Behavior Research Methods & Instrumentation
1974, Vol. 6, No. 2, 139-143

A decentralized computer network for supervision of multiple psychological laboratories*

KARL W. SCHOLZ and HENRY HALFF
University of Illinois, Champaign, Illinois 61820

The computer network described was designed to optimize the use of a number of independent minicomputers as a single integrated system for process control of several research laboratories. The implementation of the network required both the development of a special purpose interprocessor interface and the design of a software support system to direct network operations. The network was designed according to the familiar "star" configuration, with the exception that the central computer in the star does not exercise exclusive supervisory control over the system. Rather, each computer on the system "sees" the central computer as a peripheral similar in characteristics to a medium speed I/O device. This design allows each computer to serve not only as a node in the network, but also as an entirely independent process controller.

Doubtless many directors of computerized laboratories find that the direction of growth of any laboratory is not only the consequence of growing demands but is also a consequence of the specific existing resources. Although the growing demands in themselves might justify adoption of an entirely new laboratory automation scheme, financial considerations and investment in existing equipment frequently dictate the necessity for compromise.

The Illinois laboratory was faced with such a situation. The existing IBM 1800 was shared by two user communities, one using the machine for digitizing analog physiological data, and the other using the process control capabilities of the machine to control data acquisition in real time. As each group's demand for computer time increased, the necessity for a corresponding increase in the capacity of the system became apparent. One logical alternative might have been the purchase of separate computers to satisfy the needs of each user community, but this alternative clearly involved prohibitive expense. Furthermore, other researchers in the department who owned small special purpose minicomputer installations expressed a desire to interface their machines to the 1800 in order to gain access to its peripheral equipment. In view of these considerations, we finally decided to resolve the problem by designing a single unified computer network centered on the 1800.

Our design effort was directed toward satisfying the following objectives: (1) permit simultaneous time sharing of the 1800 by as many users as possible; (2) permit maximum use of the existing 1800 system, including 16K of core, two disks, mag tape, flatbed digital plotter, and assorted printers and readers. (3) keep interfacing costs and complexities to a minimum; (4) avoid excessive dependence of any single component of the network on any other component; and (5) provide expandability to accommodate future users.

The results of our efforts will be described in two sections, the first dealing with the hardware design of an interprocessor controller and the second dealing with the design of the system support software.

*Requests for reprints should be sent to Karl W. Scholz, Department of Psychology, University of Illinois, Champaign, Illinois 61820.

THE INTERPROCESSOR CONTROLLER

The interprocessor controller was designed to decentralize the control of the network by allowing all I/O transfer between the central processor (CP) and the peripheral processors (PPs) to be initiated by the PPs rather than by the CP itself. Communication between the CP and the controller uses two of the CP's data channels, one for transfer in each direction. Communication between the controller and each of the PPs was designed to accommodate either data channel transmission or interrupt-driven direct program control transfer, whichever best conforms to the configuration of a particular minicomputer.

As Fig. 1 indicates, the controller consists of two major elements, a seven-channel input multiplexor and a port-address controller. The input multiplexor can select one of the seven 16-bit wide input ports for presentation to the CP's input channel. The port-address controller processes all the control signals for both the PPs and the CP.

Port-address selection is under the control of the CP. Whenever the system is idling (i.e., no transmission is in progress), the CP selects Port-Address O. Whenever Address O is selected, the port-address controller assembles a 14-bit status word and presents it continuously on the CP's input bus. The status word consists of two bits for each peripheral processor. The

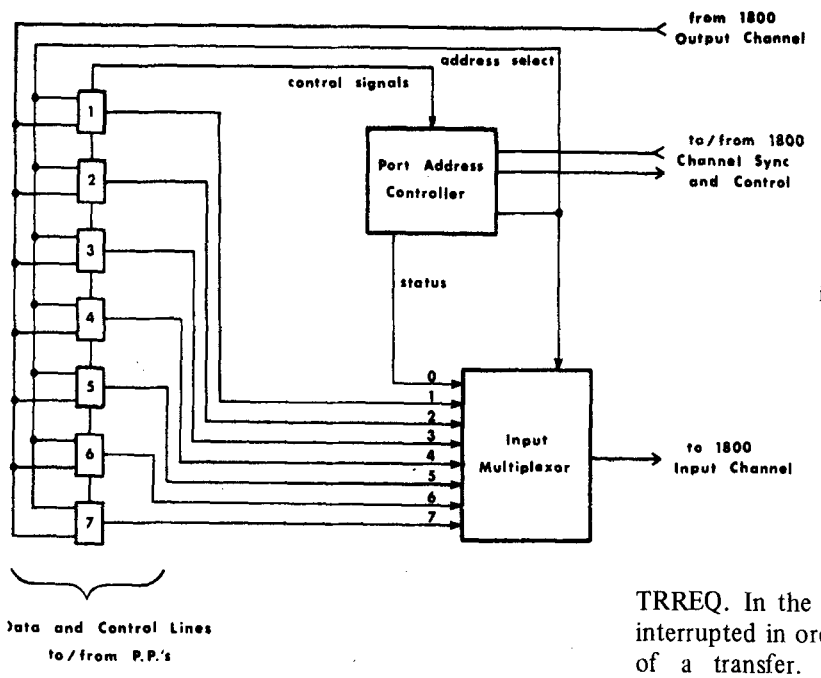

Fig. 1. Block diagram of the interprocessor controller.

transfer request bit (TRREQ) indicates a request for transfer and the transfer direction bit (TRDIR) indicates the intended direction of transfer (to or from the CP). Whenever the CP software returns to idle, the status word is checked, and processing initiated for any nonzero TRREQ bits that were detected. The first step in processing a transfer request requires initializing either the input data channel or the output data channel, depending on the status of the particular TRDIR bit. Then, the address of the PP initiating the request is transferred to the controller. Upon receiving this address, the port-address controller establishes a data path from the selected processor to the appropriate data channel on the CP. Data transfer then continues to completion under control of the CP data channel, leaving the CP free to continue background processing.

It should be emphasized that, with respect to the CP, the I/O transfer is initiated without the use of interrupts. Rather, the initiation of any I/O transfer is contingent on the CP polling the status word from the controller. Thus, during periods of time when the CP is devoting its full resources to some processing task which is unrelated to interprocessor communication, it cannot be disturbed in any way by peripheral processor activity.

From the point of view of the PPs, I/O transfer appears much as it would for any other I/O device. If the PP wishes to transmit data to the CP, it sets first the first data word, secondly the TRDIR line, and finally the TRREQ line. Once the CP has recognized the TRREQ, the CP input channel will process the data word (or byte) and then acknowledge receipt by way of an acknowledge line to the PP. Typically, the acknowledge line will interrupt the PP, causing it to fetch the next data word. This "handshake" will continue until the transfer process is complete, or until the PP drops TRREQ. In the latter case, the CP may optionally be interrupted in order to signal the requested termination of a transfer. Note that this termination request interrupt is the only interrupt generated by the controller for the CP.

If a PP desires to receive data from the CP, a similar sequence of events is initiated. First, the PP sets the TRDIR line indicating that it wishes to receive data, and then it sets the TRREQ line. Upon recognition of the request, the CP selects its output data channel, and data transmission to the PP continues until either the CP exhausts the data table which is being transmitted, or until the PP lowers its TRREQ line.

In summary, the interprocessor controller design satisfies the objective of hardware simplicity, thus reducing costs to a minimum, and yet maintains flexibility by acting as the heart of an expansible decentralized computer network.

SOFTWARE DESIGN FOR SUPPORT OF THE NETWORK

The decentralized network supervisor (DNS) operating system was designed to integrate the supervision of the interprocessor controller with supervision of all remaining CP I/O devices. The system, which executes in a 16K CPU, includes a 3K resident monitor and 13K of dynamically allocated buffers. Nonresident system functions are executed from two 320-word overlay areas, while nonresident utilities are executed when needed from a 3K partition within dynamically allocated core. Thus, 10K of buffer storage is guaranteed, while the remaining 3K is optionally available whenever the services of a nonresident utility are not required.

The I/O System

The major strength of DNS lies in its input/output system, which is to some extent modeled after the

IBM MPX I/O structure. A uniform calling sequence is used for all DNS I/O routines. As Fig. 2 indicates, each routine is called with the address of a vector containing four arguments. The first argument is a busy parameter which is set to nonzero to indicate that the I/O is in progress. This argument is returned to zero to signal the completion of the requested operation. The second argument is the link word used for I/O queue management which will be explained below. The third argument is a numeric parameter which indicates the desired I/O operation (e.g., read, write). Finally, the fourth argument is the address of the I/O buffer which is to be used in the I/O operation.

One word in the device table associated with each I/O routine is used as a listhead for the particular I/O device queue. When the I/O routine is first called (i.e., the device was previously nonbusy), IOQUE is set to the address of the I/O list (i.e., the address of the four arguments in the call which was passed to the I/O routine). The link word in the I/O list is then set to zero, and the busy parameter is set to nonzero. Next, the I/O operation is initiated, and immediately control returns to the calling program. If the same I/O routine is called while the device is still busy, IOQUE will naturally still be nonzero. The routine responds to a nonzero IOQUE by investigating the LINK word in the list to which IOQUE points. If LINK is found to be zero, it is set to the address of the new I/O list. If LINK is nonzero, it is assumed to be pointing to yet another I/O list, so its LINK word is consulted. Thus the process of traversing a linked list is continued until a zero LINK word is found, and this word is set to the address of the new I/O list.

In summary, the I/O queue for each device consists of a series of linked nodes, where each node is a four-word block in a calling program. As the I/O operation specified by a given list is completed, this four-word block is removed from the linked chain by setting IOQUE to the value of LINK. In addition, the first word of the block is set to zero to indicate completion. This method of I/O management has the obvious advantage of having no intrinsic limit (other than the size of available core) on the size of the queues for any device.

Dynamic Storage Allocation

The second important feature of DNS is its dynamic storage allocation. Since the primary function of the system is to supervise data transfer among various PPs and CP I/O devices, heavy demands are placed on the system's data buffering capabilities. All of dynamically allocated core is divided into a series of 160 82-word buffers. The in-use status of each buffer is recorded by a single bit in a 160-bit (10-word) block reservation table (BRT). Buffers are assigned either singly (for devices such as card reader punch or tyewriters) or in groups of four (for disk or tape I/O). Allocation of either an 82-word or a 328-word block is performed by searching the BRT for either a single zero-bit or four consecutive zero-bits, whichever is appropriate. Once a block has been located using the BRT, the word count of the block is written into its actual first word, and the appropriate BRT bits are set *on* indicating the block is in use. When a block is returned to available storage, its first word is interrogated to determine its size, and the appropriate bit or bits in the BRT are reset to indicate that the block is available for reassignment. All programs, including system routines, use the dynamic storage allocation system for all data storage, thus insuring a highly efficient utilization of the limited available core memory.

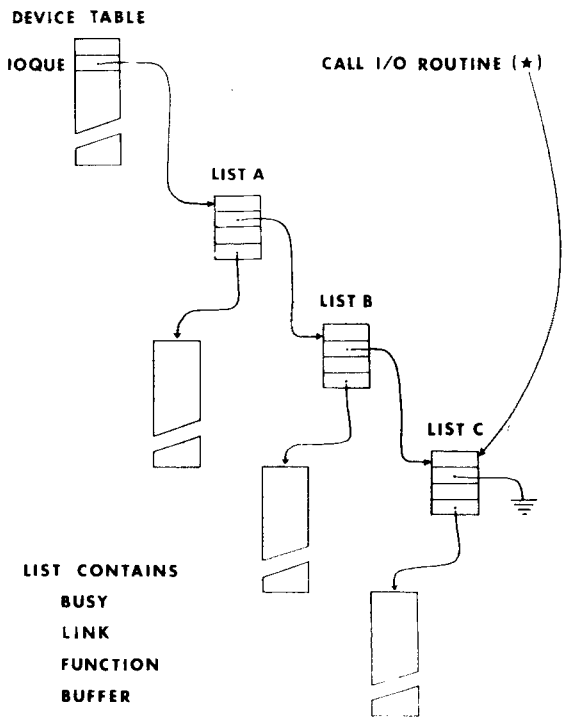

Fig. 2. An overview of the input/output queuing system.

Interprocessor Communication

A major purpose of the system is to allow the PPs to communicate with the peripheral devices interfaced to the CP. One of the simplest approaches might be to make the CP as transparent as possible to the PPs by attempting to simulate a direct interface between the peripheral devices and each PP. Although this approach has much to offer in terms of simplicity, it tends to lead to inefficient use of the system in many cases. For example, the simple operation of transferring a file from magnetic tape to the disk would involve reading each block from tape, transmitting the data to the PP, reading it back from the PP, and finally writing the data on the disk. In order to avoid such inefficient use of the system, the PP must be able to take advantage of some of the processing capacity of the CP. Our specific approach is to use the CP to execute data transfer between devices on a schedule determined by the PP.

The basic operational unit of the system is the transfer request issued by the PP. A transfer request

TRANSFER REQUEST FORMAT: ID (SOURCE, DESTINATION)
WORD COUNT

SOURCE/DESTINATION:

EXTERNAL: DISK
CARD
TAPE
KEYBOARD
PERIPHERAL PROCESSOR

INTERNAL: CONTROL BUFFER
DATA BUFFER
STATUS BUFFER

CONTROL BLOCK	DATA BLOCK	STATUS BLOCK
BLOCK IDENT	BLOCK IDENT	DEVICES AVAILABLE
BLOCK COUNT	BLOCK COUNT	CURRENT REQUEST POINTER
REQUEST IDENT REQUEST COUNT (DATA) : : REQUEST IDENT REQUEST COUNT (DATA) : : ETC	DATA	CURRENT REQUEST STATUS

Fig. 3. Format of transfer request and special blocks.

consists of an identifier specifying a source and a destination, and a word count specifying the amount of data to be transferred (see Fig. 3). Both the source and destination may be the PP requesting the transfer, any of the I/O devices, or one of three special purpose buffers or "blocks" assigned to the PP by the CP. (The function of these three blocks will be outlined below.) In honoring any transfer request, the CP reads the specified number of words from the source and writes them on the destination. Within the dynamic allocation scheme detailed above, the transfer operation is quite simple. Buffers are allocated as the source presents the information to the CP, the data are read into the buffers, written to the destination, and the buffer is returned to the pool of available space as the destination receives the data.

There are two ways in which a PP can file a transfer request with the CP. The simplest way is by directly initiating the request. To directly initiate a transfer request, the PP obtains the attention of the CP by appropriate manipulation of the TRDIR and TRREQ lines. The CP then reads the identifier and count words from the PP and initiates the specified transfer. Alternately, a transfer request may be initiated through the use of one of the special blocks mentioned above.

The CP maintains three blocks for each PP. The first of these, the control block, is used by the PP to queue transfer requests. A PP adds one or more requests to a queue by directly initiating a transfer from any source (usually the PP) to the control block. That is, the control block serves as the destination for a list of requests to be queued. In addition, small amounts of data transferred by a control block request may be retained in the control block directly following the request. If the CP encounters a request in which the control block is designated as the source, the appropriate amount of data is transferred directly from the locations following that request in the block.

The status block is the second of the three areas allocated to each PP. The status block contains information on the status of certain external devices, and on the status of certain external devices. Specifically, the status block will contain the identifier and count of the current request, and any information relating to completion, cancellation, or abortion of the request. Requests may be cancelled by a transfer request specifying the status block as the destination. Also the status block will be unconditionally set to the PP on any error condition. The status block also contains the status of certain external devices such as the card reader/punch and magnetic tape drive. These devices must be disposed to a PP before that PP can use them. By transferring information to or from the status block, the PP may change or determine the status of these devices.

The last special area which the CP maintains for each PP is a data block consisting of the most recent block of data transferred by a request from the PP. Thus, if the amount of data transferred is less than one buffer, the data block will contain all of the data transferred. The data block may be used as a source or destination to facilitate transfer of the same data to multiple devices.

The following example is presented to clarify the use of the three blocks. Suppose a PP requires the data from a deck of cards which will be read into the CP. To initiate the reading process, the PP must request that the card reader be allocately appropriately, tell the operator which deck of cards to load and then begin reading. The initialization procedure may be accomplished by queuing four transfer requests in the control block as illustrated in Fig. 4. The first request asks the CP for the card reader through a transfer from the control block to the status block. The second request transfers the message to the operator from the control block to the

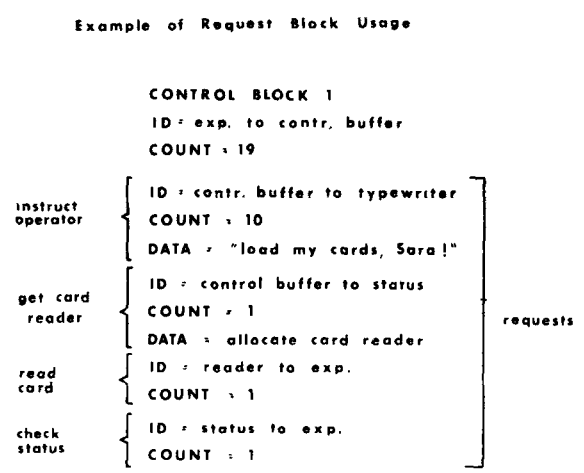

Fig. 4. Example of control block use to initialize a card reading operation.

console typewriter. The third request transfers a card image from the card reader to the PP, and the fourth request feeds the status block to the PP as a check on the operation. Once these requests have been placed in the control block, the PP will simply wait until the appropriate data and status information are returned from the CP The PP can then request reading of the second card.

In summary, the software system gives each PP a flexible system for initiating and scheduling data transfers between a variety of devices. By giving the PP the ability to initiate, queue, and monitor the transfer of data betwen any two devices, the system provides capabilities for the disposition of data which normally exceed the capacity of an independent minicomputer installation.

Behavior Research Methods & Instrumentation
1974, Vol. 6, No. 2, 143-146

An experiment control computer system time shared by several laboratories*

JOHN KNIGHT†, VICTOR COLBURN, DAVID OWENS,
LEE FREEMAN, DANIEL SYED, and WAYNE RASBAND
National Institute of Mental Health, Bethesda, Marylands 20014

An experiment control computer system, operational for more than 2 years, is discussed. The system is multiprogammed, using a vendor supplied real-time operating system. Individual experiments employ multitasking—fast response functions are implemented in core resident tasks while interactive and other slow response functions are implemented in tasks that operate under time sharing. The areas of psychological research currently supported are concept formation studies, EEG evoked response studies, monitoring the autonomic nervous system, perception studies, and family interaction studies.

Since January 1971 the computer system discussed has enjoyed a prominent role in support of intramural research programs of the NIMH. The system, which is built around a 32K word CPU and auxiliary disk storage, provides the fast response times typical of a dedicated single user system in a powerful multiprogrammed time-sharing environment. This configuration can be used concurrently to control and monitor a wide range of standard psychological experiments.

A general purpose computer with standard peripherals using vendor supplied software is at the heart of the system. The hardware is summarized in Table 1. The software includes a real-time operating system, language processors, and utilities.

Psychological protocols impose a range of differing requirements to be met by a supportive computer. The timeliness of required response varies from less than a millisecond for stimulus control and physiological data collection to several seconds for interactive parameter entry and subject instruction. Flexibility of input/output is required for the many types of equipment employed in psychological research. Finally, computational power extends from the capability to do simple computations to the more sophisticated complete experiment control. Heavily utilizing vendor supplied software, the system design accommodates these various requirements for multiple simultaneous users.

The system is multiprogrammed; this means more than one program can be run concurrently. If used as a purely time-sharing system, the computer could run perhaps 30 programs at once. The special demands of experiment control reduce the number of programs which can share the system by half. Time sharing is implemented by swapping the time-shared tasks from the desk for a time slice of execution on a round robin basis. The core-to-disk and disk-to-core swap necessary to interchange two tasks averages about 1/2 sec.

Varying response time requirements in the different phases of each experiment have led to extensive use of multitasking. The concept of multitasking is basic to the system design. Each experiment control program is reduced to a series of tasks; one task for each process necessary to control an experiment. For example, a task may control stimulus generation and data collection while another task prompts the operator for S identification and the specification of runtime parameters. Some processes require fast response and are allocated to core resident tasks. Other processes are less time dependent and can be controlled by tasks operating under time sharing. Functions which depend on fast response are not executed under time sharing. These functions are isolated and given special treatment. By allowing the associated task to remain in core while awaiting an interrupt, it has been possible to insure that

*This project is a joint effort by the Computer Systems Laboratory of the Division of Computer Research and Technology and the Technical Development Section of the National Institute of Mental Health in support of the intramural research program of the National Institutes of Health.

†Requests for reprints should be sent to John Knight, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland 20014.