Presented at 1978 Nuclear Science Symposium,
18-20 October 1978, Shoreham-Americana Hotel,
Washington, D. C.

AN APPLICATION NODE SYSTEM IMAGE MANAGER SUBSYSTEM
WITHIN A DISTRIBUTED FUNCTION LABORATORY COMPUTER SYSTEM*

F. W. Stubblefield and R. D. Beck[†]

Brookhaven National Laboratory
Upton, New York 11973

October 1978

*This research was supported by the U. S. Department of Energy:
Contract No. EY-76-C-02-0016.

[†]Present address: INTEL, Corp., Sunnyvale, Ca. 94086.

MASTER

1000

1000

AN APPLICATION NODE SYSTEM IMAGE MANAGER SUBSYSTEM
WITHIN A DISTRIBUTED FUNCTION LABORATORY COMPUTER SYSTEM

F. W. Stubbsfield and R. D. Beck

Brookhaven National Laboratory
Upton, New York 11973

Abstract

A computer system to control and acquire data from five neutron scattering, four neutron diffraction, and one x-ray diffraction experiments located at the Brookhaven National Laboratory High Flux Beam Reactor has operated in a routine manner for over three years. The computer system is configured as a network of computer processors with the processor interconnections assuming a star-like structure. At the points of the star are the ten experiment control-data acquisition computers, referred to as application nodes. At the center of the star is a shared service node which supplies a set of shared services utilized by all of the application nodes. A program development node occupies one additional point of the star. The design and implementation of a network subsystem to support development and execution of operating systems for the application nodes is described.

Introduction

A computer system¹ to control and monitor and acquire data from nine neutron and x-ray scattering and diffraction experiments located at the High Flux Beam Reactor at Brookhaven National Laboratory has operated in a routine manner for over three years. The system has been constructed according to a distributed function architecture.² Briefly, the system function, to control and monitor and acquire data from a set of laboratory experiments, is partitioned into the two functions "develop programs for experiment control and data acquisition" and "perform operations to control and acquire data from a set of laboratory experiments."

The first function is confined to a node of the system termed the program development node. The second function cannot be assigned to a system node as it stands and must be partitioned further. These first two partitioning operations are illustrated in Fig. 1.

The performance of operations to directly control and acquire data from a set of laboratory experiments is partitioned into the two functions "provide a set of shared services required to control and acquire data from a set of laboratory experiments" and "execute programs to control and acquire data from a set of laboratory experiments". The function of providing a set of shared services is confined to a shared service or common node. Additional partitioning of the shared service function is shown in Fig. 2.

The program execution function is partitioned (Fig. 3) into the n functions "execute program to control and acquire data from laboratory experiment x ", where n is the total number of laboratory experiments. These functions are confined to n program execution or application nodes. The system level functions which finally result from this partitioning process and their

confinement to nodes of the system are illustrated in Fig. 4. For completeness, an illustration of further partitioning of the program execution function at a typical application node is included as Fig. 5.

The primary reason for partitioning the system function in this particular manner is that the major parts (subfunctions) of a subsystem to support development and execution of high-level (FORTRAN) programs for experiment control and data acquisition can be confined to different nodes of the system. Both the partitioning method and the advantages held by a system partitioned along such lines have been discussed elsewhere,²⁻⁵ as have the design and implementation of the FORTRAN programming subsystem.⁶ Once such a programming subsystem has been realized, it is straightforward to recognize that a subsystem for managing application node operating system images can be defined and implemented in a very similar manner.

Application Node Operating Systems

Partitioning of the system function into program development and program execution has the direct result that the application node operating systems are not required to support program development. Thus the operating system at each of the application nodes has only to support FORTRAN program execution. In practice it has been possible to satisfy these program execution requirements with operating systems all of which have the following properties:

- (1) all application node operating systems have the same limits in logical (and physical) address space;
- (2) all application node operating systems are completely core resident (in the sense that no part of an operating system is swapped into and out of core memory in the process of operating system execution);
- (3) operating system images have a maximum length of 8,192 16-bit words of code;
- (4) a response time of $10^0 - 10^2$ seconds for loading an operating system image to an application node is tolerable.

These properties are a less stringent subset of the properties of logical program units within the FORTRAN main program and overlay manager subsystem⁶ and suggest that a similar subsystem can be established for managing application node operating system images. A description of the implementation of such a subsystem at each of the system nodes follows.

Program Development Node

The program development node contains the software processors and hardware peripheral devices required for the complete program development process. A character-string-oriented editing program⁷ is used to generate and

*This research was supported by the U. S. Department of Energy: Contract No. EY-76-C-02-0016.

*Present address: INTEL Corp., Sunnyvale, Ca. 94086.

modify source code for the application node operating system. Alternatively, source code may be created on 80-column punched cards and entered into the processing stream via a punched card reader. The medium for permanent storage of source code is DECTape,⁸ a small-capacity magnetic tape used in random-access fashion.

All source code is written in assembly language; the code is reduced to relocatable object form by a macro assembler. Object modules are placed into libraries by a librarian software processor. The object module libraries are also permanently stored on DECTape.

A core image of an application node operating system is produced by linking together a set of object modules appropriate to the node in question and assigning them to places in the logical address space of the node. This assignment must be consistent with the image properties listed above. The logical space layout of an application node operating system and the FORTRAN main program and overlay which it supports is shown in Fig. 6.

The generation and filing of object modules and the linking together of object modules from their libraries are both under control of batchstreams of control language commands. The output of the linking operation is a file which contains blocks of executable code along with a load address for each block. This file may be further processed in one of two ways:

- (1) the file may be punched on paper tape and loaded to its application node via a paper tape reader;
- (2) the file may be reduced to core image form, transmitted to the shared service node, and stored on a mass storage device.

Common Node

The common or shared service node contains a Digital Equipment Corporation PDP-11/45 computer, a set of high quality peripherals, and an operating system and collection of tasks to oversee the sharing of these peripherals among the application nodes. The peripherals, the mechanism by which they are shared, and the software which implements the sharing mechanism have been discussed elsewhere in great detail.^{3-6,9-11} Only a brief review will be given here.

The common node has been implemented as a transaction processor. Requests for service directed to this node and the subsequent response to these requests take the form of a transaction, where a transaction consists of two mandatory and one optional transmissions over the communication link between the common node and the application node requesting the service. Transactions entering the common node have some very definite properties^{3,5,10} of which two should be stated here:

- (1) the elapsed time required for complete processing of any transaction is 10^{-2} - 10^0 seconds;
- (2) all transactions have approximately equal weight with respect to their response time requirements.

Each transaction is processed by two task sequences. The first sequence accesses information in the REQUEST transaction parameter block and assembles an ACKNOWLEDGE transaction parameter block for transmission back to the requesting application node. If the optional transaction

DATABLOCK transmission is required, the second task sequence finishes the transaction processing after this transmission has taken place.

While processing a transaction, each task in these sequences may access common node resources.⁹ Statically allocated resources consist of modules of subroutines, read-only access tables, and modification access tables which are associated with the task level subsystems. Logical resources are structured flags which represent access, both read-only and modification, to various information structures within the common node. Physical resources are contiguous blocks of main memory which can be used as control blocks for executing various input/output operations or as buffers for these operations.

The overall scheme for processing transactions in the common node is the following:

All resources, both logical and physical, required to completely process a transaction are claimed prior to the start of transaction processing.

In this way the lockout problem due to conflicting resource requirements of two simultaneously processed transactions is avoided.

Application Node Operating System Image Manager Subsystem

The task level subsystem which oversees storage and retrieval of core images of application node operating systems is the application node operating system image manager subsystem. The tasks in this subsystem and the transaction functions which they perform are listed in Table I.

Images of operating systems are maintained in a data partition of 20-megaword common node disk. An important difference between the present subsystem and the FORTRAN main program and overlay manager subsystem⁶ is that each application node requires only one or two operating system images as opposed to the potentially indefinite number of FORTRAN main program and overlay images which might have to be accessed. Thus a portion of the data partition can be dedicated to each application node and no bit map for allocation of data partition space is required. In the present implementation of the operating system image manager subsystem, disk space for two images per application node has been reserved. Typically, one image contains the working copy of an operating system while the second image contains an experimental version which may undergo modifications at any time.

Each image space contains 32,768 16-bit words, an amount of address space which corresponds to the maximum which can be addressed by one mode of the PDP-11 processor. Each image has been assigned this amount of space for two reasons:

- (1) it is possible that the application node processors may be expanded to operation in two modes, kernel and user, of logical address space. If this is the case, the application node operating system will occupy the kernel mode and may expand to a maximum of 32,768 words;
- (2) it is necessary to operate the application nodes in stand-alone mode from time to time. In this case it

is convenient to automatically load not only the operating system but also a standard main program and one standard overlay (see Fig. 6).

Since it is assumed that the system will eventually expand to twenty application nodes, the subsystem data partition occupies 1,310,720 words of the common node disk.

A directory of image status and label information is contained in a control partition of the common node disk. For filing purposes, each image is considered to consist of eight sections of 4,096 words each. Images are transferred between nodes one section at a time. Thus an image of an application node operating system consists of two sections, numbers 0 and 1. Status information includes the date and time of image creation. The normal method of creating or modifying an image is to update the image on a section-by-section basis. Each section is read from the common node to the program development node, modified while in a program development node buffer, and written back to the common node. Hence a status bit which shows that a section is currently in the process of being updated is required.

With this structure for the subsystem directory and the operating system images, transactions directed to the subsystem tasks require the following resources for their complete processing:

- (1) a logical resource which represents access to both the subsystem directory and the images which the directory describes. At its first level, the resource represents access to the entire directory and all images in the subsystem. At the next lower level, the logical resource represents access to the images for only one application node and only the part of the directory which describes these images. The logical resource may be claimed for either modification or read-only access;
- (2) a physical resource of length 4,096 words. This resource is used as an I/O buffer for transfers of the subsystem directory from/to the common node disk;
- (3) a physical resource of length 4,096 words. This resource is used as a transaction datablock and contains the image sections being transferred to or from the common node disk or to or from the program development node or to an application node.

Control/Manipulation Node

It has been shown before^{4,6,11} that one additional node is required to complete the system. The control/manipulation node is used to initialize and manipulate application-node-global information maintained by the system. (Application-node-global information is control information which can have an effect on more than one application node if it is changed.) It has also been shown that this logical system node usually must be physically implemented with the program development node hardware and software, and such is the case in the present system. The program development node is used to direct control level requests to the application node operating system image manager subsystem. The system operator uses the control/manipulation node to INITIALIZE the subsystem and to obtain and list a copy of the

subsystem directory (the GET DIRECTORY function).

Application Node

As mentioned above, an operating system image may be loaded to an application node in one of two ways. The first method is to load the image from paper tape by means of the node paper tape reader. The loader program¹² for reading paper tape load modules occupies 104₁₀ words at the high address end of application node main memory (see Fig. 6). The user has to place the paper tape in the reader, set the program counter register to the start address of the loader program (by entering an octal number into a switch register), specify execution parameters by setting appropriate switch register bits, and start the application node processor. There are two difficulties with this method:

- (1) the paper tape (fan-fold) for an 8,192 word operating system is bulky and somewhat difficult to handle;
- (2) the paper tape requires approximately one minute to be read.

Alternatively, the operating system can be loaded by transmitting an image from the common node to the application node over the communication link between the two nodes. In this case, the user is required to place a short bootstrap paper tape in the reader, set the program counter register to the start address of the resident loader, specify execution parameters via switch register bits and start the processor. The bootstrap program controls transfer of the two image sections into application node memory and begins operating system execution. Loading of an operating system in this manner requires less than one second.

Conclusions

The image manager subsystem makes it possible for an application node to be initialized with its operating system in as little as a few seconds, depending on the expertise of the user. However, the main advantage afforded by the subsystem has to do with improvements in system logistics. Once the system has expanded to include ten or more application nodes, each of which may have a slightly different hardware/software configuration, the problem of keeping track of a large number of operating system paper tape modules begins to be very difficult. The only way to solve this problem is to maintain all images in one place and have an automated process for image filing and loading. This solution is provided by the present subsystem.

Future Work

The bootstrap program for loading operating system images over a communication link will be placed into read-only memory in a hardware bootstrap device. Once this is done, the user will no longer have to be proficient in handling paper tapes and their readers.

References

1. D. G. Dimmler, N. Greenlaw, M. A. Kelley, D. W. Potter, S. Rankowitz, and F. W. Stubblefield; The Brookhaven Reactor Experiment Control Facility - A Distributed Function Computer Network -. IEEE Trans. Nucl. Sci., NS-23, 398 (Feb. 1976).
2. D. G. Dimmler; Functional Distribution - An Architecture for Multi-User Computer Networks in Instrumentation. IEEE Trans. Nucl. Sci., NS-21, 838 (Feb. 1974).

3. F. W. Stubblefield and D. G. Dimmler; Transaction Processing in the Common Node of a Distributed Function Laboratory Computer System. IEEE Trans. Nucl. Sci., NS-22, 473 (Feb. 1975).
4. F. W. Stubblefield; A File Management for Experiment Control Parameters within a Distributed Function Laboratory Computer Network. IEEE Trans. Nucl. Sci., NS-24, 460 (Feb. 1977).
5. F. W. Stubblefield; Conversion of an Operating System Oriented Towards Transaction Processing from Two to Three Modes of Logical Address Space. IEEE Trans. Nucl. Sci., NS-25, 231 (Feb. 1978).
6. F. W. Stubblefield; A Main Program and Overlay Manager Subsystem within a Distributed Function Laboratory Computer System. IEEE Trans. Nucl. Sci., NS-25, 217 (Feb. 1978).
7. The software processors which execute at the program development node are EDIT-11, MACRO-11, LIBRARIAN-11, and LINK-11. They all execute under the Disk Operating System (DOS), Version 9, September 1973, Digital Equipment Corporation, Maynard, Massachusetts.
8. DECTape is a trademark of Digital Equipment Corporation.
9. F. W. Stubblefield; Logical and Physical Resource Management in the Common Node of a Distributed Function Laboratory Computer Network. IEEE Trans. Nucl. Sci., NS-23, 406 (Feb. 1976).
10. F. W. Stubblefield and D. G. Dimmler; A Task Scheduler and Service Subsystem for the Common Node of a Distributed Function Laboratory Computer Network. IEEE Trans. Nucl. Sci., NS-23, 413 (Feb. 1976).
11. F. W. Stubblefield; Continuous Sharing of a Record-Oriented Output Device within a Distributed Function Laboratory Computer Network. IEEE Trans. Nucl. Sci., NS-23, 423 (Feb. 1976).
12. This program is the DEC standard Absolute Loader.

<u>Transaction Function</u>	<u>Requesting Private Node</u>	<u>Common Node Task Name (Request Phase Sequence)</u>	<u>Common Node Task Name (Acknowledge Phase Sequence)</u>
INITIALIZE (SUBSYSTEM)	Control/Manipulation	TKPII2	-
GET (SUBSYSTEM) DIRECTORY	Control/Manipulation	TKPIDR	-
CREATE IMAGE	Program Development	TKPICR	-
DESTROY IMAGE	Program Development	TKPIDS	-
OPEN IMAGE	Program Development	TKPIOP	-
CLOSE IMAGE	Program Development	TKPICL	-
READ IMAGE SECTION	Program Development	TKPIRR	TKPIRA
WRITE IMAGE SECTION	Program Development	TKPIWR	TKPIWA
GET IMAGE	Application	TKPIIR	TKPIIA
GET IMAGE SECTION	Application	TKPISR	TKPISA

TABLE I. Application Node Operating System Image Manager Subsystem Transaction Functions

Fig. 1 First Two Levels of System Function Partitioning

Fig. 2 Partitioning of the Shared Services Function

Fig. 3 Partitioning of the Program Execution Function

Fig. 4 Confinement of System Level Functions to System Nodes

Fig. 5 Functional Partitioning Within an Application Node

Fig. 6 LAYOUT OF APPLICATION NODE
LOGICAL (AND PHYSICAL)
ADDRESS SPACE