To be published in
IEEE Trans. Nucl. Sci.

BNL# 21885

Presented at IEEE Nuclear Science
Symposium, 20-22 October 1976,
Braniff Place Hotel, New Orleans.

CONF-761006-14

A FILE MANAGEMENT FOR EXPERIMENT CONTROL PARAMETERS
WITHIN A DISTRIBUTED FUNCTION COMPUTER NETWORK*

F. W. Stubblefield

Brookhaven National Laboratory
Upton, New York 11973

October 1976

*Research carried out under the auspices of Energy Research and
Development Administration: Contract No. EY-76-C-02-0016.

MASTER

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

A FILE MANAGEMENT FOR EXPERIMENT CONTROL PARAMETERS
WITHIN A DISTRIBUTED FUNCTION COMPUTER NETWORK*

F. W. Stubblefield

Brookhaven National Laboratory
Upton, New York 11973

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Abstract

An attempt to design and implement a computer system for control of and data collection from a set of laboratory experiments reveals that many of the experiments in the set require an extensive collection of parameters for their control. The operation of the experiments can be greatly simplified if a means can be found for storing these parameters between experiments and automatically accessing them as they are required. A subsystem for managing files of such experiment control parameters is discussed.

Introduction

The main objective in the design of a functionally distributed computer system is to partition the system function into subfunctions residing at the next lower level in the hierarchy of functional complexity. Once this objective has been realized, the partitioning process is iterated in order to isolate subfunctions at still lower levels of functional complexity. Furthermore, this partitioning process must be carried out in such a manner that a functional complexity level is reached which contains a set of system subfunctions which have the following two properties:

- (1) the boundaries (i.e., the input to and output from the subfunctions) of all subfunctions in the set have approximately the same complexity level and an approximately uniform structure;
- (2) the subfunctions in the set must be separable into subsets which do not overlap each other with respect to the hardware structures and software structures which are required to implement the subfunctions.

When this functional level is reached, each isolated subfunction is confined to a node of the system, where a node contains all of the hardware and software required to implement the subfunction.

In general, the partitioning process can be iterated within a system node to isolate nodes at still lower functional levels. However, with currently available hardware and software, it is usually not economically feasible to implement more than the system level of nodes. This is the case in the present system and the system nodes will be referred to simply as nodes. The functional level of subfunctions performed by the nodes is called the node level.

Ideally, but not necessarily, the hardware and software confined to a node should not be used to implement any other node level function. The reduced (from the system level) functional complexity level of such nodes should engender similar reductions in the complexity of hardware and software required for their implementation. This reduction in complexity should make the individual nodes easier to implement, operate, and maintain. Thus a complete system constructed according to a functionally

distributed architecture should reflect the same advantages which accrue to the individual nodes. In addition, other advantages of the functionally distributed architecture have been identified.

If the computer system under consideration has as its system level function the monitoring and control of and data acquisition from a set of laboratory experiments, an attempt to isolate functions which must be provided at a level just below that of the system eventually discloses the following facts:

- (1) Most of the experiments can utilize access to two sets of numbers (parameters) to control the progress of the experiment. The first set of parameters is used to initialize the experiment apparatus, while the second describes the measurements to be made or the conditions which are to hold at various times in the course of the experiment. Several conclusions can be drawn about these sets of parameters.
- (2) The set of initialization parameters usually consists of $\sim 10^0$ to 10^2 numbers, while the set of measurement parameters can typically contain a factor of 10 times this many numbers.
- (3) The parameters do not have stringent requirements on their average access time. Access times of as much as 10^1 to 10^2 seconds could be allowed for the initialization parameters while access times of 10^0 to 10^1 seconds could be tolerated for the measurement parameters.
- (4) The format of both sets of parameters, initialization and measurement, is not simple. If the parameters are to be useful, a format for the individual numbers at least as complex as single-precision floating-point numbers must be supported.
- (5) The parameter sets are useless outside of the present system. Thus it is not necessary to provide for possible transfer of the sets to other computer systems.
- (6) A mechanism for grouping parameters into subsets and referencing the subsets by label (name) must be provided.

*Research carried out under the auspices of Energy Research and Development Administration: Contract No. EY-76-C-02-0016.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

- (7) Lastly, the types of experiments which are to be controlled have an effect on the way in which the parameter sets are managed. In many cases, two different experiments are similar enough that they require identical sets of initialization parameters. If the set of experiments is one which shares a common source of radiation, for example, every member of the set may utilize at least one common collection of initialization parameters. On the other hand, it is unlikely that two or more experiments can utilize identical sets of measurement parameters.

The above properties of the experiment control parameters suggest that these parameters should be stored as files on a random access device located at a central node which will supply the parameters upon demand to a set of experiment control nodes. The file management provided for the parameter files by the central node must allow for both single- and multi-accessor files with the experiment control nodes being limited to read-only access to the multi-accessor files. In what follows, the single-accessor files will be referred to as node files; the multi-accessor files will be called common files. Also, an additional node must be provided from which to create and update (modify) the parameters stored in the multi-accessor files.

In addition, one other aspect of experiment organization has an effect on the management of parameter files. In most cases where a set of experiments has a common source of some sort, the experiments are arranged in small groups of two or three experiments each and a group of experimenters is responsible for each experiment group. This organization has the result that the computer programs and hence the parameter files for different groups of experiments have widely varying formats, even though the experiments within the different groups may be quite similar. In addition, a group of experimenters usually wants to minimize the number of interactions within the system, accidental or allowed, with other experimenter groups. In particular, most groups want to restrict access to their parameter files to experimenters (i.e., experiments) within the group. These requirements lead to a third set of parameter files, the group files.

System Configuration

The system which contains the file management under discussion is the Reactor Experiment Control Facility. The Facility is used to control neutron scattering and x-ray diffraction experiments located at the High Flux Beam Reactor at Brookhaven National Laboratory. The system and its structure have been discussed in some detail elsewhere.^{1,2,3} Briefly, the system is a computer network which has a star-like configuration. At the points of the star are collections of hardware and software which are referred to as private nodes of the network. All of the private nodes save one are experiment control or application nodes; the one other private node serves as both a subsystem manipulation or control node and a program development node for developing application programs. At the center of the star is a shared service or common node. The private nodes are connected to the common node by high-speed, parallel-transfer communication links.

Collected at the shared service or common node are the high-quality peripheral devices which, for both economic and logistic reasons, must be shared among the application nodes. Included among these devices is a

disk to contain the files of experiment control parameters. The software routines which provide the management for these files take the form of a common node task level subsystem.^{4,5}

Experiments are controlled by high-level language (FORTRAN) programs executing at the application nodes. Initialization and measurement parameters required by these programs are requested from the common node. The high-level language mechanism for setting up and carrying out such requests is the FORTRAN direct access READ statement. Thus the application node routines required to support the parameter file subsystem include appropriate FORTRAN object-time system (OTS) routines capable of setting up transactions and application node monitor routines capable of driving the communication link between the application and common nodes.

The control node contains a set of batchstreams each of which controls the execution of a batch of jobs (programs). The jobs in each batch collectively perform some well-defined operation on the subsystem of parameter files at the common node. These operations change the status of the subsystem in a manner which can have an effect on the operation of more than one application node. Such operations include, for example, creation or destruction of common or group files.

The functional definition, implementation, and operation of each of these three types of nodes will be discussed in the context of the parameter files subsystem in the following sections.

Common Node Functional Definition

The function of the common node is to provide a set of services to the private nodes. The type of services provided is discussed in detail elsewhere.² All services are provided upon request, i.e., the private nodes must initiate the requests for service. The common node in no sense controls the operation of a private node.

Common Node Implementation

Service requests are submitted to the common node in the form of transactions. As discussed in more detail elsewhere,² common node processing for each transaction is carried out by two task sequences, one at each phase of the transaction. Routines within the common node monitor provide the function of processing the transaction to the point that the first task in the first sequence can be scheduled, processing the transaction between task sequences, and after completion of the second sequence. This function is not provided only for the parameter files subsystem, but is shared by all task level subsystems present at the common node. In addition to this transaction processing function, other functional elements are required to implement the subsystem. These include:

- (1) a subsystem directory for labelling the information maintained by the subsystem and for holding the status of the subsystem;
- (2) a random-access device to contain the files of experiment control parameters;
- (3) processor codes for updating the subsystem directory and transferring the parameters to/from the random-access device.

These functional elements of the subsystem implementation

must themselves be implemented using hardware and software elements. These elements are discussed below.

Implementation Hardware

Hardware elements used to implement a common node subsystem can be of two basic types: shared or local (dedicated). In the present case, all hardware elements are shared. Included are the following:

- (1) a controller for a large capacity disk pack drive;
- (2) a large capacity, moving-head disk pack drive;
- (3) the common node processor and its associated main memory.

Implementation Software

As in the case of the hardware, the software implementation elements can also be divided according to their type. The shared elements are those task service and lower level routines⁴ within the common node operating system which are required to support the subsystem tasks. The non-shared (local) elements constitute a task level subsystem and include the following:

- (1) a set of tasks which execute under the common node operating system;
- (2) a task level software resource which consists of a module of read-only access tables;
- (3) a task level software resource which contains a set of subsystem local subroutines.

Task Level Subsystem. As mentioned above, the above set of tasks and its associated software resources constitute a task level subsystem. Each subsystem at this level is known to the common node operating system by name. (The operating system maintains a table of task level subsystem names.) One of the properties attributed to a task level subsystem is the capability of having associated with it partitions of a partitioned device. Each task level subsystem may hold as many as two partitions of each partitioned device. These two partitions are referred to as the control partition and the data partition. The data partition contains only data, where data here means information which can be modified without any subsequent effect on the operation of the common node. Information stored in the control partition can affect common node operation, in particular, the status of the subsystem itself can be stored in this partition.

The present subsystem utilizes both a control partition and a data partition on the system random-access device (the large capacity disk mentioned above). Directories of the parameter files are stored in the control partition; the files themselves are stored in the data partition.

The storage associated with each partitioned device is considered to consist of a large number of 8-bit bytes. Common node subsystems at the task level have no knowledge of the physical location (starting byte number) of their associated partitions. These subsystems employ map offsets to compute the location of information relative to the start of a partition.

The operating system uses this relative position and its knowledge of the partition starting byte number to obtain the physical location of information transferred during an input/output (I/O) operation. The structure of the control and data partitions associated with the present subsystem is dependent upon the types of parameter files allowed.

Experiment Control Parameter Files

The structure adopted for the files of experiment control parameters is a reflection of the intrinsic properties of these parameters as outlined above. In particular, the total number of parameters required for a given experiment is known well in advance of the start of a run of the experiment. This fact is used to advantage by requiring the parameter files to have a fixed length. No extensions of a file are allowed after file definition (creation). Also, parameter files tend to have a long lifetime (many runs of an experiment). The files have a fairly uniform and short (10^2 - 10^3 bytes) length. These facts suggest that an elaborate file management scheme, i.e., a scheme for managing sequential files, is not required. In addition, the definition of the common node as a transaction processor and the properties of the transactions which it processes have an effect on the type of file management used.

As is noted elsewhere,² one of the properties of a transaction is that a time period of 10^{-2} - 10^0 seconds is allowed for its complete processing. All central processor unit instructions and peripheral device I/O operations required for the transaction must take place within this time period. The random access device used to contain the parameter files has a maximum access time of 83 milliseconds and a transfer rate of 7.5 microseconds per double byte. It is usually the case that a small fraction of the transaction time is spent executing instructions at the task level; such instructions are typically executed in less than one millisecond. The time required for the monitor to set up and provide services to the two task sequences possible for a transaction is much longer, typically 100-200 milliseconds. Included in this time is ~ 2 milliseconds to process each I/O operation request. Thus it is apparent that 3-4 I/O operations directed to the random access device can be contained within the transaction time. Once an allocation unit size has been chosen, this number of operations places an upper limit on the size of file which can be managed if a strictly sequential file management is used. (Here a sequential file is one in which each allocation unit of the file contains a pointer to the next unit of the file.) Since there is no advantage in employing a sequential scheme for managing files with the properties listed above, this type of management has been rejected. The limited number of I/O operations which can be performed within a transaction time is also an argument against file management schemes in which file allocation information is maintained separately from the file descriptive information (i.e., bit maps cannot be employed, the file allocation information must be maintained in the file directory).

The parameter files each consist of a number of contiguous bytes of storage on the random access device. The number of bytes (file length) must be an integral multiple of the allocation unit size (512(10) bytes) and is established at the time the file is created. The length information is maintained in the file directory. Parameter files are accessed by block number. Each transaction results in the transmission of one block of information. The block size must be an integral multiple of 512(10) bytes, and must also be defined in such a way that the

file contains an integral number of blocks. A maximum block size limit of 16,384 bytes is also imposed. The default block size is 512(10) bytes. In practice, blocks having the default size have been found to be satisfactory. The block size is stored along with the file name in the file directory. Provision has been made for modifying the block size of a file at a time subsequent to file creation, but this feature is very rarely used.

Justification for defining three different types of parameter files has been given above. Accordingly, both partitions belonging to the subsystem are divided into three subpartitions. These divisions are made at the task level; the operating system has no knowledge of the number, location, and size of the subpartitions. The subpartition parameters are kept in the subsystem read-only access tables resource. The contents of the subpartitions for each type of file are discussed in more detail below.

Common Files. All common files have names which begin with the letter "C". The first control subpartition contains a single directory of common files. The first data subpartition contains the parameters which constitute the common files.

An entry in the common file directory is shown in Fig. 1. The entry includes the file name, length (number of bytes), starting byte number (relative to the start of the data subpartition), and block size. Additional items of descriptive information such as the date and time of creation are also included.

The common files can be written (modified) only as a result of processing for transactions originating at the control node. Tasks which manage the common files access the logical node number of the private node which initiated the transaction request and checks this number against the logical number of the control node.

Although the common files may be written by only one mode, any node can access these files in read-only fashion. Furthermore, this access can be granted to more than one node at a time. As will be shown below, one mode of reading a common file requires that the common node maintain the number of the next block to be read. For this reason, directory entries for the number of readers and the next block to be read for each reader are maintained. Entry space sufficient to accommodate up to 32(10) readers is allocated.

Common files may be destroyed only by the control node and, furthermore, a "destroy common file" transaction is processed to completion only if the file to be destroyed is not under access by any node.

Group Files. Group files are similar to common files in that they are written only by the control node and may be accessed in read-only fashion by several nodes simultaneously. However, the reader nodes must all be members of an experiment group.

The control and data subpartitions for the group files are further divided into third-level partitions for each group. Each control third-level partition contains a directory of files belonging to a group. All group file names must have initial character "G". As in the case of common files, the directory must contain multiple entries for readers of each file and the next block to be read. Space is provided for up to 16(10) simultaneous readers (each group may contain this number of members).

Directory entry status	CDDEST (000)
Filename (six ASCII characters)	CDNAME (002)
Filename extension (three ASCII characters)	CDNESP (010)
Separator character	
File length (no. of bytes)	CDFLTH (014)
Date of file generation in form DD-MMM-YYb	CDGNDT (020)
Time of file generation in form HH:MMb	CDGNIM (032)
Offset from start of subpartition to start of file	CDSTOF (040)
Offset from start of subpartition to end of file	CDEDOF (044)
Date of last file modification in form DD-MMM-YYb	CDLMDT (050)
Time of last file modification in form HH:MMb	CDLMTM (062)
Logical block size (no. of bytes)	CDLBSZ (070)
Maximum logical block no.	CDMXBN (072)
Status	CDSTAT (074)
No. of next logical block to be written	CDNXBW (076)
No. of next logical block to be copied	CDNLBC (100)
No. 'open read next' No. 'open read'	CDNORD (102)
Node identification bits for nodes holding file 'open for read'	CDIDRD (104)
Node identification bits for nodes holding file 'open for read next'	CDIDRN (110)
No. of next logical block to be read, logical node no. 00	CDNXBR (200)
No. of next logical block to be read, logical node no. 01	
No. of next logical block to be read, logical node no. 36(8)	
No. of next logical block to be read, logical node no. 37(8)	

Fig. 1 Common Files Directory Entry
(With Map Offset Logical Names and Physical Values)

Group files are destroyed by the control node. Destruction is possible only when no node is reading or writing the file.

Node Files. Each application node has available to it a set of parameter files which can be created, modified, read, and destroyed only by the node itself. These files are the node files. Names of all node files begin with the letter "N".

The control and data subpartitions for node files are both divided into one third-level partition for each node. The control partition contains the node directory; the data partition contains the node files. No directory entries are required for multiple readers of a file since these files can be accessed only by the node in question.

It is important to emphasize that the allocation of node file space to the various application nodes is made when the subsystem tasks and resources are built (linked to logical memory space locations and loaded to physical memory space). This allocation is by no means dynamic; a long and tedious process is involved in changing the allocation. The allocation of space among nodes (and among node files, group files, and common files) is modified only at time intervals of at least a few months. The allocation scheme adopted is a reflection of the assumption that routines which control real-time experiments need to have a guaranteed amount of file space available to them even if this space has to be somewhat limited. File management schemes which allow two or more nodes to "share" a portion of the available file space also allow one node to claim so much space that a second node must abort its experiment (control routine). Here the position is taken that this latter possibility must be avoided.

File Management Functional Definition

As mentioned above and elsewhere,² the common node is set up as a transaction processor. As such, the node performs functions (provides services) at a complexity level just below that of the system itself. These functions are performed in response to service requests initiated by the control node and the application nodes. The structure of the private node routines which set up and carry out these requests will be discussed below. The response to the request at the common node consists of the execution of two task sequences. The functions provided by the sequences which manage the parameter files are discussed below.

Control Level Functions

All of the functions provided by the task level subsystem currently under discussion reside at the same complexity level, i.e., the functions are performed in response to a service request. However, two different structural levels can be distinguished. One set of functions consists of those whose performance can have a subsequent effect on the parameter information accessed by more than one application node. Functions at this level are referred to as control level functions. Functions at the second level modify the status or content of information which is accessed by only one application node. This second structural level is termed the service level. The control level functions are listed below with a short description of each function.

INITIALIZE. The INITIALIZE function sets to an initial state (zeroes) all subsystem directories. All common, group, and node file directories are set to

contain no files.

Common File Functions. The following control level functions have an effect on the status and content of the common files only.

CREATE Common File. The control node must submit a six-character name (beginning with "C") for the file and the number of 8-bit bytes which the file is to contain. If this amount of space is available in the data partition, the space is claimed and the file name is entered into the directory of common files. An additional restriction is that the submitted file name must not already be present in the directory. The number of bytes allocated is always an integral multiple of 512(10). Entries for the file length and starting byte number relative to the start of the subpartition are made in the directory. The actual space allocation process consists of making these entries in the directory.

DESTROY Common File. The directory of common files is searched for the file name submitted by the control node. If found, the file name is deleted from the directory, thereby deallocating the space occupied by the file. No deletion takes place if the file status indicates that it is in the process of being written by the control node or read by any of the application nodes.

OPEN Common File FOR WRITE. The named common file is opened for write by setting an appropriate status bit in its directory entry. The file must not be currently under read access by any application node (see service level functions, below). In this mode of writing, the common node does not keep track of the file block to be written. The block number is submitted along with the other transaction request information by the control node.

CLOSE Common File AGAINST WRITE. The 'open for write' status bit in the directory entry for the common file in question is cleared. (The file must have 'open for write' status at the time of the request.)

OPEN Common File FOR WRITE NEXT. This function is similar to the OPEN FOR WRITE function. The 'open for write next' status bit is set and the number of the next block to be written to the file is initialized to zero (first block in file). The file must not be open for write or read if this function is to be successfully completed.

CLOSE Common File AGAINST WRITE NEXT. The status bit signifying 'open for write next' is cleared for the file specified by the submitted name. The number of the next block to be written is set to -1 (non-existent).

WRITE Common File BLOCK. If the control node is the transaction requestor, a block of parameter information is written to the specified file. The file must have 'open for write' status. In this mode of writing, the control node must specify the number of the block to be written.

WRITE NEXT Common File BLOCK. This function is similar to the WRITE BLOCK function except that the number of the next block to be written is extracted from the file directory.

Group File Functions. There is a control level function operating on group files corresponding to each of the common file functions listed above. The initial task in each control level sequence obtains the logical

number of the requesting private node from the common node operating system and checks that the requestor is the control node. Other processing by the control level sequence is also similar to that performed for the common files. However, requests for these functions must include a system global group number in the parameters describing the request. Tasks in the parameter files subsystem which perform these functions convert this number to a subsystem local group number. This conversion is performed using a look-up table contained in the subsystem read-only access tables resource. The local group number points to the correct directory and file partitions to be used.

Node File Functions. There are no control level functions which operate on individual node level files. The only control level function which affects the node files is INITIALIZE.

Service Level Functions

Service level functions may be requested by any application node. As mentioned above, these functions affect the status of a file only as it pertains to one application node. Tasks at this level obtain the logical number of the requesting node via a service request to the common node operating system.

Common File Functions. The application nodes may access common files in read-only fashion.

OPEN Common File FOR READ. If the specified common file exists and is not open for either type of write, it is opened for reading by the requesting node. An appropriate status bit is set in the file directory entry. This status bit specifies that the file is 'open for read' and indicates the logical node holding this status. In this mode of reading, the requesting node must specify the file block to be read.

CLOSE Common File AGAINST READ. The specified common file is removed from 'open for read' status for the logical node in question. The node cannot read the file until it is again opened for read.

OPEN Common File FOR READ NEXT. The 'open for read next' status bit corresponding to the requesting node is set in the appropriate directory entry. The "next block to be read" entry in the directory is set to zero, the first block in the file. On a subsequent READ NEXT transaction request from the node, the first block of the file will be transmitted and the "next block" entry updated.

CLOSE Common File AGAINST READ NEXT. The indicated file, if currently 'open for read next', is removed from this status. The "next block to be read" entry is set to -1, non-existent block.

READ Common File BLOCK. A specification for the number of the block to be read must be included with this request. If the file has 'open for read' status, the block is transmitted to the requesting node.

READ NEXT Common File BLOCK. Tasks which perform this function extract the number of the next block to be read from the common files directory entry for the named file. This block is transmitted to the requesting node and the "next block" entry is incremented.

Group File Functions. A group file function at the service level which corresponds to each of the above service level common file functions is present within

the management. Tasks which perform these functions obtain the system global group number of the requesting node from the common node operating system. The system global group number is converted to a local number as in the case of the group file functions at the control level. This local number furnishes a pointer to the correct group file directory and file partitions.

Node File Functions. All functions which have an effect on the node files (except INITIALIZE) reside at the service structural level. The set of functions includes not only a function analogous to each common or group file service level function, but also a function analogous to each control level function for these files. Tasks which service these requests obtain the logical number of the requesting node from the operating system. This number is used as an index into tables within the subsystem read-only access tables resource to obtain the relative start of the directory and file partitions (third-level) for the requesting node. Directory entries for node files can be smaller in size since no multi-accessor information need be retained. Otherwise processing is similar to that for the common and group file functions.

Control Node Functional Definition

A need for a control node arises from the definition of the common node as a transaction processor. The only method of eliciting a function from the common node is to submit to it a service request which obeys the transaction protocol. There is no other method of interaction. On the other hand, however, the common node is a shared service node, where the sharing of services is among the application nodes. If there exist functions to be performed by the common node which change its status with respect to two or more application nodes, these functions must be requested by a structural entity in the system which resides at a higher structural level than the application nodes. Since the only method of interacting with the node is by means of a transaction, this entity must be a private node at a structural level above the service level. The private node is labelled the control private node; its structural level in the system is referred to as the control level.

As a subsidiary effect, establishment of a control node also alleviates another serious problem at the common node. For reasons detailed elsewhere,^{2,4} the transaction processing routines are implemented as common node task level subsystems. These subsystems must be initialized at the time that the common node is first started up. Initialization in this manner would require that at least one task belonging to each subsystem be executed just after the operating system routines have been initialized. Execution of tasks in this manner would require additional operating system routines of a complex nature and, furthermore, such execution would be a violation of the common node definition. In addition, there are both theoretical and practical reasons for separating initialization of the task level subsystems from initialization of the operating system itself (service request level subsystems) and for providing for selective initialization of the task level subsystems:

- (1) The common node must be shut down at regular intervals for routine maintenance. The easiest way to restart the node is to initialize the operating system while leaving the information at the task level undisturbed.
- (2) The common node fails upon infrequent

occasion. Again, the easiest way to restart the node after failure diagnosis is to initialize only the operating system.

- (3) As new task level subsystems are added to the common node, it is most convenient to initialize only the additional subsystem.

Defining a control node as a functional element of the system and endowing it with the capability of requesting an INITIALIZE function solves all of the above problems.

Control Node Implementation

The control node is implemented using a commercial⁶ single-user operating system capable of executing a batchstream (file of statements in a job control language). In addition to the processor and its associated memory, the peripherals necessary to support this operating system are collected at the node. Also present are the peripherals necessary to manipulate the common and group files (and provide similar support for other subsystems).

The batchstream execution technique provides a very simple means for the system operator to invoke a control level function. A batchstream is called into operation by typing its six-character (or less) file name at the control node keyboard. Once this initial entry has been made, the programs necessary to set up and request the desired function are executed automatically. All additional information required from the operator is solicited by the node and is accompanied by a generous amount of prompting information.

The structure of control node routines which constitute a batchstream execution is illustrated in Fig. 2. All batchstreams call a high-level language (FORTRAN) program which obtains the parameters of the request, such as the filename, via I/O requests to the keyboard. Standard FORTRAN-callable subroutines are used to load the transaction information to a transaction parameter block contained in main memory. An additional FORTRAN subroutine is used to load the appropriate function code.² One of these additional subroutines exists for each function which can be requested. The subroutines in turn utilize software interrupts to perform the actual loading. The last subroutine and software interrupt invoke the communication link driver. The driver has the processor execute a wait loop until the transmissions corresponding to a transaction have been completed. Upon its return to execution, the main program may utilize standard FORTRAN subroutines to check for errors and report the error parameters at the keyboard. The function request ends on exit from the batchstream.

Application Node Functional Definition

The function of an application node is to execute a high-level language program which monitors and controls and acquires data from an experiment. In the process of performing this function, the node may request services from the common node.

Application Node Implementation

The application nodes are implemented using a single-user, core-resident, real-time operating system⁷ which is capable of executing an overlaid FORTRAN program. The routines which are necessary to implement

the management for parameter files are of two types:

- (1) FORTRAN-callable subroutines are used to request all functions except READ, READ NEXT, WRITE, and WRITE NEXT.
- (2) FORTRAN direct access I/O statements "READ(u'r) list" and "WRITE(u'r) list" are used to request READ, READ NEXT, WRITE, and WRITE NEXT functions.

The structure of routines which support an application node FORTRAN program call to a file management subroutine is illustrated in Fig. 3. The FORTRAN compiler converts the FORTRAN call into a reference to an assembly language subroutine. Such assembly-language-coded subroutines are referred to as FORTRAN system subroutines and are maintained in a standard library for linking to the compiler generated assembly language code. The FORTRAN system subroutines utilize system subroutines at a lower level to set transaction parameters into a transaction parameter block and call the communication link driver. These lower level system routines establish the interface between the experiment control level and the application node operating system level. The interface mechanism is a software interrupt. The operating system executes a wait loop until the transaction request to the common node has been completed. Additional system level subroutines can be employed to check for transaction errors and set the function success flag to be returned to the FORTRAN level program.

The FORTRAN compiler converts direct access READ and WRITE statements into a series of calls to assembly language subroutines⁸ contained within a collection of support routines supplied by the compiler manufacturer. These routines contain imbedded software interrupts into an operating system⁶ also supplied by the compiler manufacturer. The compiler itself cannot be modified since it is used to generate code to be used at other nodes which utilize the commercial operating system (and it is not considered feasible to modify the compiler and maintain two versions). Thus routines in the support package must be modified or removed or routines to support the imbedded software interrupts must be added to the application node operating system. The second technique has been adopted. One consequence of not modifying the support package is that a file block size of 512(10) 8-bit bytes must be used. This buffer size is assumed by many of the support routines. This is the reason that this size has been chosen for the default size and, in practice, no blocks of a different size have been used.

The manner in which the number of the file block to be transmitted is set deserves some discussion. The status of a FORTRAN logical unit* is set to 'open for read(write)' or 'open for read(write) next' when the logical unit is opened. When a READ or WRITE statement is executed, the block number is assumed to be contained in the statement block number variable if the unit has 'open for read(write)' status. If not, the common node supplies the block number. The READ/WRITE mode to be used is dependent on the way in which the parameter file is to be scanned. If the blocks are to be scanned sequentially, the common node is able to keep track of the next block and does so because this node is more reliable than the application nodes. If true random access to the parameters is required, only the application node program can decide on the next block to be read.

*The FORTRAN statement "READ(u'r)" means read file block number r from logical unit number u.

Fig. 2. Structure of Control Node Batchstream Procedures

Fig. 3 Structure of Application Node FORTRAN Support Routines

Upon execution of a READ or WRITE statement, one of the imbedded software interrupts moves execution control to a service routine within the application node operating system which in turn calls the communication link driver. At completion of the transaction request, a check for transaction error is made. If none has occurred, the input list is loaded (in the case of READ) and control returns to the FORTRAN main program (or overlay). The operating system goes to a standard error loop upon transaction error.

Acknowledgements

The common node tasks were written by M. A. Kelley. R. K. McMullan provided useful information on the types of parameter files which would be most useful to the users of the system. P. D. Mansfield typed the source code and maintained the object libraries required for the parameter files subsystem. B. D. Gaer typed the draft and final manuscript for this article.

References

1. D. G. Dimmler; IEEE Trans. Nucl. Sci., Vol. NS-21, No. 1, (February 1974).
2. F. W. Stubblefield and D. G. Dimmler; IEEE Trans. Nucl. Sci., Vol. NS-22, No. 1, (February 1975).
3. D. G. Dimmler, N. Greenlaw, M. A. Kelley, D. W. Potter, S. Rankowitz, and F. W. Stubblefield; IEEE Trans. Nucl. Sci., Vol. NS-23, No. 1, (February 1976).
4. F. W. Stubblefield and D. G. Dimmler; IEEE Trans. Nucl. Sci., Vol. NS-23, No. 1, (February 1976).
5. F. W. Stubblefield; IEEE Trans. Nucl. Sci., Vol. NS-23, No. 1, (February 1976).
6. Disk Operating System (DOS), Version 9, September 1973, Digital Equipment Corporation, Maynard, Massachusetts.
7. D. G. Dimmler; "Network Specific Software Structures in a Private Node of a Distributed Function Laboratory Computer Network." (In preparation)
8. FORTRAN Object Time System (OTS), Version 22, September 1973, Digital Equipment Corporation, Maynard, Massachusetts.