, process B is restarted at the location specified in the SEND operation. As soon as the message is fully received at process A, process A is restarted at the location specified in the RECEIVE operation. Just how the processes come by the correct port numbers with which to communicate with other processes is not the concern of the monitor -- this problem is left to the processes.

When a SEND is executed, nothing happens until a matching RECEIVE is executed. Somewhere in the monitor there must be a table of port numbers associated with processes and restart locations. The table entries are cleared after each SEND/RECEIVE match is made. If a proper RECEIVE is not executed for some time, the SEND is timed out after a while and the SENDing process is notified. If a RECEIVE is executed but the matching SEND does not happen for a long time, the RECEIVE is timed out and the RECEIVing process is notified.

The mechanism of timing out "unused" table entries is of little fundamental importance, merely providing a convenient method of garbage collecting the table. There is no problem if an entry is timed out prematurely, because the process can always re-execute the operation. However, the timeout interval should be long enough so that continual re-execution of an operation will cause little overhead.

A RECEIVE ANY never times out, but may be taken back using a supervisor call. A message resultant from a SEND FROM ANY is always sent immediately and will be discarded if a proper receiver does not exist. An error message is not returned and acknowledgment, if any, is up to the processes. If the table where the SEND and RECEIVE are matched up ever overflows, a process originating a further SEND and RECEIVE is notified just as if the SEND or RECEIVE timed out.

The restart location is an interrupt entrance associated with a pseudo interrupt local to the process executing the operation specifying the restart location. If the process is running when then event causing the pseudo interrupt occurs (for example, a message arrives satisfying a pending RECEIVE), the effect is exactly as if the hardware interrupted the process and transferred control to the restart location. Enough information is saved for the process to continue execution at the point it was interrupted after the interrupt is serviced. If the process is asleep, it is readied and the pseudo interrupt is saved until the process runs again and the interrupt is then allowed. Any RECEIVE or RECEIVE ANY message port may thus be used to provide process interrupts, event channels, process synchronization, message transfers, etc. The user programs what he wants.

It is left as an exercise to the reader to convince himself that the monitor he is saddled with can be made to provide the six operations described above -- most monitors can since these are only additional supervisor calls.

An example. Suppose that our model time-sharing system is initialized to have several processes always running. Additionally, these permanent processes have some universally known and permanently assigned ports<2>. Suppose that two of the permanently running processes are the logger-process and the teletype-scanner-process. When the teletype-scanner-process first starts running, it puts itself to sleep awaiting an interrupt from the hardware teletype scanner. The logger-process initially puts itself to sleep awaiting a message from the teletype-scanner-process via well-known permanent SEND and RECEIVE ports. The teleype-scanner-process keeps a table indexed by teletype number, containing in each entry a pair of port numbers to use to send characters from that teletype to a process and a pair of port numbers to use to receive characters for that teletype from a process. If a character arrives (waking up the teletype-scanner- process) and the process does not have any entry for that teletype, it gets a pair of unique numbers from the monitor (via UNIQUE) and sends a message containing this pair of numbers to the logger-process using the ports for which the logger-process is known to have a RECEIVE pending. The scanner-process also enters the pair of numbers in the teletype table, and sends the character and all future characters from this teletype to the port with the first number from the port with the second number. The scanner-process must also pass a second pair of unique numbers to the logger-process for it to use for teletype output and do a RECEIVE using these port numbers. When the logger-process receives the message from the scanner-process, it starts up a copy of what SDS 940 TSS [6] users call the executive<3>, and passes the port numbers to this copy of the executive, so that this executive-process can also do its inputs and outputs to the teletype using these ports. If the logger-process wants to get a job number and password from the user, it can temporarily use the port numbers to communicate with the user before it passes them on to the executive. The scanner-process could always use the same port numbers for a particular teletype as long as the numbers were passed on to only one copy of the executive at a time.

It is important to distinguish between the act of passing a port from one process to another and the act of passing a port number from one process to another. In the previous example, where characters from a particular teletype are sent either to the logger-process or an executive-process by the teletype-scanner-process, the SEND port always remains in the teletype-scanner-process while the RECEIVE port moves from the logger-process to the executive process. On the other hand, the SEND port number is passed between the logger-process and the executive-process to enable the RECEIVE process to do a RECEIVE from the correct SEND port. It is crucial that, once a process transfers a port to some other process, the first process no longer use the port. We could add a mechanism that enforces this. The protected object system of [9] is one such mechanism. Using this mechanism, a process executing a SEND would need a capability for the SEND port and only one capability for this SEND port would exist in the system at any given time. A process executing a RECEIVE would be required to have a capability for the RECEIVE port, and only one capability for this RECEIVE port would exist at a given time. Without such a protection mechanism, a port implicitly moves from one process to another by the processes merely using the port at disjoint times even if the port’s number is never explicitly passed.

Of course, if the protected object system is available to us, there is really no need for two port numbers to be specified before a transmission can take place. The fact that a process knows an existing RECEIVE port number could be considered prima facie evidence of the process’ right to send to that port. The difference between RECEIVE and RECEIVE ANY ports then depends solely on the number of copies of a particular port number that have been passed out. A system based on this approach would clearly be preferable to the one described here if it was possible to assume that all autonomous time-sharing systems in a network would adopt this protection mechanism. If this assumption cannot be made, it seems more practical to require both port numbers.

Note that in the interprocess communication system (IPC) being described here, when two processes wish to communicate they set up the connection themselves, and they are free to do it in a mutually convenient manner. For instance, they can exchange port numbers or one process can pick all the port numbers and instruct the other process which to use. However, in a particular implementation of a time-sharing system, the builders of the system might choose to restrict the processes’ execution of SENDs and RECEIVEs and might forbid arbitrary passing around of ports and port numbers, requiring instead that the monitor be called (or some other special program) to perform these functions.

Flow control is provided in this IPC by the simple method of never starting data transmission resultant from a SEND from one process until a RECEIVE is executed by the receiver. Of course, interprocess messages may also be sent back and forth suggesting that a process stop sending or that space be allocated.

Generally, well-known permanently-assigned ports are used via RECEIVE ANY and SEND FROM ANY. The permanent ports will most often be used for starting processes and, consequently, little data will be sent via them. If a process if running (perhaps asleep), and has a RECEIVE ANY pending, then any process knowing the receive port number can talk to that process without going through loggers. This is obviously essential within a local time-sharing system and seems very useful in a more general network if the ideal of resource sharing is to be reached. For instance, in a resource sharing network, the programs in the subroutine libraries at all sites might have RECEIVE ANYs always pending over permanently assigned ports with well-known port numbers. Thus, to use a particular network resource such as a matrix manipulation hardware, a process running anywhere in the network can send a message to the matrix inversion subroutine containing the matrix to be inverted and the port numbers to be used for returning the results.

An additional example demonstrates the use of the FORTRAN compiler. We have already explained how a user sits down at his teletype and gets connected to an executive. We go on from there. The user is typing in and out of the executive which is doing SENDs and RECEIVEs. Eventually the user types RUN FORTRAN, and executive asks the monitor to start up a copy of the FORTRAN compiler and passes to FORTRAN as start up parameters the port numbers the executive was using to talk to the teletype. (This, at least conceptually, FORTRAN is passed a port at which to RECEIVE characters from the teletype and a port from which to SEND characters to the teletype.) FORTRAN is, of course, expecting these parameters and does SENDs and RECEIVEs via the indicated ports to discover from the user what input and output files the user wants to use. FORTRAN types INPUT FILE? to the user, who responds F001. FORTRAN then sends a message to the file-system-process, which is asleep waiting for something to do. The message is sent via well-known ports and it asks the file system to open F001 for input. The message also contains a pair of port numbers that the file-system process can use to send its reply. The file-system looks up F001, opens it for input, make some entries in its open file tables, and sends back to FORTRAN a message containing the port numbers that FORTRAN can use to read the file. The same procedure is followed for the output file. When the compilation is complete, FORTRAN returns the teletype port numbers (and the ports) back to the executive that has been asleep waiting for a message from FORTRAN, and then FORTRAN halts itself. The file-system-process goes back to sleep when it has nothing else to do<4>.

Again, the file-system process can keep a small collection of port numbers which it uses over and over if it can get file system users to return the port numbers when they have finished with them. Of course, when this collection of port numbers has eventually dribbled away, the file system can get some new unique numbers from the monitor.

3. A System for Interprocess Communication Between Remote Processes

The IPC described in the previous section easily generalizes to allow interprocess communication between processes at geographically different locations as, for example, within a computer network.

Consider first a simple configuration of processes distributed around the points of a star. At each point of the star there is an autonomous operating system<5>. A rather large, smart computer system, called the Network Controller, exists at the center of the star. No processes can run in this center system, but rather it should be thought of as an extension of the monitor of each of the operating systems in the network.

If the Network Controller is able to perform the operations SEND, RECEIVE, SEND FROM ANY, RECEIVE ANY, and UNIQUE and if all of the monitors in all of the time-sharing systems in the network do not perform these operations themselves but rather ask the Network Controller to perform these operations for them, then the problem of interprocess communication between remote processes if solved. No further changes are necessary since the Network Controller can keep track of which RECEIVEs have been executed and which SENDs have been executed and match them up just as the monitor did in the model time-sharing system. A networkwide port numbering scheme is also possible with the Network Controller knowing where (i.e., at which site) a particular port is at a particular time.

Next, consider a more complex network in which there is no common center point, making it necessary to distribute the functions performed by the Network Controller among the network nodes. In the rest of this section I will show that it is possible to efficiently and conveniently distribute the functions performed by the star Network Controller among the many network sites and still enable general interprocess communication between remote processes.

Some changes must be made to each of the four SEND/RECEIVE operations described above to adapt them for use in a distributed Network Controller. To RECEIVE is added a parameter specifying a site to which the RECEIVE is to be sent. To the SEND FROM ANY and SEND messages is added a site to send the SEND to although this is normally the local site. Both RECEIVE and RECEIVE ANY have added the provision for obtaining the source site of any received message. Thus, when a RECEIVE is executed, the RECEIVE is sent to the site specified, possibly a remote site. Concurrently a SEND is sent to the same site, normally the local site of the process executing the SEND. At this site, called the rendezvous site, the RECEIVE is matched with the proper SEND and the message transmission is allowed to take place from the SEND site to the site from whence the RECEIVE came.

A RECEIVE ANY never leaves its originating site and therein lies the necessity for SEND FROM ANY, since it must be possible to send a message to a RECEIVE ANY port and not have the message blocked waiting for a RECEIVE at the sending site. It is possible to construct a system so the SEND/RECEIVE rendezvous takes place at the RECEIVE site and eliminates the SEND FROM ANY operation, but in my judgment the ability to block a normal SEND transmission at the source site more than makes up for the added complexity.

At each site a rendezvous table is kept. This table contains an entry for each unmatched SEND or RECEIVE received at that site and also an entry for all RECEIVE ANYs given at that site. A matching SEND/RECEIVE pair is cleared from the table as soon as the match takes place. As in the similar table kept in the model time-sharing, SEND and RECEIVE entries are timed out if unmatched for too long and the originator is notified. RECEIVE ANY entries are cleared from the table when a fulfilling message arrives.

The final change necessary to distribute the Network Controller functions is to give each site a portion of the unique numbers to distribute via its UNIQUE operation. I'll discuss this topic further below.

To make it clear to the reader how the distributed Network Controller works, an example follows. The details of what process picks port numbers, etc., are only exemplary and are not a standard specified as part of the IPC.

Suppose that, for two sites in the network, K and L, process A at site K wishes to communicate with process B at site L. Process B has a RECEIVE ANY pending at port M.

```
                        SITE K                        SITE L

                        ______                        ______
                       /      \                      /      \
                      /        \                    /        \
                     /          \                  /          \
                    /            \                /            \
                   |              |              |              |
                   |   Process A  |              |   Process B  |
                   |              |              |              |
                    \            /                \            /
                     \          /      RECEIVE--> port M      /
                      \        /       ANY          \        /
                       \______/                      \______/
```

Process A, fortunately, knows of the existence of port M at site L and sends a message using the SEND FROM ANY operation from port N to port M. The message contains two port numbers and instructions for process B to SEND messages for process A to port P from port Q. Site K's site number is appended to this message along with the message's SEND port N.

```
                        SITE K                        SITE L

                        ______                        ______
                       /      \                      /      \
                      /        \                    /        \
                     /          \                  /          \
                    /            \                /            \
                   |              |              |              |
                   |   Process A  |              |   Process B  |
                   |              |              |              |
                    \   port N   /                \   port M   /
                     \          /--->SEND FROM --->\          /
                      \        /        ANY         \        /
                       \______/                      \______/

                                   to port M, site L

                                   containing K,N,P, & Q
```

Process A now executes a RECEIVE at port P from port Q. Process A specifies the rendezvous site to be site L.

```
                        SITE K                        SITE L

                        ______                        ______
                       /      \                      /      \
                      /        \                    /        \
                     /          \        Rendezvous/          \
                    /            \            table            \
                   |              |              |              |
                   |   Process A  |           ^  |   Process B  |
                   |              |           |  |              |
                    \   port P   /            |   \            /
                     \          /             |    \          /
                      \        / <--RECEIVE __/     \        /
                       \______/     MESSAGE          \______/

                                    to site L

                                    containing P, Q, & K
```

A RECEIVE message is sent from site K to site L and is entered in the rendezvous table at site L. At some other time, process B executes a SEND to port P from port Q specifying site L as the rendezvous site.

```
                        SITE K                        SITE L

                        ______                       ______
                       /      \                     /      \
                      /        \                   /        \
                     /          \       Rendezvous/          \
                    /            \           table            \
                   |              |             |              |
                   |   Process A  |             |   Process B  |
                   |              |             |              |
                    \   port P   /        <--------- port Q   /
                     \          /                 \          /
                      \        /        SEND       \        /
                       \______/                     \______/

                                        to site L

                                        containing P & Q
```

A rendezvous is made, the rendezvous table is cleared, and the transmission to port P at site K takes place. The SEND site number (and conceivably the SEND port number) is appended to the messages of the transmission for the edification of the receiving process.

```
                        SITE K                         SITE L

                        _______                       _______
                       /      \                      /      \
                      /        \                    /        \
                     /          \                  /          \
                    /            \                /            \
                   |              |              |              |
                   |   Process A  |              |   Process B  |
                   |              |              |              |
                    \   port P   /                \   port Q   /
                     \          /<--transmission<--\          /
                      \        /                    \        /
                       \______/   to port P, site K  \______/

                                  containing data and L
```

Process B may simultaneously wish to execute a RECEIVE from port N at port M.

Note that there is only one important control message in this system which moves between sites, the type of message that is called a Host/Host protocol message in [2]. This control message is the RECEIVE message. There are two other possible intersite control messages: an error message to the originating site when a RECEIVE or SEND is timed out, and the SEND message in the rare case when the rendezvous site is not the SEND site. There must also be a standard format for messages between ports. For example, the following:

```
         _________________           __________________      _____________
        | rendezvous site |  <6>    | destination site |    | source site |
        |-----------------|         |------------------|    |-------------|
        |    RECEIVE port |         |   RECEIVE port   |    | RECEIVE port|
        |-----------------|         |------------------|    |-------------|
        |    SEND port    |         |   SEND port      |    | SEND port   |
        |-----------------|         |------------------|    |-------------|
        |                 |         |   source site    |    |             |
        |                 |         |------------------|    |             |
        |                 |         |                  |    |             |
        |                 |         |                  |    |             |
        |                 |         |                  |    |             |
        |                 |         |                  |    |             |
        |     data        |         |     data         |    |   data      |
        |                 |         |                  |    |             |
        |                 |         |                  |    |             |
        |                 |         |                  |    |             |
        |                 |         |                  |    |             |
        |_________________|         |__________________|    |_____________|
         transmitted                 transmitted                 received
         by SEND                     by Network                  by RECEIVE
         process                     Controller                  process
```

In the model time-sharing system it was possible to pass a port form process to process. This is still possible with a distributed Network Controller.

Remember that, for a message to be sent from one process to another, a SEND to port M from port N and a RECEIVE at port M from port N must rendezvous, normally at the SEND site. Both processes keep track of where they think the rendezvous site is and supply this site as a parameter of appropriate operations. The RECEIVE process thinks it is the SEND site also. Since once a SEND and a RECEIVE rendezvous the transmission is sent to the source of the RECEIVE and the entry in the rendezvous table is cleared and must be set up again for each further transmission from N to M, it is easy for a RECEIVE port to be moved. If a process sends both the port numbers and the rendezvous site number to a new process at some other site which executes a RECEIVE using these same old port numbers and rendezvous site specification, the SENDer never knows the RECEIVEr has moved. It is slightly harder for a send port to move. However, if it does, the pair of port numbers that has been being used for a SEND and the original rendezvous site number are passed to the new site. The process at the new SEND site specifies the old rendezvous site with the first SEND from the new site. The RECEIVE process will also still think the rendezvous site is the old site, so the SEND and RECEIVE will meet at the old site. When they meet, the entry in the table at that site is cleared, and both the SEND and RECEIVE messages are sent to the new

SEND site just as if they had been destined for there in the first place. The SEND and RECEIVE then meet again at the new rendezvous site and transmission may continue as if the port had never moved. Since all transmissions contain the source site number, further RECEIVEs will be sent to the new rendezvous site. It is possible to discover that this special manipulation must take place because a SEND message is received at a site that did not originate the SEND message<7>. Note that the SEND port and the RECEIVE port can move concurrently.

Of course, all of this could have also been done if the processes had sent messages back and forth announcing any potential moves and the new site numbers.

A problem that may have occurred to the reader is how the SEND and RECEIVE buffers get matched for size. The easiest solution would be to require that all buffers have a common size but this is unacceptable since it does not easily extend to a situation where processes in autonomous operating systems are attempting to communicate. A second solution is for the processes to pass messages specifying buffer sizes. If this solution is adopted, excessive data sent from the SEND process and unable to fix into the RECEIVE buffer is discarded and the RECEIVE process notified. The solution has great appeal on account of its simplicity. A third solution would be for the RECEIVE buffer size to be passed to the SEND site with RECEIVE message and to notify the SEND process when too much data is sent or even to pass the RECEIVE buffer size on to the SEND process. This last method would also permit the Network Controller at the SEND site to make two or more SENDs out of one, if that was necessary to match a smaller RECEIVE buffer size.

The maintenance of unique numbers is also a problem when the processes are geographically distributed. Three solutions to this problem are presented here. The first possibility is for the autonomous operating systems to ask the Network Controller for the unique numbers originally and then guarantee the integrity of any unique numbers currently owned by local processes and programs using whatever means are at the operating system’s disposal. In this case, the Network Controller would provide a method for a unique number to be sent from one site to another and would vouch for the number’s identity at the new site. The second method is simply to give the unique numbers to the processes that are using them, depending on the non-malicious behavior of the processes to preserve the unique numbers, or if an accident should happen, the two passwords (SEND and RECEIVE port numbers) that are required to initiate a transmission. If the unique numbers are given out in a non-sequential manner and are reasonably long (say 32 bits), there is little danger. In the final method, a user identification is included in the port numbers and the individual operating systems guarantee the integrity of these identification bits. Thus a process, while not able to be sure that the correct port is transmitting to him, can be sure that some port of the correct user is transmitting. This is the so-called virtual net concept suggested by W. Crowther [2].<8>

A third difficult problem arises when remote processes wish to communicate, the problem of maintaining high bandwidth connections between the remote processes. The solution to this problem lies in allowing the processes considerable information about the state of an on-going transmission. First, we examine a SEND process in detail. When a process executes a SEND, the local portion of the Network Controller passes the SEND on to the rendezvous site, normally the local site. When a RECEIVE arrives matching a pending SEND, the Network Controller notifies the SEND process by causing an interrupt to the specified restart location. Simultaneously the Network Controller starts shipping the SEND buffer to the RECEIVE site. When transmission is complete, a flag is set which the SEND process can test. While a transmission is taking place, the process may ask the Network Controller to perform other operations, including other SENDs. A second SEND over a pair of ports already in the act of transmission is noted and the SEND becomes active as soon as the first transmission is complete. A third identical SEND results in an error message to the SENDing process. Next, we examine a RECEIVE process in detail. When a process executes a RECEIVE, the RECEIVE is sent to the rendezvous site. When data resultant from this RECEIVE starts to arrive at the RECEIVE site, the RECEIVE process is notified via an interrupt to the specified restart location. When the transmission is complete, a flag is set which the RECEIVE process can test. A second RECEIVE over the same port pair is allowed. A third results in an error message to the RECEIVE process. Thus, there is sufficient machinery to allow a pair of processes always to have both a transmission in progress and the next one pending. Therefore, no efficiency is lost. On the other hand, each transmission must be preceded by a RECEIVE into a specified buffer, thus continuing to provide complete flow control.

4. A Potential Application

Only one resource sharing computer network currently exists, the ARPA Computer Network. In this section, I discuss application of the system described in this paper to the ARPA Network [2][5][9].

The ARPA Network currently incorporates ten sites spread across the United States. Each site consists of one to three (potentially four) independent computer systems called Hosts and one communications computer system called an IMP. All of the Hosts at a site are directly connected to the IMP. The IMPs themselves are connected together by 50-kilobit phone lines (much higher rate lines are a potential), although each IMP is connected to only one to five other IMPs. The IMPs provide a communications subnet through which the Hosts communicate. Data is sent through the communications subnet in messages of arbitrary size (currently about 8000 bits) called network messages. When a network message is received by the IMP at the destination site, that IMP sends an acknowledgment, called a RFNM, to the source site.

A system for interprocess communication for the ARPA Network (let us call this IPC for ARPA) is currently being designed by the Network Working Group, under the chairmanship of S. Crocker of UCLA. Their design is somewhat constrained by the communications subnet [5]<9>. I would like to compare point-by-point IPC for ARPA with the one developed in this paper; however, such a comparison would first require description here, almost from scratch, of the current state of IPC for ARPA since very little up-to-date information about IPC for ARPA appears in the open literature [2]. Also, IPC for ARPA is quite complex and the working documents describing it now run to many hundred pages, making any description lengthy and inappropriate for this paper.<10> Therefore, I shall make only a few scattered comparisons of the two systems, the first of which are implicit in this paragraph.

The interprocess communication system being developed for the ARPA Network comes in several almost distinct pieces: The Host/IMP protocol, IMP/IMP protocol, and the Host/Host protocol. The IMPs have sole responsibility for correctly transmitting bits from one site to another. The Hosts have sole responsibility for making interprocess connections. Both the Host and IMP are concerned and take a little responsibility for flow control and message sequencing. Applications of the interprocess communication system described in this paper leads me to make a different allocation of responsibility. The IMP still continues to move bits from on site to another correctly but the Network Controller also resides in the IMP, and flow control is completely in the hands of the processes running in the Hosts, although using the mechanisms provided by the IMPs.

The IMPs provide the SEND, RECEIVE, SEND FROM ANY, RECEIVE ANY, and UNIQUE operations in slightly altered forms for the Hosts and also maintain the rendezvous tables, including moving of SEND ports when necessary. Putting these operations in the IMP requires the Host/Host protocol program to be written only once, rather than many times as is currently being done in the ARPA Network. It is perhaps useful to step through the five operations again.

SEND. The Host gives the IMP a SEND port number, a RECEIVE port number, the rendezvous site, and a buffer specification (e.g., start and end, or beginning and length). The SEND is sent to the rendezvous site IMP, normally the local IMP. When a matching RECEIVE arrives at the local IMP, the Host is notified of the RECEIVE port of the just arrived message. This port number is sufficient to identify the SENDing process, although a given operating system may have to keep internal tables mapping this port number into a useful internal process identifier. Simultaneously, the IMP begins to ask the Host for specific pieces of the SEND buffer, sending these pieces as network messages to the destination site. If a RFNM is not received for too long, implying a network message has been lost in the network, the Host is asked for the same data again and it is retransmitted.<11> Except for the last piece of a buffer, the IMP requests pieces from the Host which are common multiplies of the word size of the source Host, IMP, and destination Host. This avoids mid-transmission word alignment problems.

RECEIVE. The Host gives the IMP a SEND port, a RECEIVE port, a rendezvous site, and a buffer description. The RECEIVE message is sent to the rendezvous site. As the network messages making up a transmission arrive for the RECEIVE port, they are passed to the Host along with RECEIVE port number (and perhaps the SEND port number), and an indication to the Host where to put this data in its input buffer. When the last network message of the SEND buffer is passed into the Host, it is marked accordingly and the Host can then detect this. (It is conceivable that the RECEIVE message could also allocate a piece of network bandwidth while making its network traverse to the rendezvous site.)

RECEIVE ANY. The Host gives the IMP a RECEIVE port and a buffer descriptor. This works the same as RECEIVE but assumes the local site to be the rendezvous site.

SEND FROM ANY. The Host gives the IMP RECEIVE and SEND ports, the destination site, and a buffer descriptor. The IMP requests and transmits the buffer as fast as possible. A SEND FROM ANY for a non-existent port is discarded at the destination site.

In the ARPA Network, the Hosts are required by the IMPs to physically break their transmissions into network messages, and successive messages of a single transmission must be delayed until the RFNM is received for the previous message. In the system described here, since RFNMs are tied to the transmission of a particular piece of buffer and since the Hosts allow the IMPs to reassemble buffers in the Hosts by the IMP telling the Host where to put each buffer piece then pieces of a single buffer can be transmitted in parallel network messages and several RFNMs can be outstanding simultaneously. This enables The Hosts to deal with transmissions of more natural sizes and higher bandwidth for a single transmission.

For additional efficiency, the IMP might know the approximate time it takes for a RECEIVE to get to a particular other site and warn the Host to wake up a process shortly before the arrival of a message for that process is imminent.

5. Conclusion

Since the system described in this paper has not been implemented, I have no clearly demonstrable conclusions nor any performance reports. Instead, I conclude with four openly subjective claims.

1) The interprocess communication system described in Section 2 is simpler and more general than most existing systems of equivalent power and is more powerful than most intra time-sharing system communication systems currently available.

2) Time-sharing systems structured like the model in Section 2 should be studied by designers of time-sharing systems who may see a computer network in their future, as structure seems to enable joining a computer network with a minimum of difficulty.

3) As computer networks become more common, remote interprocess communication systems like the one described in Section 3 should be studied. The system currently being developed for ARPA is a step in the wrong direction, being addressed, in my opinion, more to communication between monitors than to communication between processes and consequently subverting convenient resource sharing.

4) The application of the system as described in Section 4 is much simpler to implement and more powerful than the system currently being constructed for the ARPA Network, and I suggest that implementation of my method be seriously considered for adoption by the ARPA Network.

<Footnotes>

1. Almost any of the common definitions of a process would suit the needs of this paper.

2. Or perhaps there is only one permanently known port, which belongs to a directory-process that keeps a table of permanent-process/well-know-port associations.

3. That program which prints file directories, tells who is on other

teletypes, runs subsystems, etc.

4. The reader should have noticed by now that I do not like to think of a new process (consisting of a new conceptual copy of a program) being started up each time another user wishes to use the program. Rather, I like to think of the program as a single process which knows it is being used simultaneously by many other processes and consciously multiplexes among the users or delays service to users until it can get around to them.

5. I use operating system rather than time-sharing system in this section to point up the fact that the autonomous systems at the network nodes may be either full blown time-sharing systems in their own right, and individual process in a larger geographically distributed time-sharing system, or merely autonomous sites wishing to communicate.

6. For a SEND FROM ANY message, the rendezvous site is the destination site.

7. For readers familiar with the once-proposed re-connection scheme for the ARPA Network, the above system is simple, comparatively, because there are no permanent connections to break and move; that is, connections only exist fleetingly in the system described here and can therefore be remade between any pair of processes which at any time happen to know each other’s port numbers and have some clue where they each are.

8. Crowther says this is not the virtual net concept.

9. As one of the builders of the ARPA communications subnet, I am partially responsible for these constraints.

10. The reader having access to the ARPA working documents may want to read Specifications for the Interconnection of a Host to an IMP, BBN Report No. 1822; and ARPA Network Working Group Notes #36, 37, 38, 39, 42, 44, 46, 47, 48, 49, 50, 54, 55, 56, 57, 58, 59, 60.

11. This also allows messages to be completely thrown away by the IMP subnet it that should ever be useful.

[REFERENCES]

1. Ackerman, W., and Plummer, W. An implementation of a multi-processing computer system. Proc. ACM Symp. on Operating System Principles, Gatlinsburg, Tenn.,

Oct. 1-4, 1967.

2. Carr, C. Crocker, S., and Cerf, V. Host/Host communication protocol in the ARPA network. Proc. AFIPS 1970 Spring Joint Comput. Conf., Vol. 36, AFIPS Press, Montvale, N.J., pp. 589-597.

3. Dennis, J., and VanHorn, E. Programming semantics for multiprogrammed computations. Comm. ACM 9, 3 (March, 1966), 143-155.

4. Hansen, P.B. The nucleus of a multiprogramming system. Comm. ACM 13, 4 (April, 1970), 238-241, 250.

5. Heart, F., Kahn, R., Ornstein, S., Crowther, W., and Walden, D. The interface message processor for the ARPA computer network. Proc. AFIPS 1970 Spring Joint Comput. Conf., Vol. 36, AFIPS Press, Montvale, N.J., pp. 551-567.

6. Lampson, B. SDS 940 Lectures, circulated informally.

7. _______. An overview of the CAL time-sharing system. Computer Center, University of California, Berkeley, Calif.

8. _______. Dynamic protection structures. Proc. AFIPS 1969 Fall Joint Comput. Conf., Vol. 35, AFIPS Press, Montvale, N.J., pp. 27-38.

9. Roberts, L., and Wessler, B. Computer network development to achive resource sharing. Proc. AFIPS 1970 Spring Joint Comput. Conf., Vol. 36, AFIPS Press, Monvale, N.J., pp. 543-549.

10. Spier, M., and Organick, E. The MULTICS interprocess communication facility. Proc. ACM Second Symp. on Operating Systems Principles, Princeton University, Oct. 20-22, 1969.

Author’s Address

D. C. Walden
Bolt Bernakek and Newman, Inc.
Cambridge, Massachusetts